**CS542200 Parallel Programming**

**Homework 1: Odd-Even Sort Report**

106062210 陳則翰

* Implementation

本次作業實作Odd-Even Sort，我先將所有n個elements平分給所有MPI Process，如果有不能平分的情況就把多餘的k個分給首k個Process。之後，給每個process定一個odd\_rank跟一個even\_rank，來決定他們在每一個phase要跟哪個Process溝通。之後在sort過程中，首先讓每個process先把local的element做一次std::sort()(只有一開始需要 因為之後都是sorted sub array)，然後讓相鄰的兩個process做一次merge\_sort裡sort的動作。如果只把一邊的檔案送過去做merge的話就只有一個process在工作，為了提升work load並且避免算完的東西還要送回去，我直接將兩邊的array都互相傳送。讓他們各自merge完之後把自己應該拿的部分放進本來的local位置裡。之後就照著bubble sort的sorting principle: sort到沒有改變為止。

* Experiments

註:testcase38

* Discussion

根據在圖表裡所能看到的，當process數量上升時，Scalability簡直奇差無比，可以說只有當process從1個提昇至4個時有速度上的提升。至於會這樣的理由，很明顯地，雖然提升process的數量的確有助於需要運用到計算的部分有顯著的成長，但所帶來的IO時間跟Comm時間成長實在是太顯著了。這邊我推測是因為當我們的process數成長，我們在一輪內所需要作的SendRecv次數也相對成長，自然Comm時間就會成長，如果想要改善這個問題，可以使用更聰明的演算法，例如，當我的rank在另外一個phase有改變時才會進行SendRecv的比較(但是如果要知道另外一個rank有沒有改變就又需要用另一輪SendRecv來作，也許不會有提升)。另一方面，不知道為什麼IO時間在process數上升時也跟著上升了，而且上升幅度還比Comm時間來得大。

* Difficulties

這次HW中我遇到一個不可期的問題，雖然在coding的時候有查到當我需要同時send跟recv東西時，使用Sendrecv()會好於分別使用Send()跟Recv()。但我沒有深究也沒有實際避免這麼做。結果當我的node數增加時(我本來都跑node不多的側資)，就有process莫名其妙收不到Message的情況發生，一把Send(),Recv()改成Sendrecv()之後就沒事了，實在是很莫名其妙，大概就是functiony在我看不到的地方的優化造成的這種現象吧。