CS542200 Parallel Programming

Homework 1: Odd-Even Sort

107062103 王依婷

1. Implementation

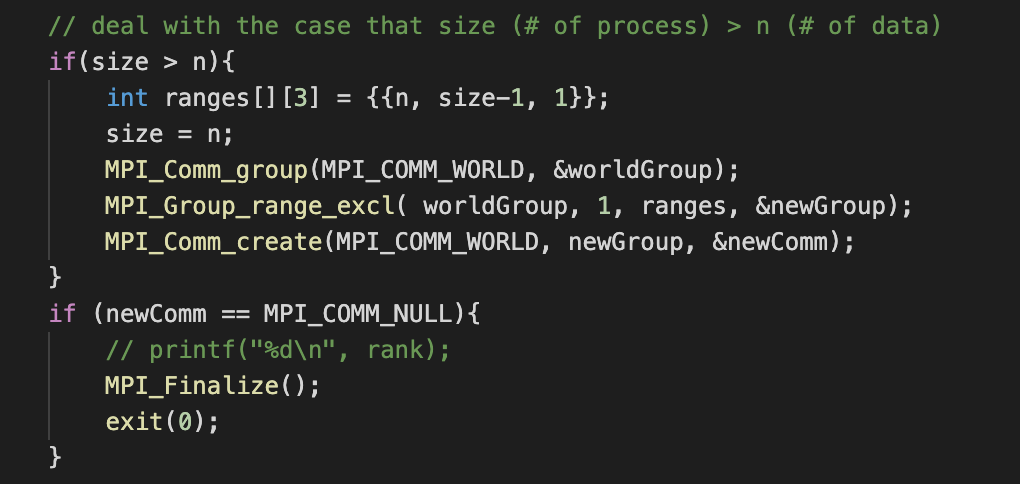
這次的作業要使用MPI實作Odd-Event sort，以下將整個程式部分分成I/O、process之間的溝通及sorting來簡單說明：

1. I/O

做I/O前，首先要確定每個process該取到的data數量。input的檔案大小對於實作有兩種狀況，一種是數字的數量（n）大於等於process數量（size），另一種是相反。若是第一種情況，則會將所有數字平均分配給所有的process，也就是說每個process都至少會拿到n/size個data，剩下餘數的部分再分配下去。



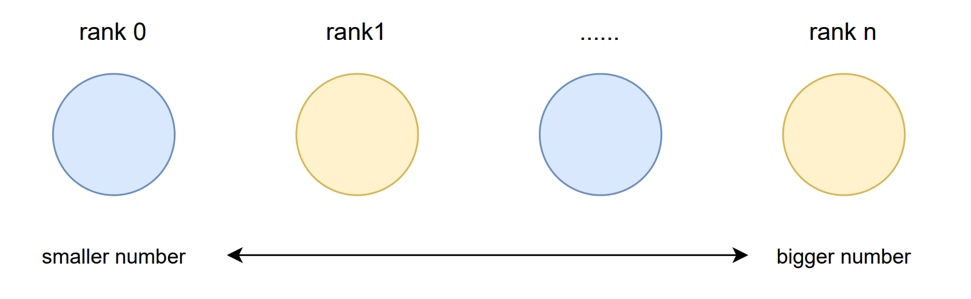
若是第二種情況，將會有一些多出來的process沒有事情做，因此將這些多的process結束掉，避免不必要的困擾。做法為建立一個新的group和communicator，排除掉多餘的process，呼叫MPI\_Finalize()把他們結束，後續做的一切溝通都使用新的communicator。第二種情況下每個process都剛好拿一個data。

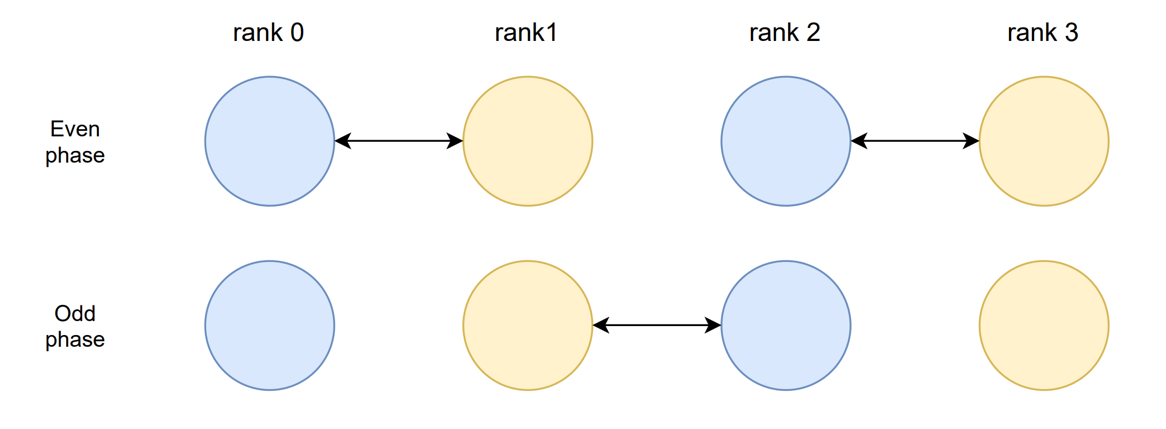


接著使用MPI\_File\_open / MPI\_File\_close開關檔案，再進行檔案的讀取。老師在上課時提到，MPI I/O有independent和collective兩種，collective在單一process的data並不大時使用會比較快，因為一次讀/寫較大量的data會比很多次的讀/寫少量data還來得快。因此我在這邊做個很粗略的判斷，如果data數量夠多時就使用independent I/O（MPI\_File\_read\_at/MPI\_File\_write\_at），否則使用collective I/O（MPI\_File\_read\_at\_all/MPI\_File\_write\_at\_all）。讀寫時的offset由process所取的data數量計算而來。

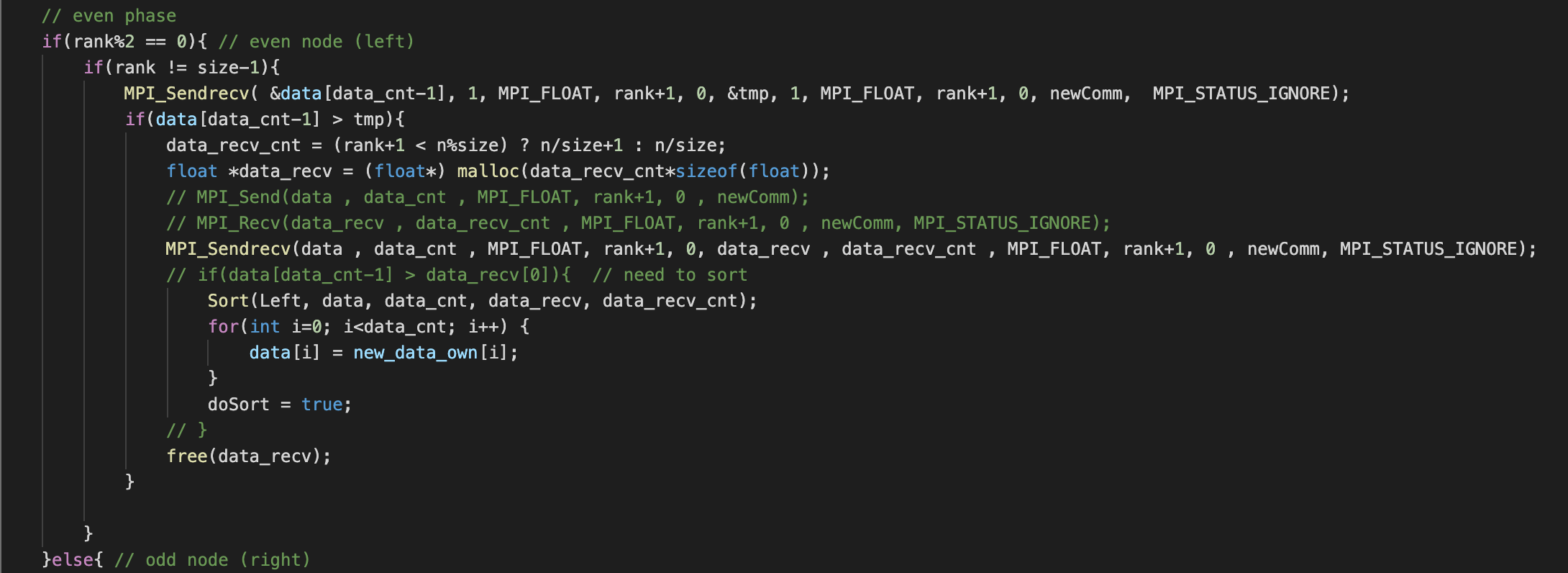
1. process間的溝通（Communicate）

每個process都有一串待排序的數字，內部排好之後會再和相鄰的process互相傳資料溝通比較，做進一步的排序。最後排成如圖所示之數字由小到大順序，寫到檔案中。



一開始我的做法是編號較小的process會將所有data傳給他右邊編號比較大的process，排序好所有data之後將較小的那一半送回去。但這樣一來，所有的計算只集中在一半的process之上，非常不平衡，因此後來修改成左右兩個process都會將自己排序好的data傳送給對方，一起做排序的動作，並且留下正確的那半邊。

首先，編號較小process傳送最大的數字給編號較大的process，且接收對方傳來的最小的數字，當編號較小process內最大的數字仍小於編號較大process內最小的數字，代表數字的順序已經排好，不需要再做排序，也不需要再傳接所有數字data；反之，互傳整串data來做進一步的排序。要注意的是編號最小（頭）和編號最大（尾）的process不一定會參與傳接data的部分，需要用判斷式判斷。

在傳送及接收data的部分，我都是使用花費較少時間的MPI\_Sendrecv來取代分開的MPI\_Send和MPI\_Recv。這邊附上Even phase中even node的程式作為輔助說明。

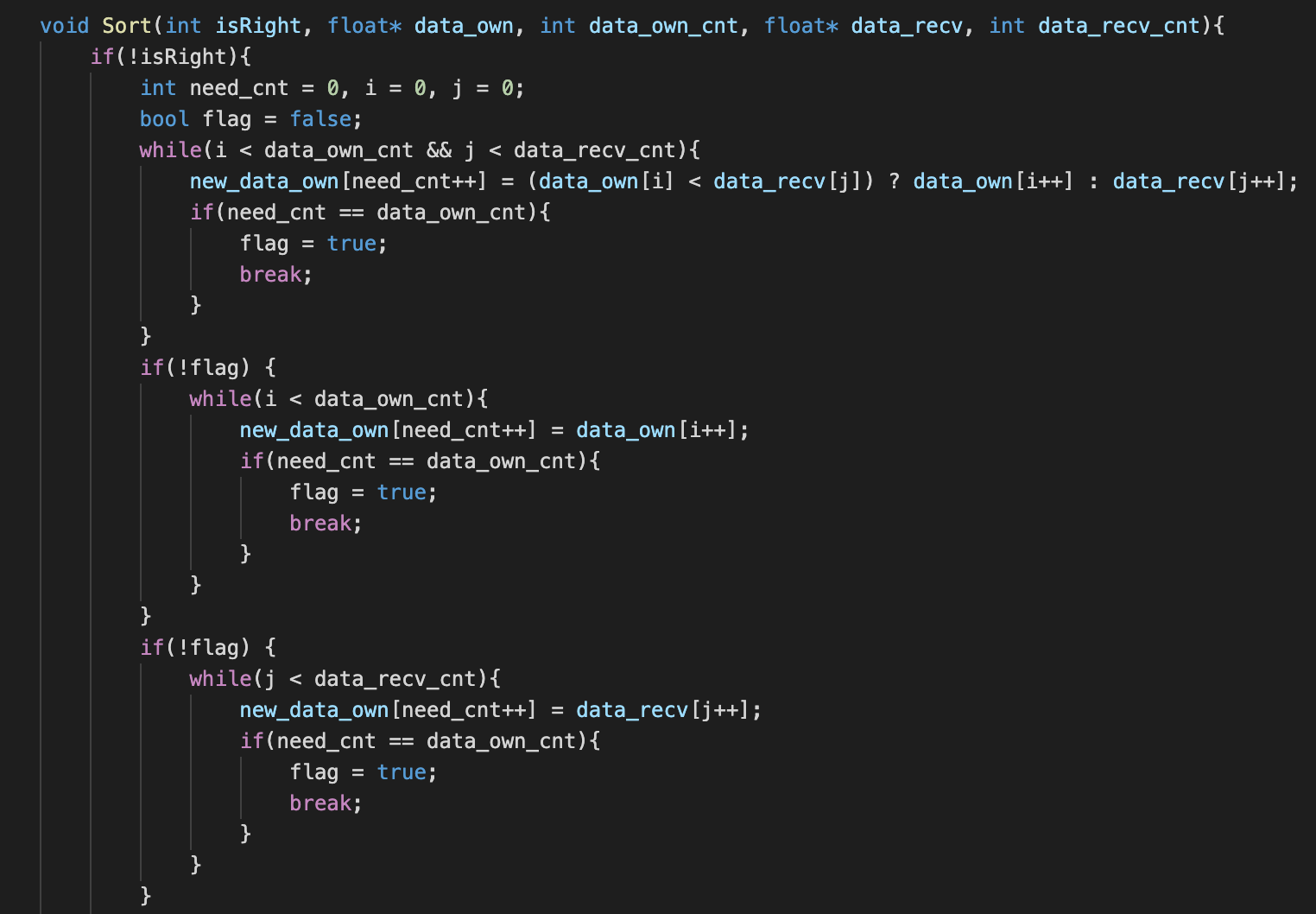
編號最大的process右邊沒有別人了，不參與溝通

先判斷是否需要進一步的比較、排序

1. Sorting（Compute）

process內部的排序，使用boost library中的spreadsort，選擇浮點數的排序float\_sort。兩個process進一步排序的部分，如上圖所示會呼叫自己寫的Sort function，傳入data及自己的位置（是編號較小的左邊還是較大的右邊）。

每次做進一步的排序時，都會將doSort設成true，代表這次的phase中有做排序。當所有process經過一個even phase和一個odd phase後，doSort都還是false的話，代表已經將所有data排序完成，這就是排序的終止條件。因此在兩個phase之後，會將每個process的doSort做logical OR，再發送給所有process來判斷是否要繼續排序，這個部分我使用了MPI\_Allreduce這個API來做。

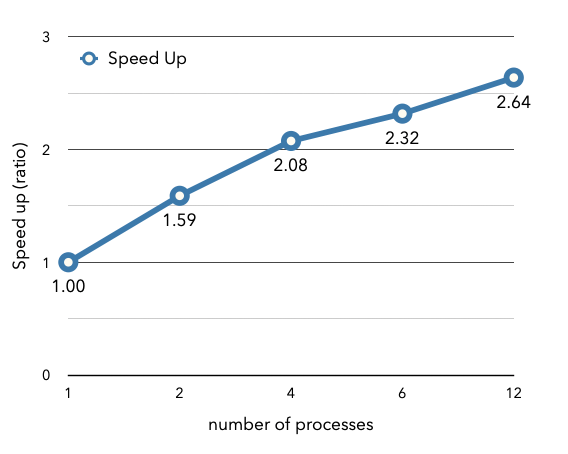
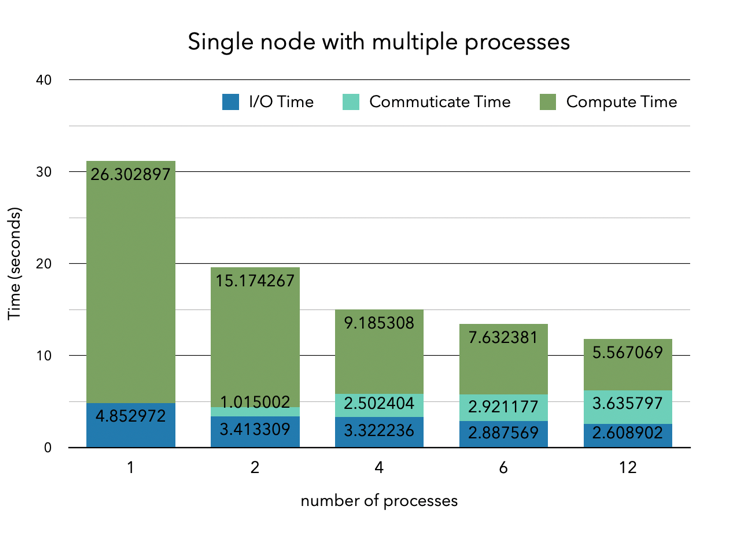
而在function中會像是要合併兩串有序data一樣，一一比較兩串data中的數值大小，放到新的array中。若自己是左邊，代表要拿數值較小的那半邊，因此從兩串data最小的數開始比較起，直到已取得足夠數量的數字為止。若自己是右邊，則從最大的數開始比較。如此一來，不但process間的工作量較為平均，計算量的差異最多也就是一個數字，更不用將兩串data完整的合併排序完才能得到結果，每個process也只要比較、排序到data的數量足夠就好。以下附上部分程式。

1. Experiment & Analysis
   1. Methodology
      1. System spec: 使用課程所提供的，並沒有使用其他額外系統。
      2. Performance metric: 在每個process上使用MPI\_Wtime() 這個API取得I/O前後、溝通前後的時間，相減後得到所花費的時長，而Computing time則是由全程花費時間減去I/O和溝通時間。 最後將所有process的各種時間用MPI\_Reduce加總取平均，reduce到rank=0的process印出，每種實驗設定都跑五次取得平均，繪製成數據圖。Speed up的部分是將實驗中最基本設定的花費時間除以其他設定花費時間，所計算出的比值。
   2. Plots

取第35個testcase作為實驗的case。

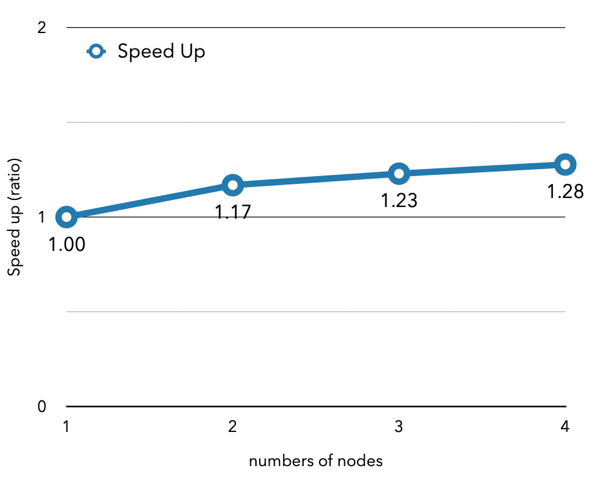
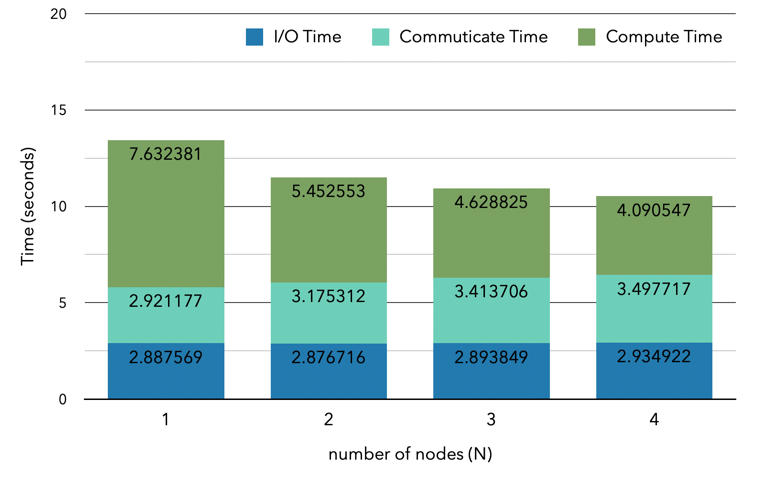
* + 1. Single node (N=1) with different numbers of processes (n):

探討在固定single node的情況下，增加process數量對performance的改變。



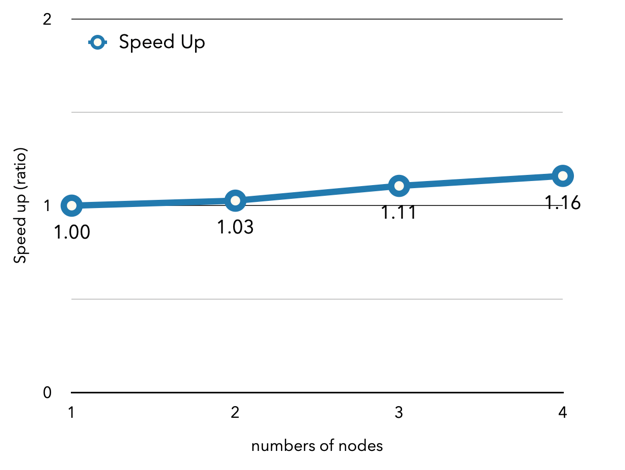
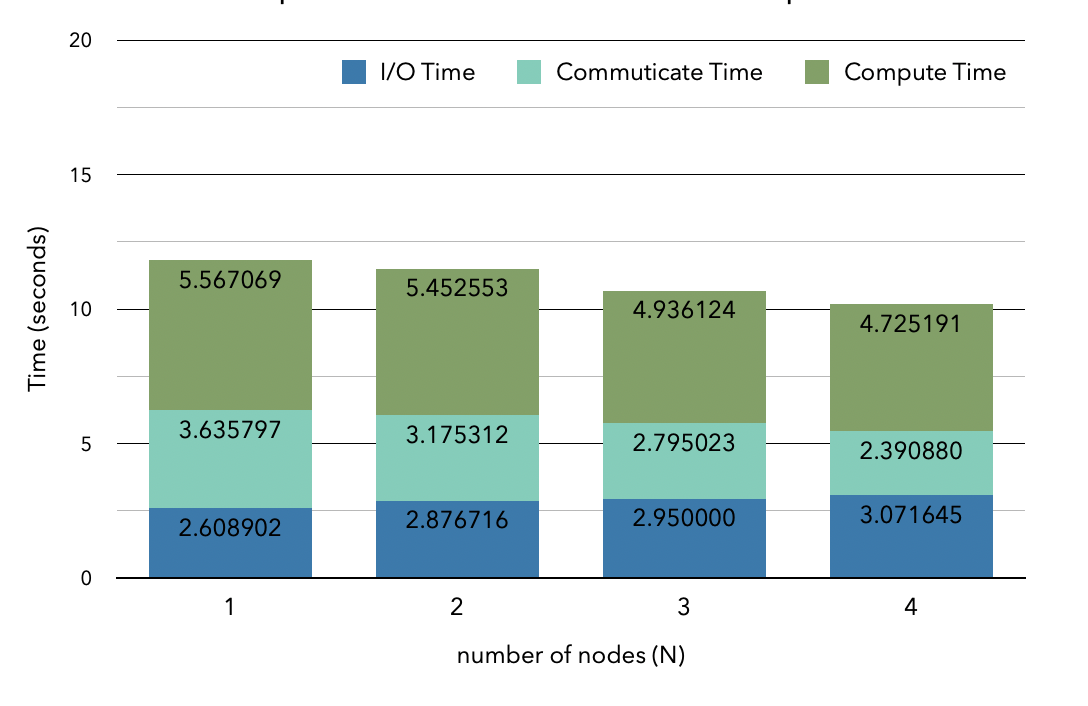
* + 1. Multiple nodes with fixed numbers of processes (n=6) per node:

每個node所使用的process數是固定的，改變總共使用的node數量，看看會有什麼差異？

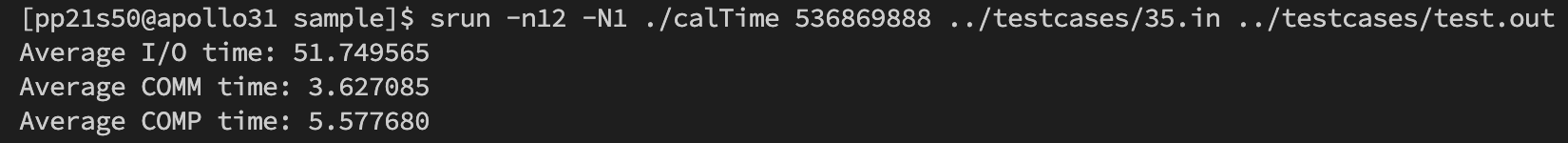


* + 1. Multiple nodes with fixed numbers of processes in total (n=12):

接著換成固定process的總量，試著分配給不同數量的node。



* 1. Discussion
     1. 除了同樣資料量的實驗之外，我也有試著以相同的配置跑不同大小的資料，發現當要處理、運算的資料量比較大時，花費最多時間的就是Computing time（資料量較小時是I/O）。在ii(a)的實驗中，process＝1時，代表所有的資料是全部一起一口氣做sorting，而process數量增加代表在同一時刻，不需要排序那麼多的數字，因此process內部排序所花的時間會成比例減少。但是實際上的數據在Computing time的地方減幅是越來越小的，可知比較大的問題是出在process間進一步的排序，所以可以針對這部份優化，Computing time就能夠減少。（當然，在process間的排序也能再使用更快的排序演算法來加速！）

不過在做實驗時，也發現如果有很多人同時在跑時，整體的速度會受到影響而變慢，其中I/O的速度慢了非常非常多，因此如果都是在多人使用的環境下，I/O就會成為最大的bottleneck。

* + 1. 我覺得這個program的scalability並沒有到很好，從ii(a)(b)都可以看到增加process數量後，speed up的變化是趨緩的，尤其是ii(b)，從single node共6個process增加到4 nodes共24個process，speed up都沒有超過1.5。可以觀察到在ii(b)中，I/O的時間並沒有很明顯的改變，但Computing time減幅不大，再加上較無法避免的Communication time增加，造成速度差異不大。也許因為這次的program比較沒有做到computing和其他部分的overlap，所以不能將時間吸收掉，若能試著在同樣的時間內做更多不同的事情，也許能夠增加scalability。

不過從ii(c)可以看到雖然process總量相同，將他們平均分配到不同node（機器）上能夠稍稍提高performance。

1. Conclusion

像老師、助教所說的，這次的作業要完成沒有到很困難，主要是要讓我們對沒使用過的MPI熟悉、上手，一開始覺得最困難的是有bug但是資訊沒有很多，找不太到問題出在哪邊，好難debug，還有還沒有那麼習慣平行程式的觀點。雖然作業本身不難，不過如果要讓performance有較顯著的提升，就不是那麼容易了，除了計算本身之外，目前還有點不知該從何處下手的感覺。透過第一次的作業，我覺得對parallel programming實作更熟悉了一些，以前只是在講義、課本中的知識、理論，在自己實際做了之後才更能體會「平行」帶來的優化和要關注的點。

1. Reference
   1. <https://www.open-mpi.org/doc/v4.0/>
   2. <https://stackoverflow.com/questions/13774968/mpi-kill-unwanted-processes>
2. 實驗數據連結<https://drive.google.com/file/d/1FbAmsrxmqQvZs1uWnqIkZCqbc3oU93FW/view?usp=sharing>