HW1 report

Name：卓榮祥　　student ID：111062678

1. Implementation

MPI API：

Message passing	Input & Output
MPI_Send 、MPI_Recv	MPI_File_write_at、MPI_File_read_at

Memory allocation (3 Buffer)：

Buffer name	data	Buf	srBuf
用途	存放 local data	做資料交換	放置 merge 結果
size	num	num + rightnum	num

Handle arbitrary number of input items and processes：

對於隨機給定的 process 數量 p 和 input item 數量 n，首先我將 n 個 input item 先平均分配給 p 個 processes，因此每個 process 會先獲得 n/p 個 item，若還有其他 item 沒被分配到(n % p != 0)，則將剩下的依照 rank 由小到大分配出去，因此每個 rank 被分配到的 item 個數為 n/p 或 n/p + 1。

Odd-even phase：

Local data sorting 的部分我是採用 C 的 qsort() 函式，qsort()複雜度為 nlogn 且 sorting 過程不會額外 allocate memory。

對於每個 rank 只需 call 一次 qsort() 來排序 data，而在進行 odd even sort 時，在右邊 rank 傳資料給左邊 rank 的時候， 因為最初已經用 qsort() 對每個 rank 中的 data 排序完成，因此在 odd-even sort 裡頭只需將兩個 node 的 sorted array merge to one sorted array 即可完成排序，再將較大的另一半 data 傳給右邊的 node 即可。如此一來，不須將 sorting algo 放置於 odd even phase 實作的迴圈中， 只需要用 merge 進行合併排序便可大幅縮減 computation time。

2. Experiment & Analysis

Experiment data set：

File name：40.in **Size**：536869888

Performance Metrics：

利用在程式碼中插入 **MPI_Wtime()** 的函式測量 computing time、communication time and IO time。

Strong scalability：

Figure 1: Speedup

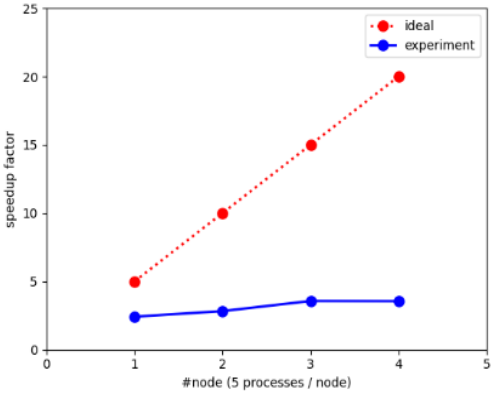
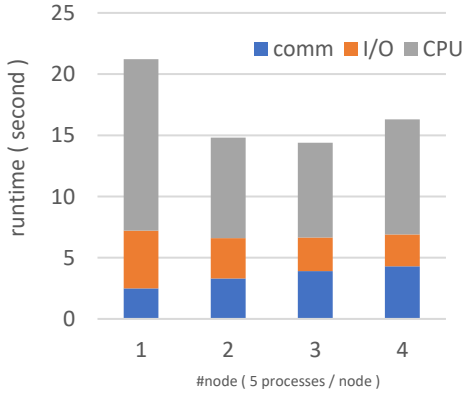


Figure 2 : Time profile



Discussion：

由 figure1 看到 ideal speedup factor 的成長和 experiment 差距甚大，可見有許多 parallel 的 overhead 造成這些結果。

figure2 也呈現出 node 數增加會造成 communication cost 會增加、I / O cost 會減少，但在 node3 和 node4 之間 CPU time 卻不減反增，原因為 process 數量變多， 導致 odd even sort 的執行次數增加造成 cpu time 計算次數變多，也導致 node4 的 Speedup factor 相較 node3 低了一點，我認為我程式的 scalability 不太好，需再針對 CPU time、communication time 做改進，看能不能找到更好的 sorting algo.來對 local data 做排序，或者是將 buffer 數量減少以減少 memory allocation，亦或者更加善用 MPI API 盡可能地減少 communication time 的時間，但對於 I/O 很難再利用程式的優化來大幅降低時間，因為對於每個 process，讀 / 寫 檔案都已經利用 MPI lib.平行運作了，而 CPU time 和 communication time 相較於 I/O 會有更多優化的空間。

3. Experiences / Conclusion

在上這門課前，對於平行程式這門學問相當陌生，在這次作業中讓我了解到 memory 的分配、process 間的 massage passing 和平行度對於降低程式執行時間相當重要。些微的調整經常能大幅降低程式執行時間。在這次作業當中，起

初對程式語言、作業環境相當不熟悉，花了很多時間熟悉環境和 debug，看似不太困難的演算法，但一路還是跌跌撞撞，自己寫程式總習慣用 serial 的角度思考，在寫完第一版時程式執行結果相當不甚理想，大量的 data 檔案底下跑到一百秒都有過，看到這”驚人”的成績就促使我開始思考如何提升平行度、如何減少記憶體的分配，在調整過後在大資料量的檔案下 run time 時間縮減了四倍左右，讓我感受到平行度的魅力以及重要性。實驗的數據以及製作的圖表看見了 scalability、bottleneck、速度等相當豐富的資訊，能夠看清楚資源的分布之外，也讓我對自己寫的程式有更進一步的瞭解。