S103062631 鄭裔中 HW1

Implement basis :

如果每個process得到的為偶數

在even phase

0 1 2

0 1 2 3 4 5 6 7 8 9 10 11

在odd phase

0 1 2

0 1 2 3 4 5 6 7 8 9 10 11

如果每個process得到的為奇數

在even phase

0 1 2

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

在odd phase

0 1 2

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

由上又可分出odd跟even的rankID

一共會有8種case

其中又可化簡在偶數個element中even跟odd rankID行為一致

而又可化簡在奇數個element中even跟odd rankID行為相反

則最後分成四種case去比較執行

*而advance則是process中先行sorted

i、 System Spec

在提供的工作站上面執行

ii、 Strong Scalability & Time Distribution

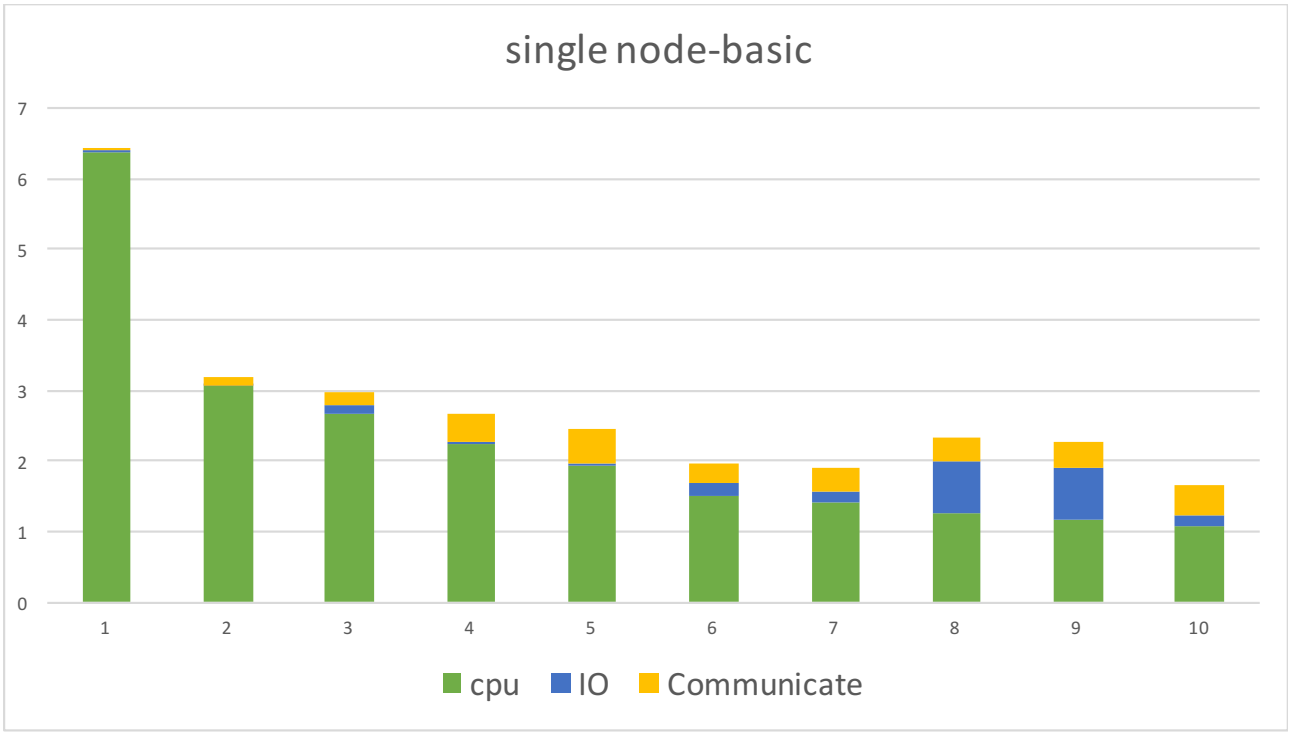
使用MPI_Wtime()去夾所要求的時間,是為了避免在ＭＰＩ底下又去呼叫其它計算時間的function

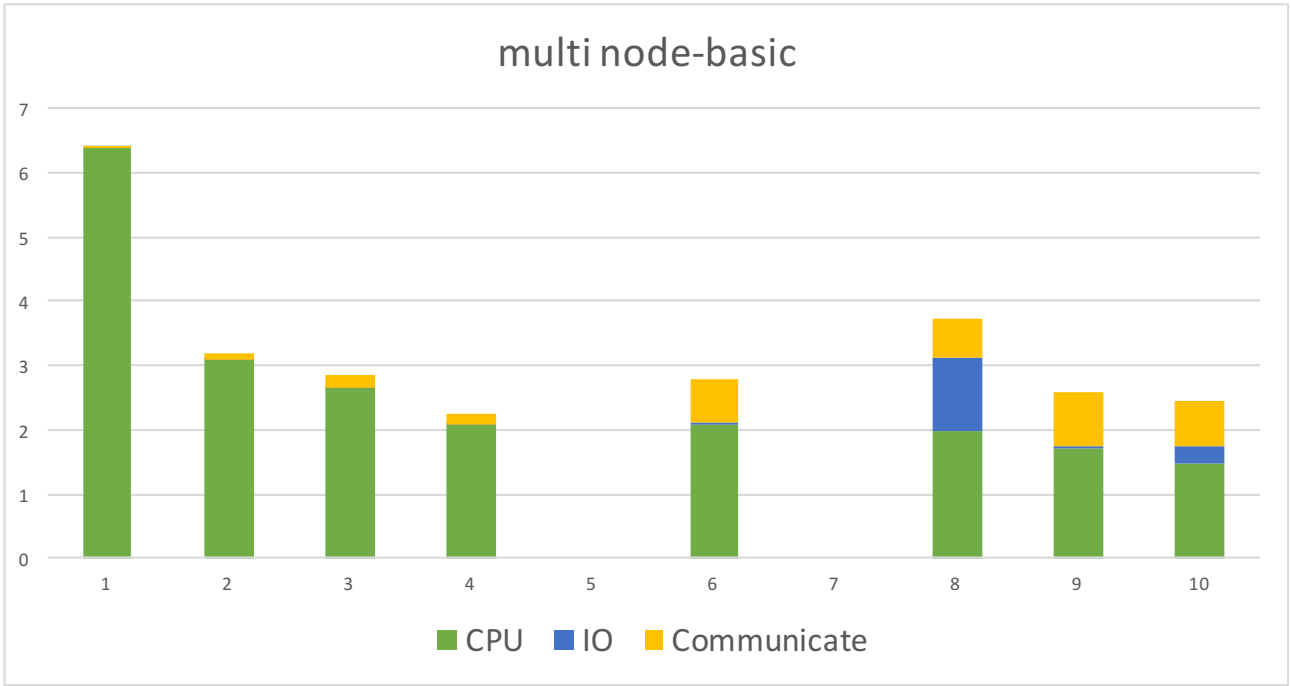
RANKID=ROOT為主要時間,且使用MPI_Barrier去穩定process一致

是使用助教提供的testcase8 共有65537筆資料

結果如下

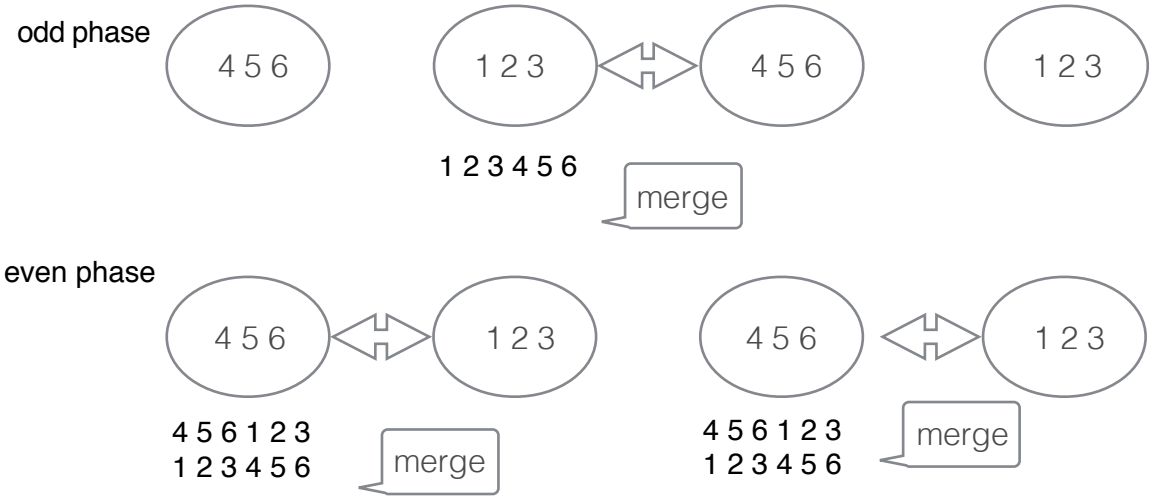
BASIC

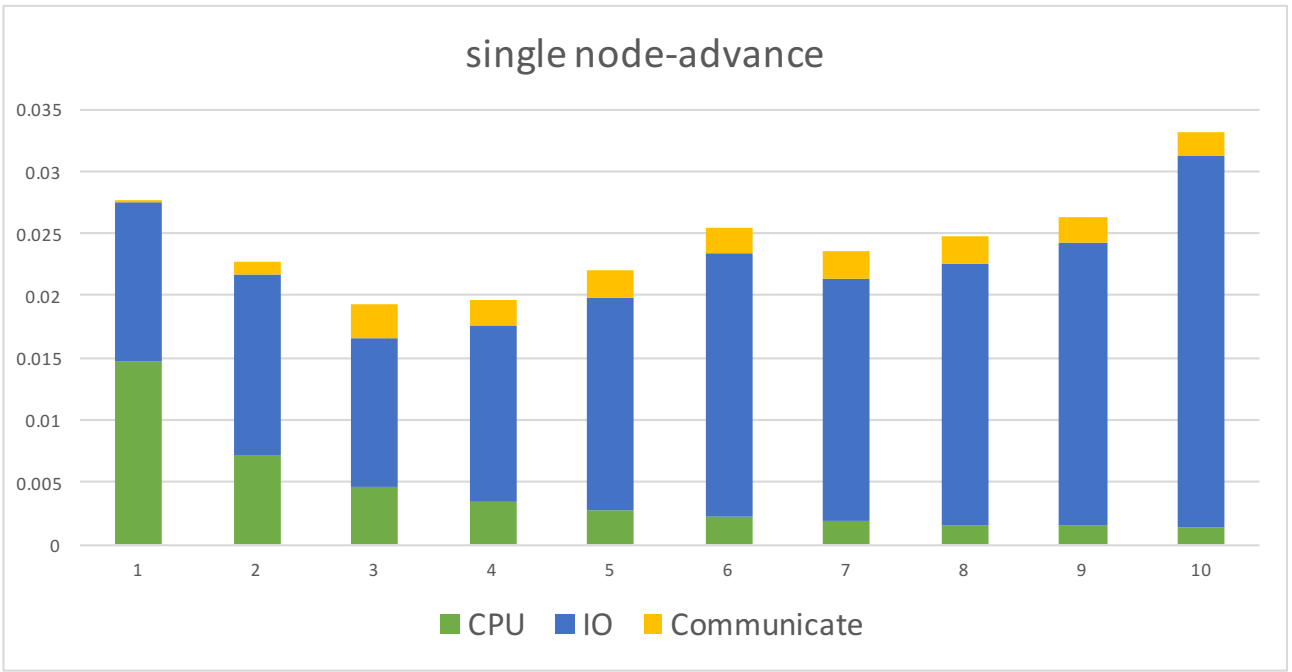


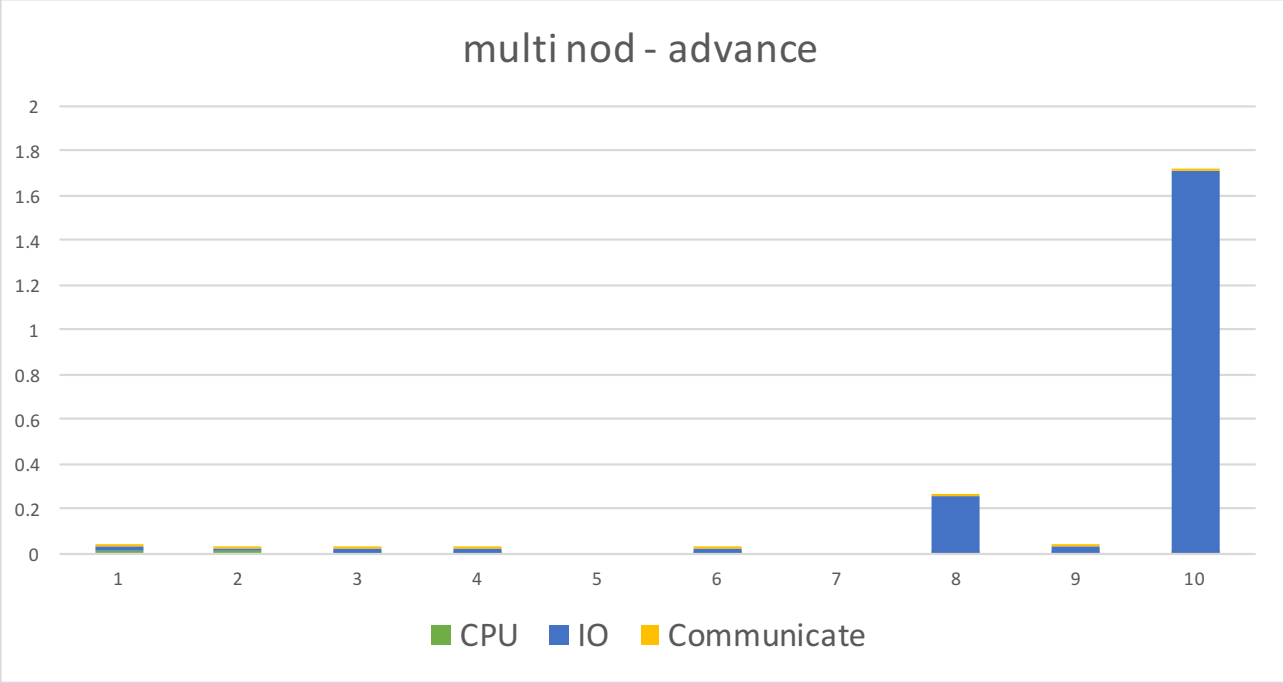


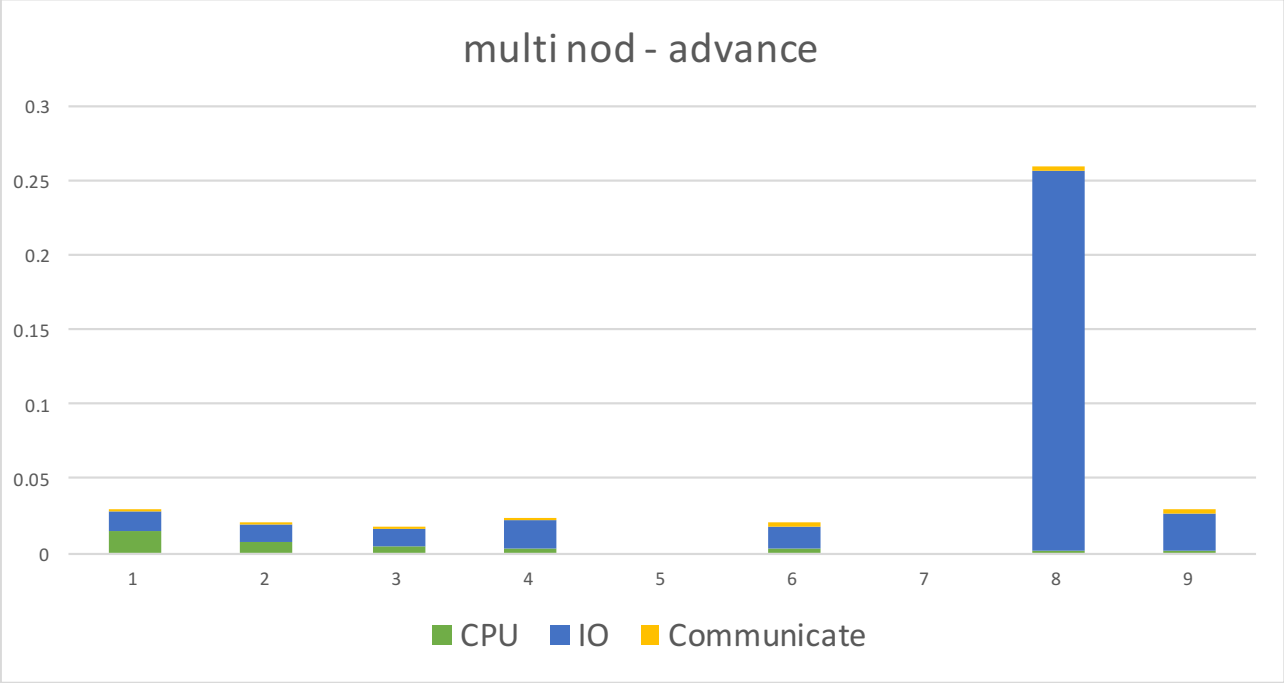
Advanced：

advance主要是先用quick sort,然後再透過oddevensort的架構來做merge,因為已經被sorted的數字可以直接互相做merge而不用再排序或swap,這樣可以透過大量數的傳遞來達到更快的速度.





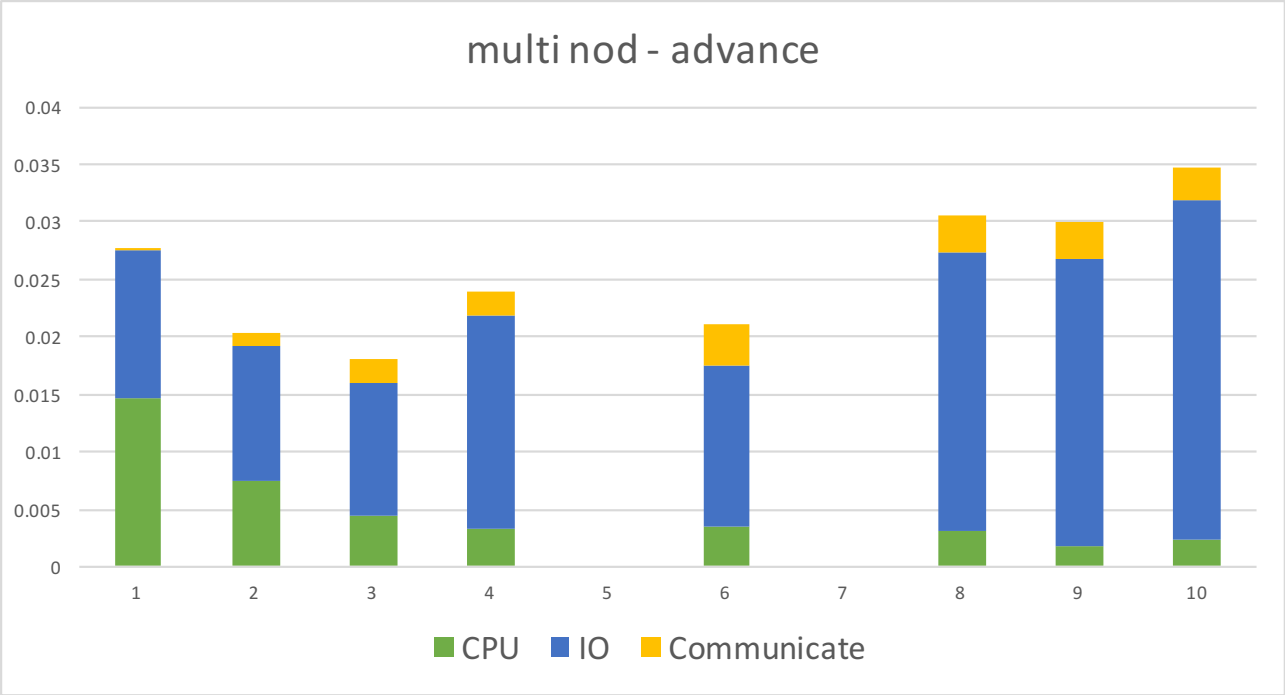




由上面幾張圖可得知,在所有情況底下的cpu時間是真的有跟著processor的數量提升,並且可以看到在multi node中的communicate的時間跟single node中的時間是有明顯的差別的,因為跨node傳送的時間一定比在node中的processor之間的傳遞還要長,所以當使用multi node時communicate的時間有顯著上升.

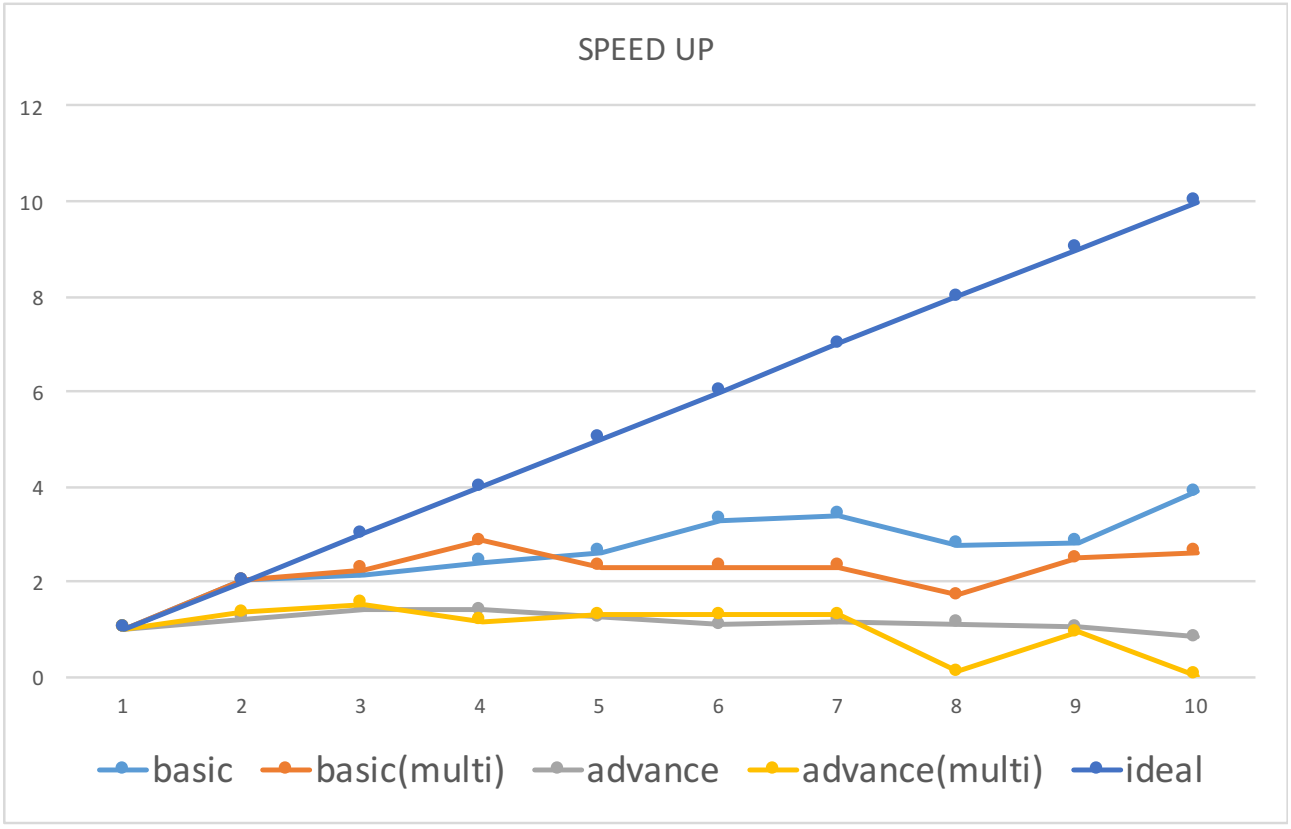
而IO部分就因為使用MPI_IO可能先把資料抓到buffer中,並且會受到node的loading所影響,由下圖可知:

這是在其他時間所測試的,可以發現IO部分的時間有大幅下降,因此可以猜測IO部分跟node上面的loading有關.

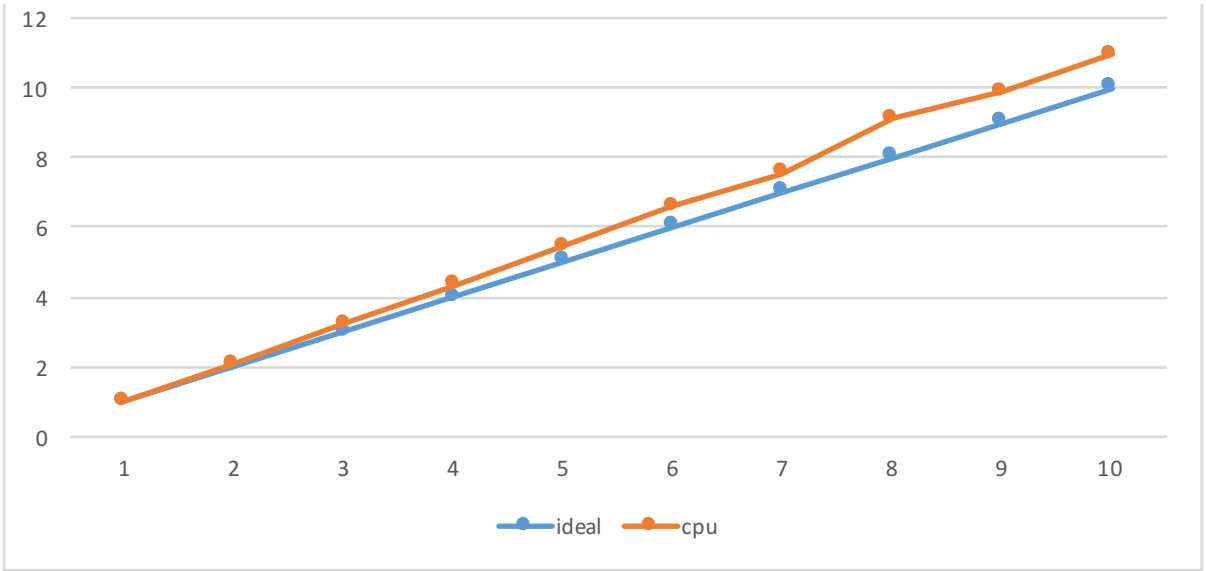


註：因為5跟7無法使用multi node因為無法使用超過5個node去跑,所以在multi node中沒有5跟7的資料

iii、 Speedup Factor

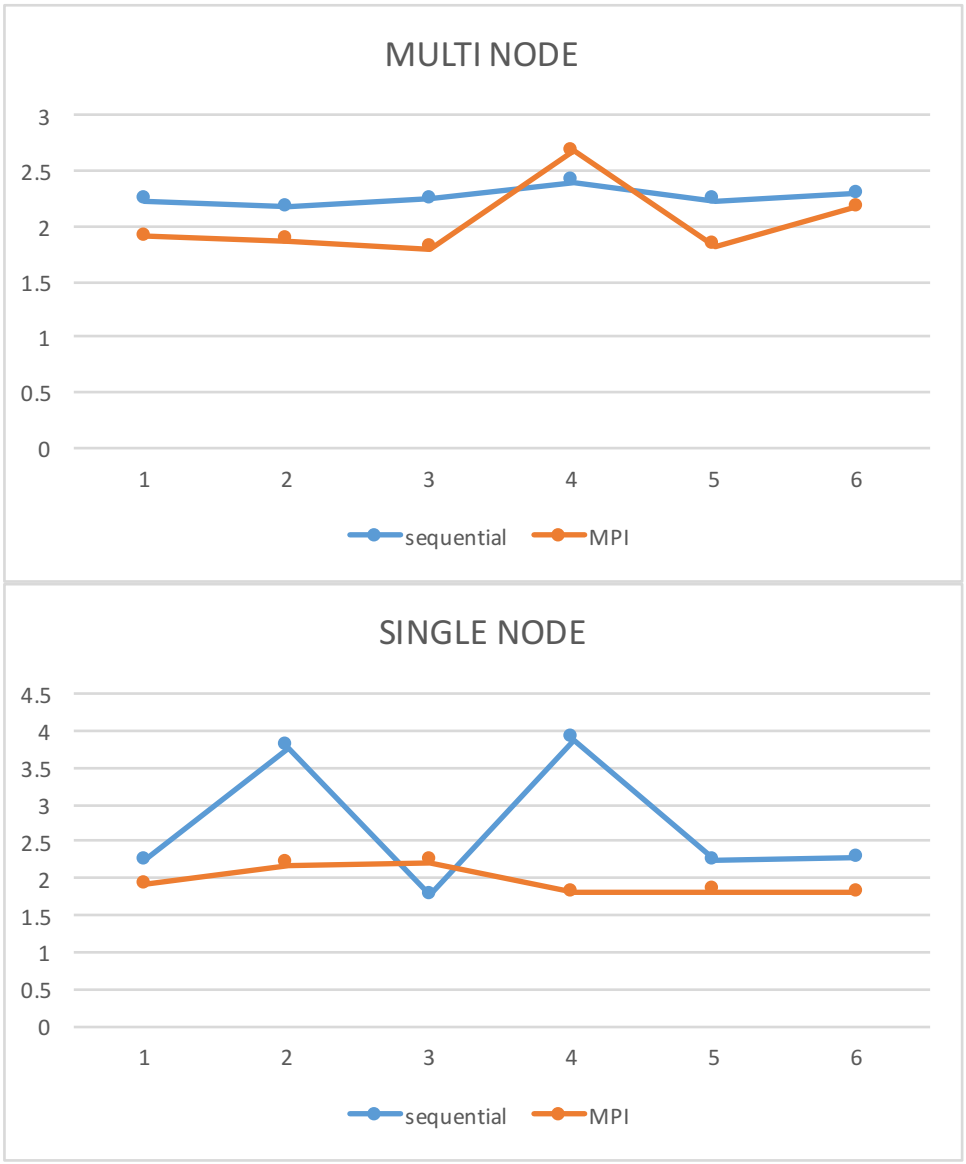


有這邊可以看出其實無法透過processor的提高來以倍數加快,這都是因為有溝通跟IO上的問題,當processor越多時,communication跟IOtime也都會跟著往上攀伸.



由上圖可知,若是單純比較CPU的執行時間,則可以發現倍數成長的能力幾乎超過ideal,所以當IO跟溝通時間往下降的話,可以提升平行運算上很大的效能.

iv、 Performance of different I/O ways



在大部分情況MPI都比sequential還要快,只是效果似乎不顯著

方法： 先亂數50000000個數字的檔案出來,然後在使用MPI讀取,以及使用fopen跟fread跟scatter等方式來做讀取,結果竟然並沒有差很多.

結論：

若要使用平行化的話就要考慮到每個process的load跟互相溝通時間,因為有時候因為減少運作時間但卻增加了溝通或是IO時間導致比沒有平行的效能還要糟糕.