

REPORT

1. Title, name, student ID

Parallel_Programmin_HW1 吳竣宇 109062671

2. Implementation

- a. How do you handle an arbitrary number of input items and processes?

由於n不一定為process的倍數，為了盡可能將elements平均分配到process中，我將 $n/\text{proc} + n\% \text{proc}$ 個elements放在最後一個process中，其餘的processes的local elements皆為 $\text{int}(n/(\# \text{ of processes}))$ 個，若是遇到processes比elements多的情形，則將每個element放在不同processes中並利用communication implement Odd-Even sort。

- b. How do you sort in your program?

(a) 首先，我先對各個process的local elements進行sorting,

(b) 進到Odd-Even sort的Even-phase,

將rank為Odd的Process P的numbers傳到Process P-1,

(c) 並於Compare奇偶Process numbers的大小過程中利用merge()->partition(),

將value較小的numbers留在Process P-1中，較大value的numbers0傳回Process P

(d) 接著進入Odd-Even sort的Odd phase,

將rank為Even的Process Q的numbers傳到Process Q-1,

(e) 並於Compare奇偶Process numbers的大小過程中利用merge()->partition(),

將value較小的numbers留在Process Q-1中，較大value的numbers0傳回Process P

(f) 重複執行以上(a)~(e)的步驟，直到在兩個phase中的comparison都不需要交換 or sorting為止，我們就完成了global的sorting

- c. Other efforts you've made in your program.

(a) 一開始嘗試將sequential code的swap平均分散在各個process中執行，

也就是若processes的local numbers在global中index為奇/偶的elements進行swap，

在process之間每次只傳輸一個element，此方法慢的原因主要是Odd-Even phase的迭代執行次數過多，

會浪費不少時間在processes之間的communication，最後的實驗結果為百萬筆以上的測資會TLE。

(b) 我在b.(c) Compare奇偶Process numbers的大小過程的implementation嘗試過先找出兩個sorted的array的median,

再利用median去判斷elements是否要傳回原來的Process，但在嘗試這個方法時出現了兩個問題：

我的terminating condition為Process P-1原有的sub array尾端 $<=$ 從Process P傳來的buffer前端，

由於有些測資中具有過小的element，在比較中位數和其他elements之間的大小時可能出了問題導致我無法達成terminating condition，

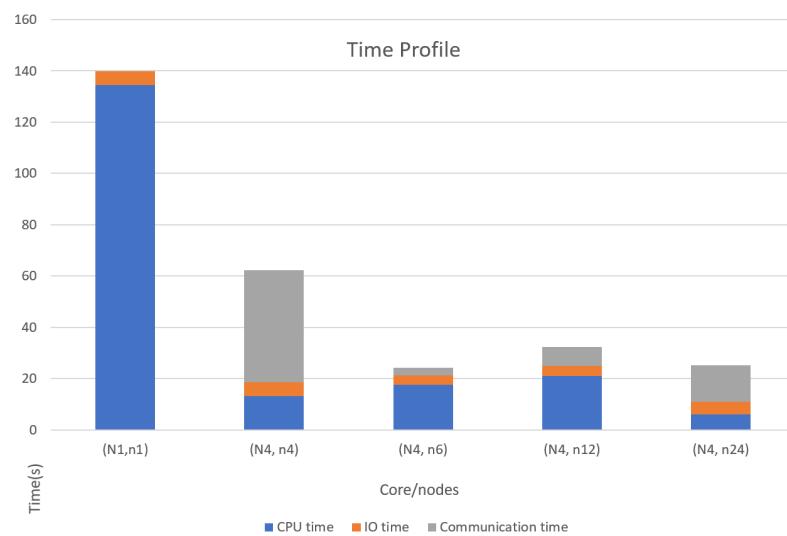
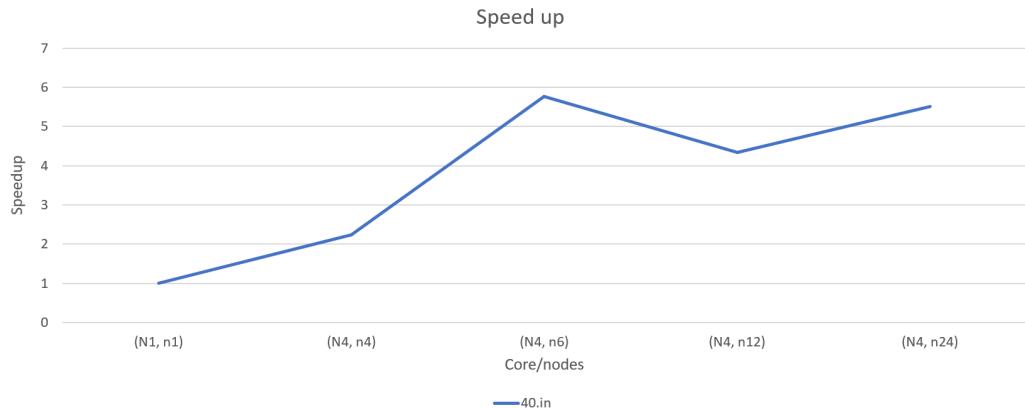
另外起初使用找中位數的方法的原因是因為其只需要 $O(\log n)$ 的時間，將array和buffer的elements重新分配所需的時間是 $O(n)$ ，

不過因為我們一個iteration有Even phase和Odd phase，若要用median的方法implement，

我們就得在Even phase和Odd phase之間做process內部的local sorting，如此一來可能就沒辦法比使用merge two arrays的方法還快，

3. Experiment & Analysis

- i、Methodology:
 - (a). System Spec: 我的實驗皆在apollo的server進行
 - (b). Performance Metrics: MPI_Wtime()
 - IO time = File read time + File Write time in parallel
 - Communication time = Total time of sending buffer to other process and receive back
 - CPU time = Total execution time - IO time - Communication time
- ii、Plots: Speedup Factor & Time Profile



- iii、Discussion (Must base on the results in your plots)
 - uni process時CPU time佔比較高
 - multi processes時Communication time的佔比較高

- 關於(N4, n4)的communication time異常高我並沒有理解確切原因，可能跟server使用人數過多有關
- 我認為bottleneck在CPU time，因為我把n/processes的remainder都丟給last process了，可以藉由更好的scheduling減少CPU time才對

4. Experiences / Conclusion

- Your conclusion of this assignment:
 - 學到MPI相關的API操作
 - 透過觀察得知測資的設計方式
- What difficulties did you encounter in this assignment:
 - MPI相關的API操作
 - debug不易