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 |

P-1 P

Buffer

1.build

3.Compare and swap

4.send

2.send

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: P-1 P

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Data | Data | Data | … | Data | Data |

P-1 P

Buffer

1.build

3.Compare and swap

4.send

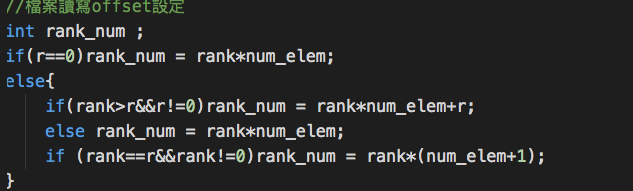
2.send

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** 

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:**

1. Methodology
2. System Spec:  
   使用apollo機器
3. **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 。

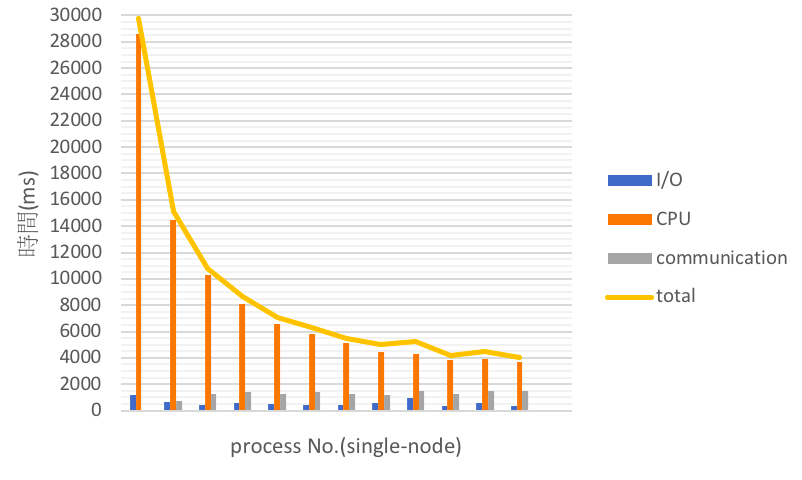
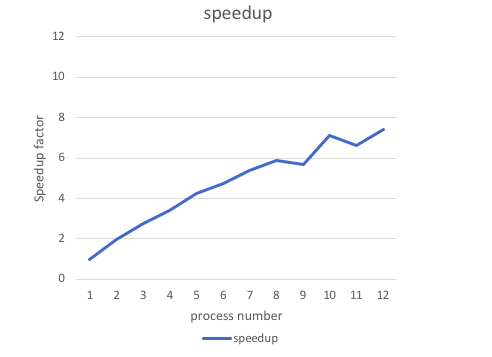
1. Speed up factor and Time profile

Data size:64123483(testcase 30)

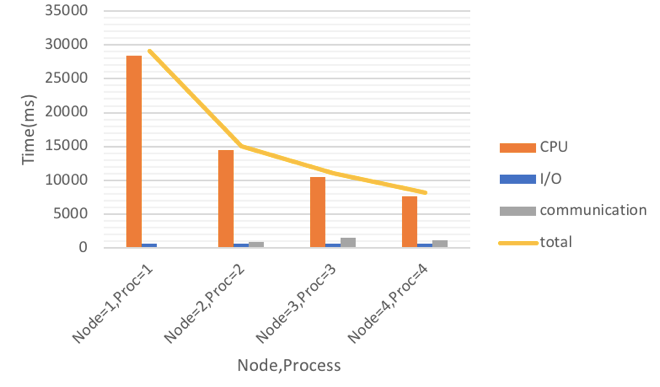
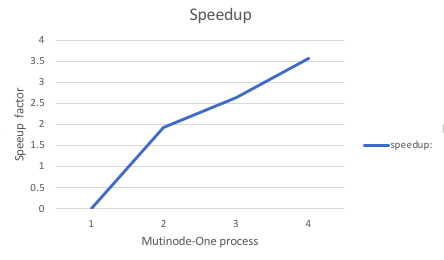
實驗採用strong scalability

1. Single node multiple process

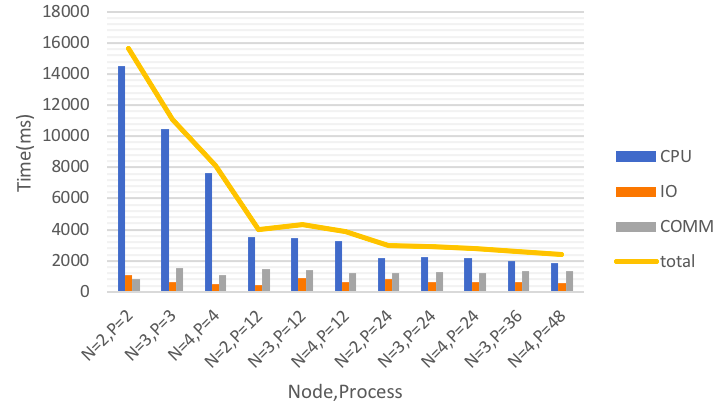
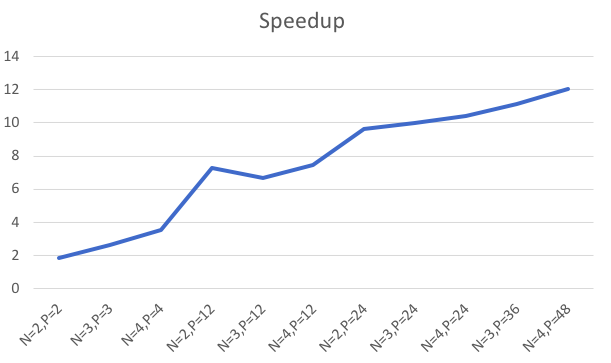
X軸：process 數(單一node) Y軸:時間

1. One process per Multinode   
   X軸：Node數, Process數 Y軸：時間

1. Multi-process in Multi-node

1. 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。

