Parallel Programming hw1 ----OddEvenSort

學號：109062639　姓名：葉哲欣

Implementation:

N: number of input　size:number of process　rank:process id

將 N 除以 size,使得每個 process 可以得到 N/size 的 data，若 N%r != 0,則將剩餘 data 平攤給 rank< N%r 的 process。

My Odd-Even Sort algorithm:

Step1:

Rank:

0	1	2.	……	size-2	size-1
Data	Data	Data	…	Data	Data

Step2:Pre-processing the local array .Use qsort to make array sorted.

Rank:

0	1	2.	……	size-2	size-1
Data	Data	Data	…	Data	Data
小→大	小→大	小→大	小→大	小→大	小→大

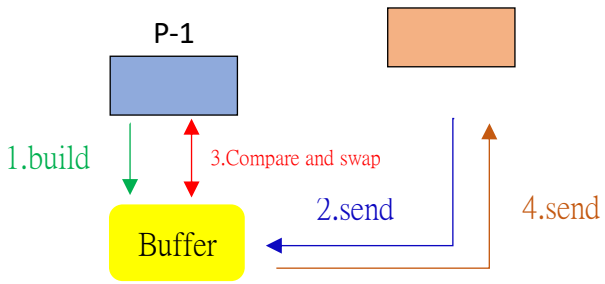
Step3:Odd-Even Sort operation

In even Phase:

P:odd rank　P-1:even rank

Rank:

P-1	P			
Data	Data	…	Data	Data



1.P-1 builds a buffer for P's Array data

2.P send local array's data to P-1's buffer

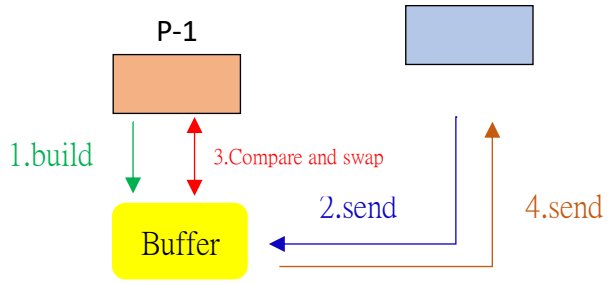
3. P-1 compares with Buffer,if P-1[i]>Buffer[j] ,swap and keep the array sorted.

4.Complete(3.)and send Buffer's array data to P

In odd Phase:

P:even rank P-1:odd rank

Rank:



1.P-1 builds a buffer for P's Array data

2.P send local array's data to P-1's buffer

3. P-1 compares with Buffer,if P-1[i]>Buffer[j] ,swap and keep the array sorted.

4.Complete(3.)and send Buffer's array data to P

當所有 local array 執行都沒有 odd even sort 兩兩沒有交換動作，即完成排序。

MPI I/O ---- read write file offset

```
//檔案讀寫offset設定
int rank_num ;
if(r==0)rank_num = rank*num_elem;
else{
    if(rank>r&&r!=0)rank_num = rank*num_elem+r;
    else rank_num = rank*num_elem;
    if (rank==r&&rank!=0)rank_num = rank*(num_elem+1);
}
```

上圖是我在作業中對 read write 的 offset 作設定，以免再寫入檔時發生 data overload 問題

r= N%size

若 r!=0 →代表 rank>= N%size 的 process 的 offset 會 overload 前 r 筆資料

Experiment & Analysis:

I. Methodology

A. System Spec:

使用 apollo 機器

B. **Performance matric**

Time measure method:

使用 lab1 助教教的 MPI_Wtime( ):

Ex:

```
start = MPI_Wtime();
//compute
end = MPI_Wtime();
Time = end -start;
```

程式結束前將每個 process 的 CPU time、I/O time、以及 Communication time 平均後 print 出來，統計下來後繪成 plot。實驗中取作業的 testcase 做 data，分別為 17,18 ,29,30 最後選擇 testcase 30 製作圖表 。

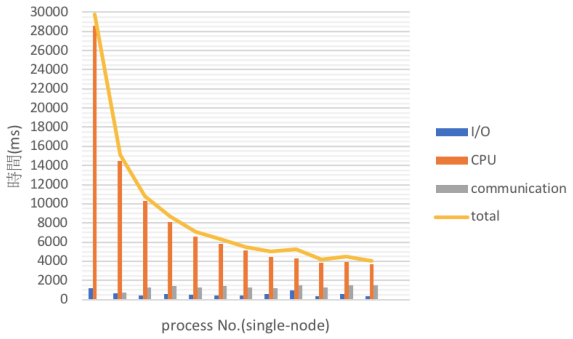
II. Speed up factor and Time profile

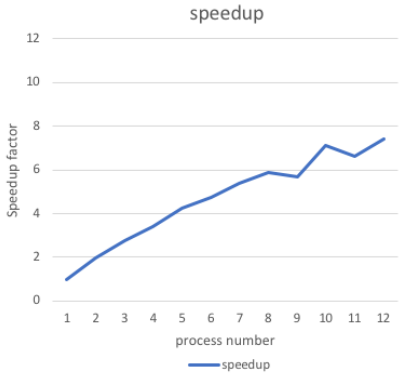
Data size:64123483(testcase 30)

實驗採用 strong scalability

A. Single node multiple process

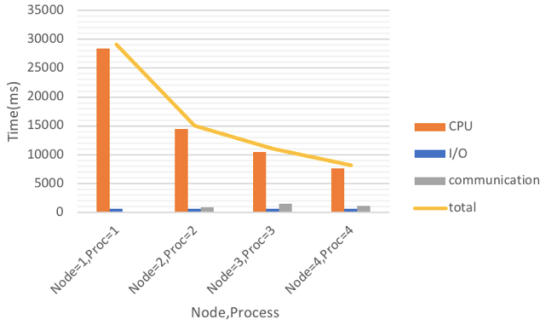
X 軸：process 數(單一 node) Y 軸:時間

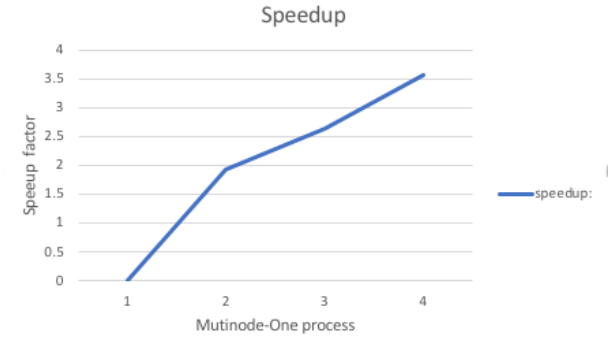




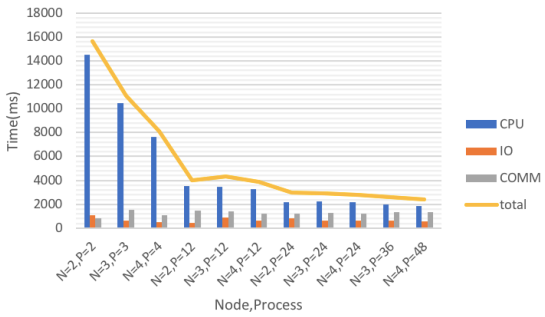
B. One process per Multinode

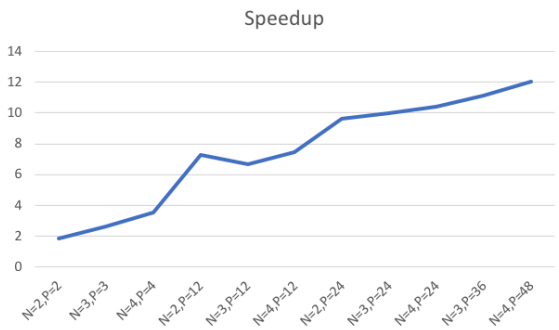
X 軸：Node 數, Process 數 Y 軸：時間





C. Multi-process in Multi-node





III. Discussion

- Compare performance:

從 上面 3 個圖我們可以看到，process 越多，不代表執行時間會越少。當 process 的數量到一定程度時，CPU time 時間差不多
，然而 communication time 逐漸增加，但與 CPU time 相比情況下成長幅度較小。我的實驗遇到的 bottleneck 主要是 cpu time 的 performance 的改善，作業中我將 input 平分給每個 process 下去做處理，執行 odd even sort，兩兩 process 需比較大小並排序，我使用到 merge sort 的 comapre and swap 方式，由左邊 neighbor (process)接收右邊鄰居(process) data。且需要額外建一個 2 倍長的 buffer 來做暫存排序，排序完後將結果回傳給右邊鄰居(process)。此方法讓一半的 process 都在等待排序結果，造成 performance 下降原因。

- Scalability
  從 A ,B ,C 三個 time-plot 分析，圖上的黃線(total)顯示，我覺得我的 program 的 scalability 效果不錯，有隨著 process 增加而減少執行時間，三組 Speed up 圖也有一定的成長幅度。但仍然有很大的進步空間。C 圖 就很明顯發現 process number 來到 48，speed up 只有 14 倍左右，效能非常不理想。

  除了優化 program 的 Cpu time，另外 communication time 也須改善，需要再減少 process 彼此溝通的次數。

Experience and Conclusion

這次作業讓我受益良多，學到 MPI 的 point to point 的操作，以及了解 process 彼此該如何溝通，如何設計 send ,recv 才不會造成 deadlock，也學習到 process 該如何有效平行處理 data，以及存取上的設計，才不會造成 data overload 。

作業主要面臨的困難是 MPI 的操作，data 該如何分配給 process 以及演算法上的設計。從完全沒有基礎，慢慢找資料，了解 MPI 怎麼溝通，怎麼交換資料，process 該怎麼讀取檔案，寫檔案。另外 makefile 的撰寫也是困難之一，這部分由於還在研究該怎麼寫以及怎麼運行，所以這次作業並沒有對 makefile 做進一步的修改。

```
make: Leaving directory '/home/pp20/pp20s54/.judge.086957476'
04.txt     0.82   accepted
01.txt     0.97   accepted
02.txt     1.62   accepted
03.txt     2.02   accepted
05.txt     1.57   accepted
06.txt     1.17   accepted
07.txt     0.87   accepted
08.txt     1.77   accepted
09.txt     1.72   accepted
10.txt     1.77   accepted
12.txt     0.67   accepted
11.txt     1.77   accepted
13.txt     1.77   accepted
15.txt     1.82   accepted
16.txt     1.42   accepted
14.txt     1.67   accepted
17.txt     1.87   accepted
18.txt     1.87   accepted
19.txt     1.72   accepted
20.txt     1.72   accepted
21.txt     1.67   accepted
23.txt     1.77   accepted
22.txt     1.77   accepted
24.txt     1.77   accepted
25.txt     1.67   accepted
26.txt     1.72   accepted
27.txt     1.72   accepted
28.txt     1.82   accepted
29.txt     1.87   accepted
30.txt     3.97   accepted
31.txt     3.52   accepted
32.txt     2.37   accepted
34.txt    25.28   accepted
33.txt    28.24   accepted
36.txt    13.90   accepted
35.txt    21.98   accepted
37.txt    11.19   accepted
38.txt    20.88   accepted
39.txt    17.92   accepted
40.txt    14.56   accepted
Removing temporary directory /home/pp20/pp20s54/.judge.086957476
Scoreboard: updated {40 226.48} --> {40 210.18}
```