Parallel Programming
Homework 1 Report

Student ID: 105062635
Name: 吳浩寧

1. IMPLEMENTATION

1.) Basic Version

a. 分組方式：

使用 epn()計算每個 process 要分配多少個數，為了減少訊息交換次數，盡量讓每個 process 上的數字量都是偶數，比如 20 個數分配給 4 個 process，由於 20/4=5，取最接近的偶數 4 或 6，可能分配方式如下(4,4,4,8)、(6,6,6,2)，選取最大值較小者，即為(6,6,6,2)。

b. 計算步驟：

每一輪中各個 process 內分別進行一次 Even-phase 和 Odd-phase，由於進行 Odd-phase 時的第一和最後一個數會落單，因此將每個 process 最後一個數傳至下個 process，與第一個元素做比較，再將較小者傳回。

c. 特殊情況：

若輸入的資料量較小，小於 process 數*2 時，就從第一個 process 開始，每個 process 分配給兩個數。

d. 終止情況：

每輪計算結束會判斷一次，若該輪有進行 process 中，或 process 間的數字交換，即會繼續進行計算。

2.) Advanced Version

a. 分組方式：由於這次並非在每個 process 內進行 odd-even sort，因此只須平均分配數字到每個 process 即可。

b. 計算步驟：

首先利用 qsort()將每個 process 進行排序，接著進行 process 間的 odd-even sort，使用 ratio 控制每階段傳送的數字占 process 的比例，經測試後發現每次全部都傳速度最快，傳送後的數會和下個 process 中的數進行 merge，再將較小的那一半數字傳回

c. 終止情況：

每輪計算結束會判斷一次，merge 回傳值為 0，即代表 process 間有數字進行交換，得繼續進行下一輪計算。

d. Timing Mechanism：

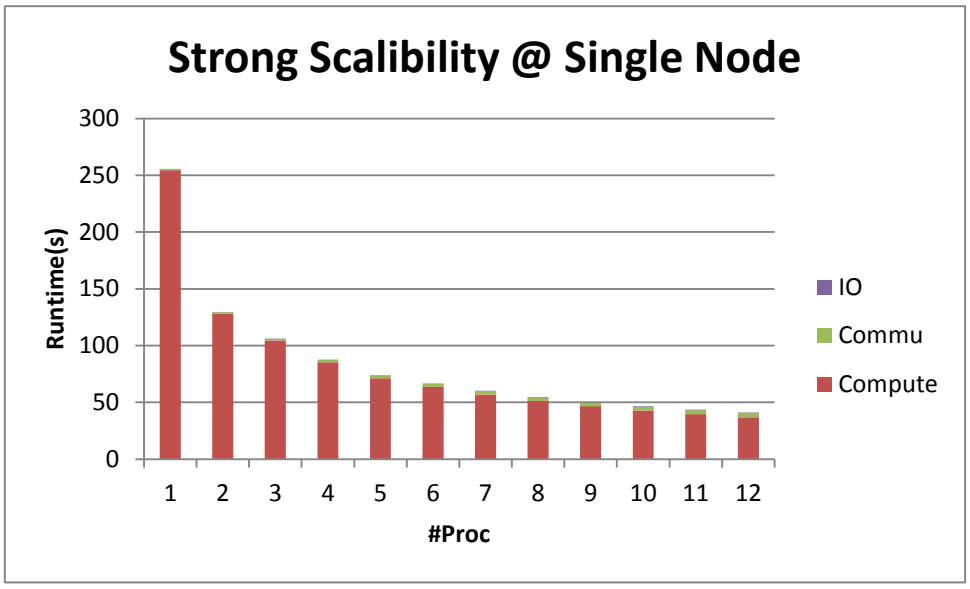
在程式碼中插入 MPI_Wtime()取得當下時間，再分段相減，計算檔案操作的 IO 時間，process 間 send、receive、broadcast 等訊息交換的時間，和剩下的 CPU 計算時間。

2. EXPERIMENT & ANALYSIS

除了 I/O 測試外，Advanced 版本皆使用 testcase_4G 中的 5*10^8 筆資料，Basic 版本使用自己產生的 5*10^5 筆資料，資料皆測試 5 次，取靠近眾數資料的平均

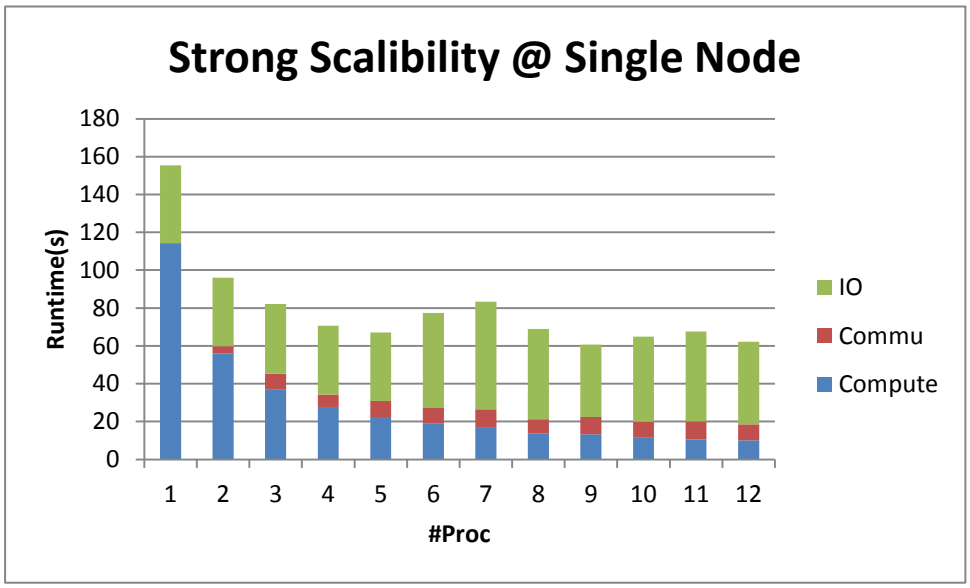
1.) Strong Scalability & Time Distribution

a.) Basic version on a single node



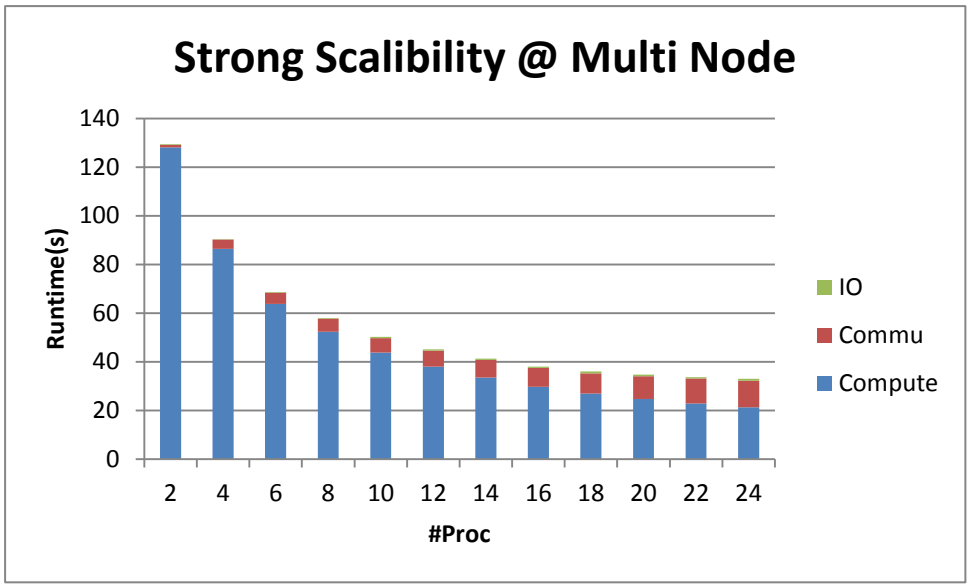
由於 Basic 版跑不動較大的資料，IO 時間佔的比例並不明顯，大部分的時間還是花在 process 內的數字交換上面

b.) advanced version on a single core



由於 IO 花的時間感覺受到網路狀況的影響滿大的，所以沒有像在 pp11 上執行那樣穩定，而 communication 時間隨 process 數變多有稍稍增加但不明顯，畢竟也是平行傳送資料，而 compution 的加速最顯著。

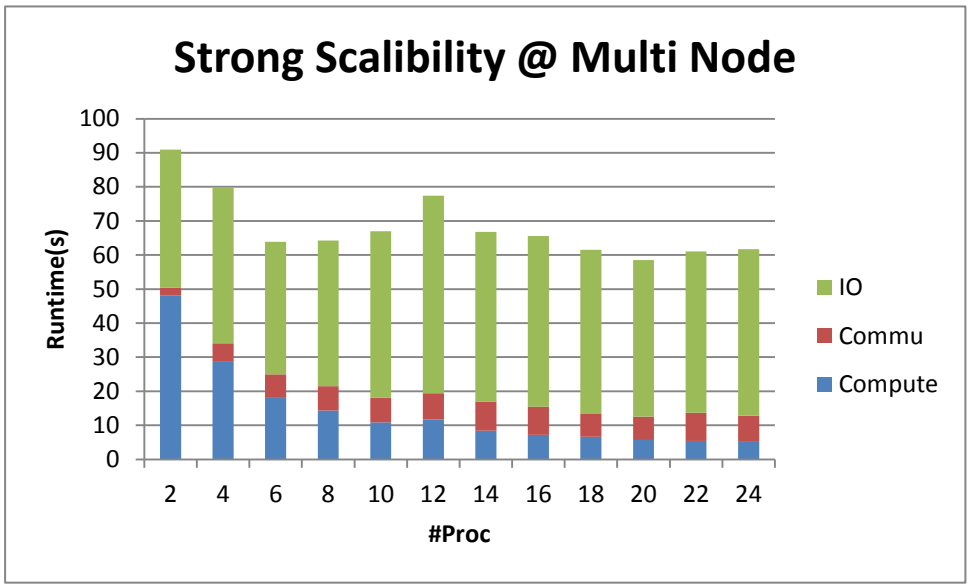
c.) Basic version on 2 nodes



可以注意到 communication 時間有顯著的增加，可能因為一次只傳一個數，造成 overhead 的影響較大

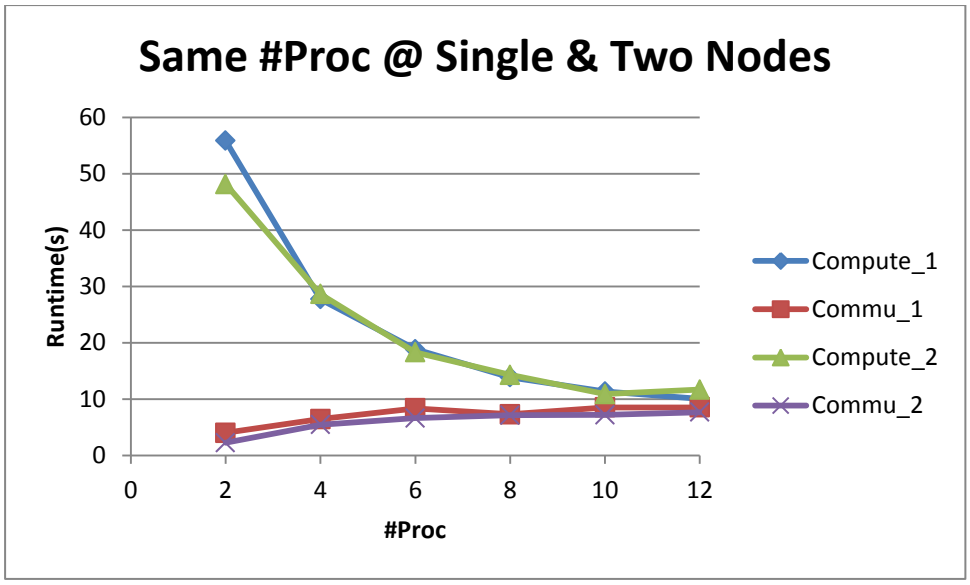
d.) Advanced version on 2 nodes

在兩個 nodes 上跑



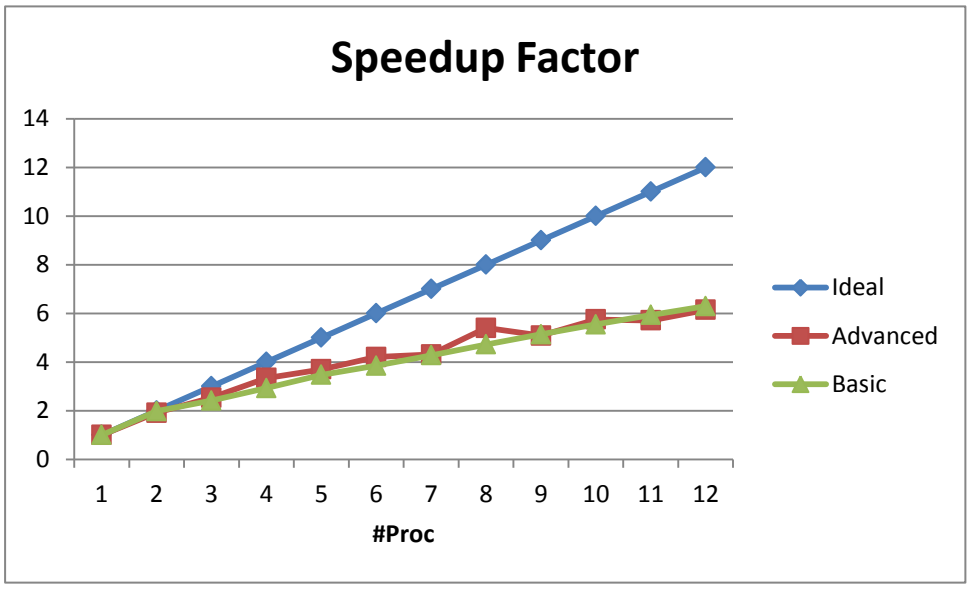
因使用的 process 數變多，IO 佔的比重也較高，加速變得不明顯

e.) Same number of process on single and two nodes respectively



執行的是 Advanced 版本，可以發現時間並無顯著差異，本來以為 communication 時間在 2 個 nodes 時會較高

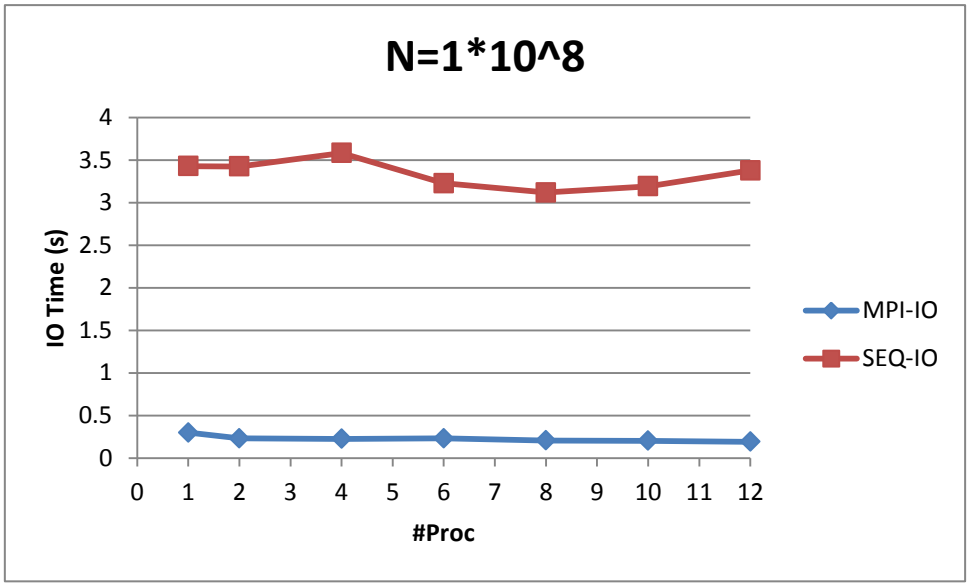
2.) Speedup Factor



從結果可以看出 Basic 和 Advanced 版本 system efficiency 都差不多是 S(p)/p = 6/12 = 50%。S(p)=p/[1+(p-1)f]，計算後不可平行化的 f 大概佔 1/11。

3.) Performance of different I/O ways

使用 hw1gen 產生的 10^8 筆 int，取 5 組資料平均。



MPI-IO 我使用 MPI_File_read_all()、MPI_File_write_all()搭配 MPI_File_seek()；sequential IO 則使用 fread()、fwrite()在 root process 一次讀寫全部檔案，可以發現 MPI-IO 速度差不多為 sequential IO 的 13 倍，不過越多 process 需要的資料交換時間也越多，因此 scalibility 較不明顯。

4.) Compare two implementations

首先就 sorting 方式來看，basic 版本就完全是用複雜度 O(n^2)的 odd-even sort 處理全部的數，最壞情況有多少個數就必須交換幾輪；Advanced 版本先用平均 O(nlogn)的 qsort()處理各 process 內部的數，之後的 merge 複雜度 O(n)，最差情況數字僅需交換 process 的個數，速度快了 200 倍以上。

3. EXPERIENCE / CONCLUSION

第一次寫平行化的程式還滿新鮮的，一開始有時還會誤以為變數是共享的，而 MPI 提供了很完整的訊息傳送方式，和平行化的 API 來處理檔案讀寫，廣播訊息等操作，唯一的缺點可能是無法動態產生 process，若需要類似共享變數的控制也需要額外的訊息交換時間。寫程式上遇到的困難，主要是了解 MPI 各引數意義，還有存資料的陣列長度的設定，此外在工作站測試跑資料也很花時間，從決定測試資料的大小，到真的開始實驗需要一段時間；輸出的時間也受伺服器負荷量的影響，差異非常大，因此跑了很多次無用的資料。

從這次實驗結果發現要達成理想的 speedup 並不容易，感覺自己寫的 Advanced 版本僅僅是演算法上的優勢，Scalibilty 並沒有較高，目前沒有想到更好的辦法。