Homework 2: Mandelbrot Set

Name：卓榮祥　　student ID：111062678

➢ Implementation

pthread version：

create 出的 thread 數量 = height size(number of points in the y-axis for output)，每個 thread 需處理一個 row 的 data，最由 main thread 將 Data 寫入 png 中。

Ex：若 png 圖片 size 為 200 *100，則 create 出 100 個 thread，每個 thread 處理該 row 的計算。透過 thread concurrent 執行減少 execution time、increase scalability。

Hybrid version：

利用 Dynamic Task Assignment 的機制減少 execution time 和 increase scalability，master thread(在 rank0 中)先將須處理的 row number 利用 message passing 傳給每個 process，每個 process 一次處理一個 row，且會利用 openMP create 出 thread 來進行計算，若有任何的 process 計算完成，則會向 master 提出 request，請求 master 傳送新的 row number 給他，若無工作可做，則結束。

Ex：若 png 圖片 size 為 200 *100，且有三個 process，則 master 先分配 row1 給 rank1，row2 給 rank2，row3 給 rank3，假設 rank1 先做完該 row 的計算，則 rank0 request 後 master 會再分配 row4 給它，以此類推，直到 master 沒工作可給為止。

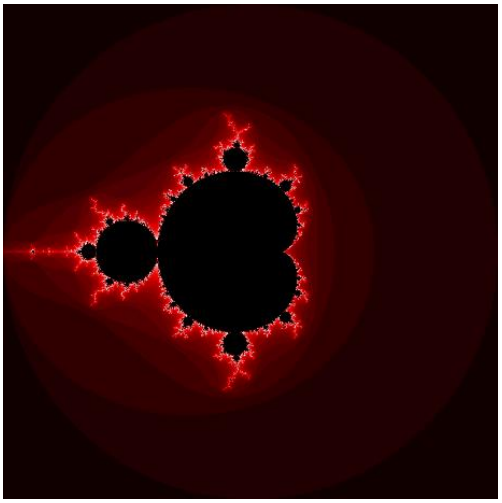
➢ Experiment & Analysis

Performance Metrics：

利用在程式碼中插入 **clock_gettime(CLOCK_MONOTONIC)** 的函式測量 run time

Experiment data set：

$iter	$x0	$x1	$y0	$y1	$w	$h
18667	-2	2	-2	2	2000	2000

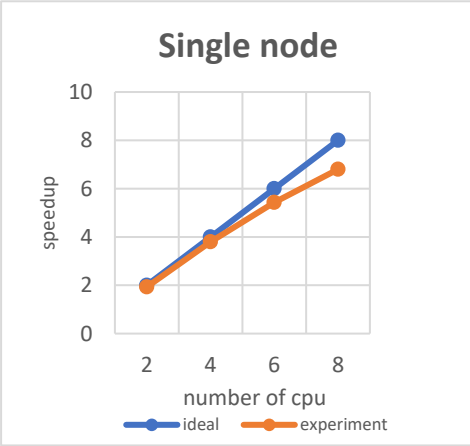


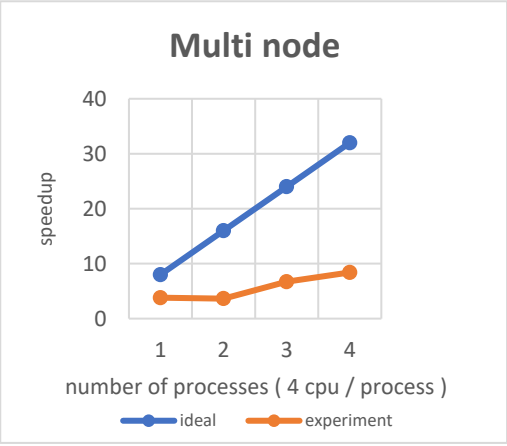
圖中有些 row 的黑色部分明顯較多，有些部分較少，因此 row 之間的計算量差距可以很大，且程式 run 的時間不會太長導致運行太久，也不會短到看不出 experiment 的執行差異，適合用來做 experiment。

Performance Metrics：

■ Strong scalability：

Hybrid version



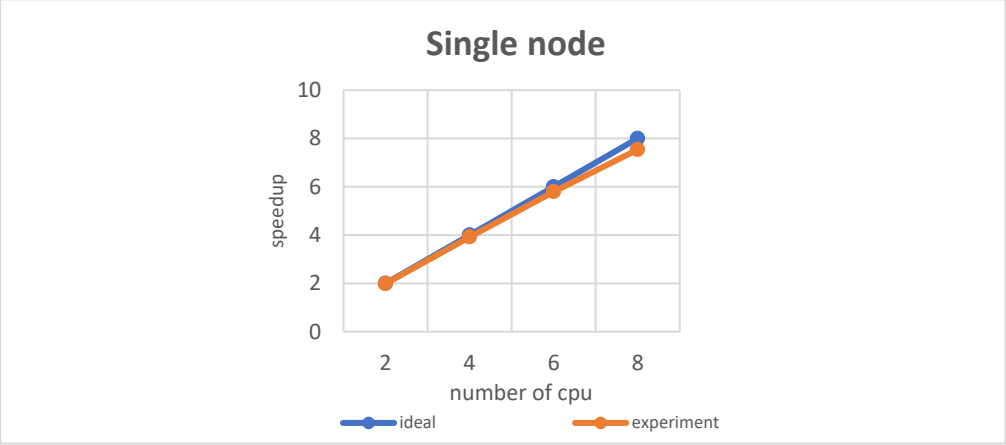


在 single node 中，cpu 越增加的話程式的 speed up 和 ideal 的 speed up 相差越來越多，因為隨著 CPU 數量的增加，但將 data 寫入 png 時並不是平行執行，成為了平行運算的 bottleneck，使 scalability 稍微差些。

而 multi node 中，我覺得 scalability 不太好，原因是在 1 個 process 到 2 個 processes 的 speedup factor 有稍微下降的趨勢，而往後才開始緩慢上升，且和 ideal 的 speedup factor 差距還是很大。

在實作過程中，在 rank0 master 進行 schedule 時，雖然是動態分配 task 給其他的 rank (worker)，但為了要讓 rank0 自己也能處理 data，於是我只讓它處理固定的量，並沒有做到 dynamic scheduling 的部分，導致 scalability 不佳，若是能讓 rank0 也參與 dynamic schedule，則相信 scalability 會變得更好。

Pthread version



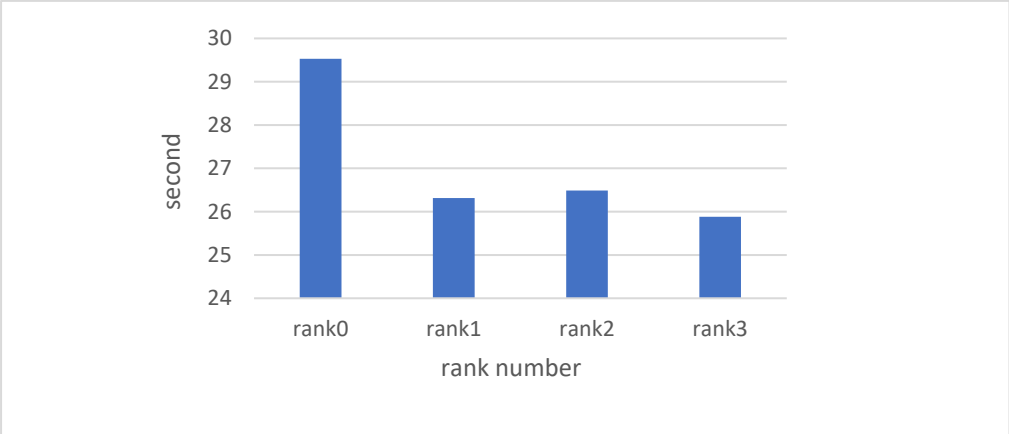
在 pthread version 中，我覺得我的 scalability 對於這個 Experiment data set 還算不錯，但除了將 data 寫入 png 檔的 overhead 外，每個 worker 的 loading 不平均也是導致 speedup factor 跟 ideal 有些微落差的原因。

■ Load balance (for Hybrid version) :

Mathod：計算每個 rank 的 run time 看是否 load balnce

為了更看清楚大資料量下時間的差距，將$w = 8000，$h = 8000

Number of process：4	Number of CPU / process：8

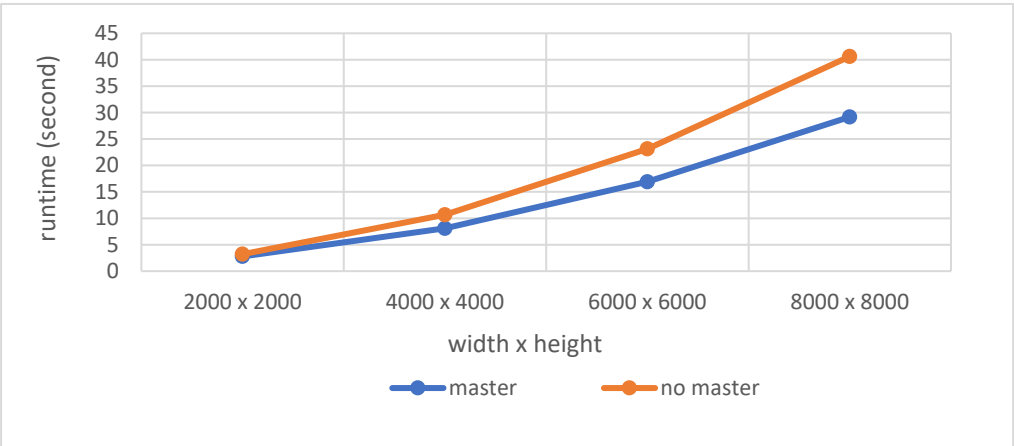


圖中 rank0 的部分和其他 rank 相比之下時間高了不少，因為需要負責 schedule task，且須是等其他 rank 都結束計算時才結束 scheduler 的工作，且又需將 data 寫入 png 中，多增加了一些 overhead，形成 bottle neck，因此認為 load balance 不太好。

■ 有 master scheduler vs 沒有 master scheduler (for Hybrid version)

Method：對於相同的 Data，比較有利用 master scheduler 和沒有 master scheduler(也就是將資料 static 分配給每個 rank 執行) run time 時間的差異。

Number of process：4	Number of CPU / process：8

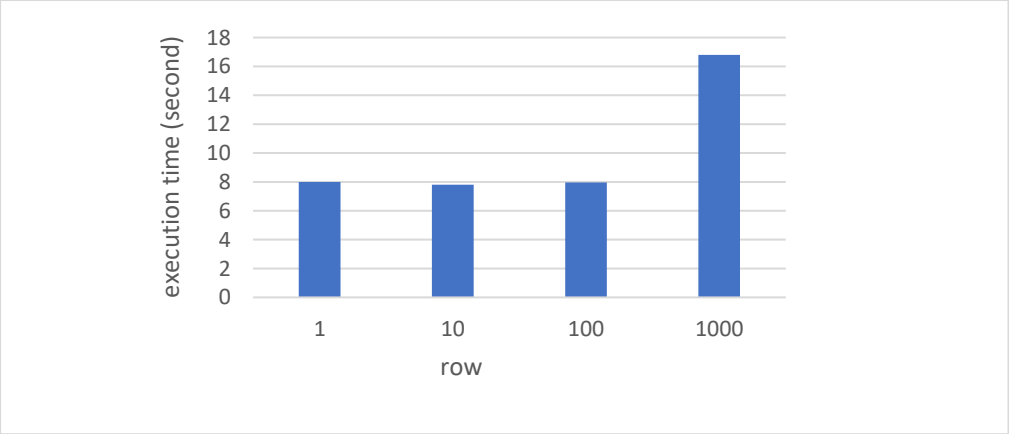


上圖隨著處理的資料量越多，時間差距越來越大，有 master scheduler 能夠提升 load balance，node idle 的時間較短。若無 master scheduler，則乍看之下每個 node 工作量是相同的，但是計算量差距可能會很大，可能造成許多 node 花大量時間等待還在處理工作的 node，導致 load balance 大幅下降。

■ Master 一次需配置多少資料量給 worker 做? (for Hybrid version)

Method：提高 row 的分配個數觀察時間的差距
Ex：row = 10，意即一次給 10 個 row 供一個 worker process 計算
為了更看清楚大資料量下時間的差距，將$w = 4000，$h = 4000

Number of process：4	Number of CPU / process：8



上圖中若 master 一次讓 worker 計算十個 row 的 data 量，則 execution time 為最高，原因為每個 worker 都分配了很大的工作量，且每個工作的計算量差距可以到非常大，因此有些 worker 需要執行更久讓其他的 worker 等它，造成 load balance 非常差。

➢ Experiences / Conclusion

之前有碰過 pthread，而覺得現在新學的 openmp 非常方便，尤其是將 nested for loop 進行 partition 的部分，再寫這次作業的期間，發現自己對 thread 的性質、API 的利用還不太熟練，參數的傳遞也寫得很卡，最後真的是一步一步研究才把作業寫出來。

這次的作業除了要把程式平行化，更要學習如何分配工作，因為自己出於好奇寫了

static schedule task 跟 dynamic schedule task 的 hybrid version code 才發現其中的差異，若每個 task 的 execution time 不一定相同，則利用將 task dynamic assign 給每個 worker 提升 load balancing 可大幅降低 execution time。

雖然自己對程式執行結果不是很滿意，還有許多需要改進之處，像是 Hybrid version rank 0 動態分配給自己的 worker 的 code 還是沒有寫出來，但這次作業讓我學到了 dynamic schedule task 的技術，對於提升 scalability、load balance 不失為一種好方法。