

# 平行程式作業一報告

李浩榮 110062401

## Odd-Even-Sort

### 1. Implementation 實作

這次的作業要做的事是通過使用MPI的interface去實作Odd-Even-Sort演算法。整個流程大概可以分為分配記憶體，讀取input data, MPI溝通，回收統整，輸出output。

首先是分配記憶體。這部分是整個程式中非常重要的概念，我們要根據testcase的要求看有多少n (亦即input data)以及有多少#proc(number of process)可以用。為了把要處理的input data平均佈置給每個proc，會以n/size去算出一個proc要拿多少input data，而大部分的測資#inputdata都是會多過#proc,所以必然會有一些proc要處理比其它proc多一個的資料(這部分以n%size)去實作。最後決定一個proc要拿多少inputdata則是以quotient + (rank<remainder)為準則去assign。

而針對每一個proc, 我都會分配三個記憶體空間(data, temp, buf)給它們。首先是data ( n ) 負責儲存當下這個proc真正需要用的資料或者保存經過另一個proc sort排好後傳回來的input data們最後reduce的時候要寫入到output file當中 ;temp ( n ) 是用於第k個proc把資料傳給第k+1個proc做mergesort的時候一個暫時存放的空間，最後會把它存的資料正確地assign給data ;最後是buf ( n+1 ),buf的功用是複製data的內容並且擔任send/recv交流的記憶體，這裡空間會給多一個當然是因為有些proc會擁有多一筆的資料，所以這裡buf的空間配置會以每個proc最多可能處理的inputdata為上限。

讀取inputdata就以MPI內建的API (MPI\_File\_read\_at) 去一次全部讀取進來，分配給每一個proc。

再來是核心的sorting實作部份，Odd-Even-Sort會以Odd Stage和Even Stage兩個階段交互進行。在Even Stage中，只有偶數idx(rank%2==0 或者 n&1)可以和右邊的單數idx(rank%2==1)做communication，我把在Even Stage下的奇數idx下的proc的擁有的資

料(MPI\_Send)傳給左邊的偶數idx後，奇數就進入MPI\_Recv listening的狀態，等左邊的proc把sort好的資料傳回來。在Even Stage中偶數idx會一開始就MPI\_Recv等待來自奇數的数据(這裡是以buf接著)，再和自己的data做mergesort。實際上的做法是data和buf去iter跑過n次比較(array index 為到n-1為止)，比較小的值就會先被存在temp中，直到data, buf其中一個先iter完它的n，注意這裡假設data內的資料先被拿到temp中，這有可能buf中還有資料沒有被放到temp中，所以還要進行一次for迴圈，直到n筆資料結束buf把剩下的值也做完。Odd Stage做的事情也是和Even Stage一樣，但是差別是這次是由偶數idx先MPI\_Send，奇數idx負責MPI\_Recv接資料處理。

整個循環是以while迴圈進行，break while迴圈的關鍵是MPI\_Reduce回收全部proc的has\_swap值並做MPI\_LOR運算，結果存在result上，如果每個proc的has\_swap都是false(亦即沒有再做swap的動作，全部proc都已經sort好)，result的結果就會為false，跳出迴圈。has\_swap的改變契機是mergesort的時候，如果buf(從別人那裡接來的資料)需要assign給當下proc的data內(亦即進行兩個proc的資料交換)，便可以設定has\_swap為true。

## 2. Optimization & Effort Enhance 優化與改進

- Rank判斷以&代替%：

這樣可以減少處理的步驟，有學過計算機組織結構的話，&可以直接以兩個空間內的值做and。比起%運算子的算術運算，會需要call三個operation(搬出A變數，搬出B變數，叫出%運算子，再處理stack最上面的兩個memory(A,B)，再assign)可以節省不少時間(當然data量少比較難看出，當量一變大，就很有效)。

- 減少使用三元運算 - Condition?(True:False)

這個運算的步驟也很冗長，盡量避免。

- 陣列空間的的Memory swap以pointer實作

程式中頻繁地需要assign某個資料如buf,data到temp上，或者整個兩兩陣列交換值。一般人可能會以for迴圈去實作，但是其實可以重新指定array pointer head到對方的第一個memory上就能完成陣列交換。

- $\text{++i} > \text{i++} > \text{i+=1} > \text{i=i+1}$

前兩個加一的方式最有效率的，步驟最短，時間耗時最少。因為 $++i$ 會直接找到 $i$ 但位置加1沒有放去其他地方(temp);  $i++$ 會先叫出 $i$ 放在一個地方(temp)，再對其加1，再寫入回去； $i=i+1$ 會叫出 $i$ 放在一個地方，叫出1放在另外一個地方，叫出加法，算術運算，儲存寫入回去。這是很重要的概念，但是這裡我為了程式的可讀性，且這次作業的data量沒有到需要這麼做的地步，所以就沒有強烈完全遵從該規則。

- 盡量避免**Call Function**

大量迴圈下時常呼叫函數也是個耗時的動作，可以的話盡量把(小運算)程式寫在一起。但要注意的是，這會造成整個程式的可讀性，模組性，維護性降低。

- **Sorting Algorithm**慎選

在mergesort之前的前提是sorted array，至於要先用什麼方法sort都沒問題，但是畢竟不好的方法如bubblesort的在這就先避免，畢竟我們追求Time Complexity。我是用quicksort，但是手刻的quicksort和C內建lib的quicksort有的不一樣，他們已有先對寫法優化過，所以直接使用內建的lib有時候會有更好的效果，且非常明顯。

## 3. Experiment & Analysis 實驗與分析

### 1. Methodology 方法

#### 1.1. System Spec

- 這次的作業是直接跑在老師的lab上的server，每個人分配到的配額是最多4個Nodes，每個Node有12顆CPU，如果一個CPU跑一個thread，最多可以跑48個thread。
- 48GB disk space per user
- OS: Arch Linux
- Compilers: GCC 10.2.0, Clang 11.0.1
- MPI: Intel MPI Library, Version 2019 Update 8
- Scheduler: Slurm 20.02.5
- Network: Infiniband

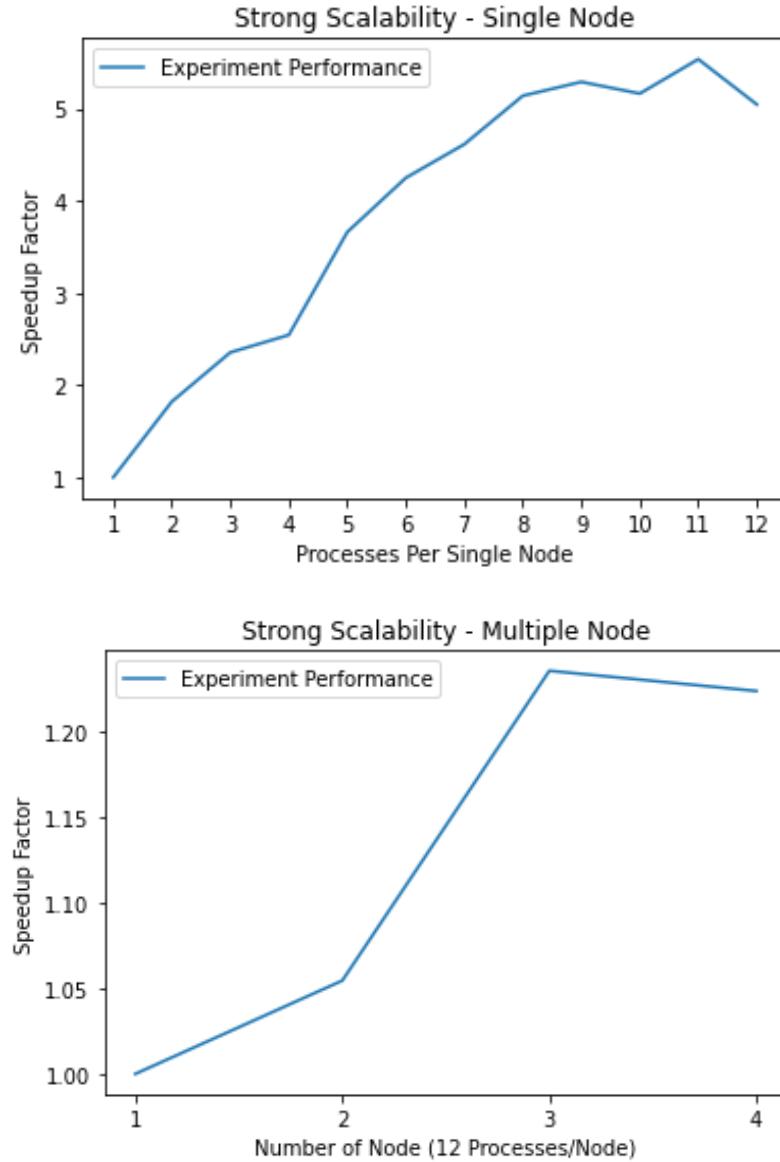
#### 1.2. Performance Metrics 衡量標準

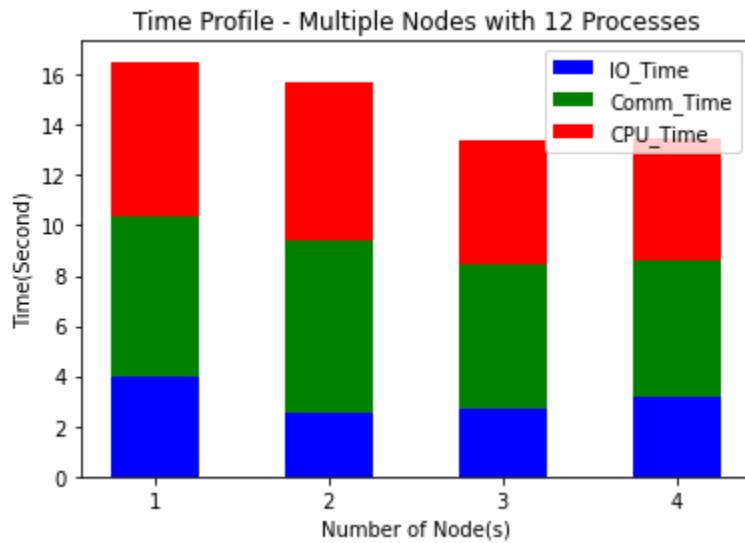
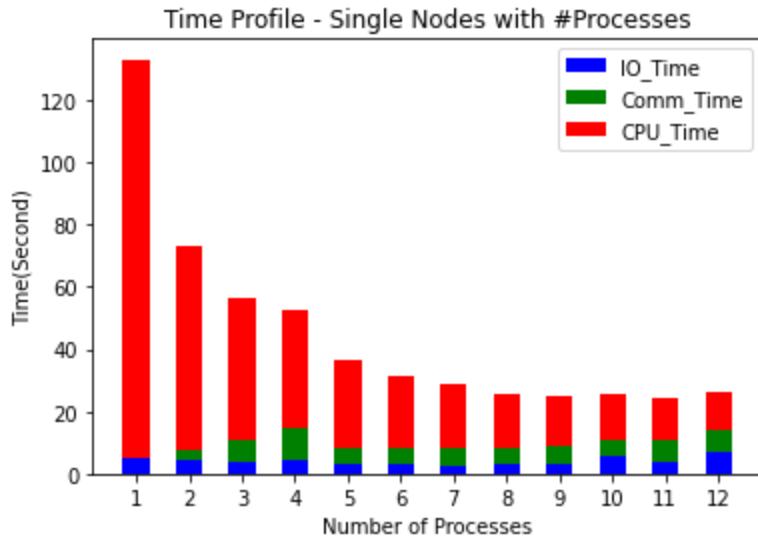
- 該作業衡量的時間實際上是要看**I/O Time**整個程式在前置作業完後，讀寫資料耗費的時間；**Comm Time** (Communication Time)各個proc使用MPI平台交流所消耗

的時間；以及**CPU Time** (Computation Time) 程式剩下的記憶體配置，算術運算等工作事項所耗費的時間。

- 我這裡是以MPI的MPI\_Wtime()去抓某個動作開始以及結束的時間，並計算時間差，記錄下來。

## 2. SpeedUp Factor & Time Profile 加速因子與耗費時間





### 3. Discussion 討論

這裡的實驗都是以 testcase 34 來執行, data 有 536869888 筆, 這裡選擇 testcase 34 是因為在 hw1-judge 的時候, 進入到 testcase 33 之後就開始需要花費比較多的時間, 所以這裡隨即選擇了 testcase 34 以彰顯程式的效能表現。其實從我的圖來看, 這結果離理想表現還有一點距離。首先是 Strong Scalability 的 Speedup Factor, 可以看到不管是 Single Node 還是 Multiple Node 隨著時間推移, Performance 加速都有一定的提升, 但是效果並沒有很明顯。但是還是有些特定 process 的時候, 表現會有點異常, 比如 single node 的 proc12 會突然降下來, 以及 multi node 的最後一個 node4 也把表現拉下

來。這可能是因為最後一個node只能和左邊的node溝通，裡面的proc效率也跟著提不上去，sort完成的速度會花比較多時間。

另外是Time Profile, SingleNode顯示隨著Process的增加，可以多個一起處理data，的確程式執行所需要的時間會有所明顯的下降。特別的是singlenode, singleprocess因為沒有其他process, 所有沒有communication time。而隨著時間越往後推移，IO\_Time幾乎固定(最少需要的時間)，CPU\_Time也有顯著下降，而Comm\_Time成本就變高，影響到整個時間的關鍵部分。

## 4. Conclusion 結論

第一次碰到這種trade-off看重Time Complexity，效率的程式作業。我有了前所未有的感覺，這堂課真的需要把大學部資工系教的所有基礎重要的知識點拿出來靈活運用才能完成的一個有意義的課。我重新體會到自己的寫法會導致電腦怎麼對記憶體操作，經過大量的data放大後的影響。對資料結構和演算法優化的重要性也不在話下，是這堂課這份作業需要運用到的，雖然我的ranking還是很後面～

硬要說在這份作業學到什麼，就是在參考和朋友討論後發現pointer的用法很重要，可以避免很多呼叫的步驟，是優化程式，演算法和資料結構的關鍵。具體的例子是上面提到的兩個陣列內容交換。還學到了除了Pthread外的平行程式溝通平台MPI的實作方法。

其實這份程式很簡單(trivial, 行數不長)，甚至關鍵的Odd Even Stage的部分蠻相似的，還有很多MPI\_Wtime()的呼叫，都會導致程式效率提不上的原因。在實作上，個人對資料結構比較不熟，配置記憶體空間後的運算常常會超出範圍，得到異常的亂數，基於此又有平行的多支程式的記憶體要一起抽象考慮是最難debug的地方，這也導致我個人對這些支持程式平行化的平台的完善性有了一定程度的懷疑，如果能夠有好的flag檢查MPI的回傳機制，以及debug error message等將會帶給想要平行化程式的工程師好的體驗。

最後希望助教能開放固定的office hour給修課的學生們，即使是半小時也好，能讓我們和助教討論想法，確認概念，最後再由我們自己實作，相信是會對學生們的學習，課堂，以及教授都有好處的。或者有一個Docs，能夠把同學們問過的問題以及助教們的回答寫上去，讓有相同問題的同學可以先預覽過，加速學習效率以及降低助教們的負擔～