HW2 Mandelbrot Set

109062101 許佳綺

Implementation

以下會將在pthread及hybrid段落中提及data partition及其他optimization(含reduce time及increase scalibility)。

Pthread

首先，直觀的使用`pthread_create`及`pthread_join`去create thread，由於這次作業講究load balance，並且每個pixel的計算量都不太一樣(可能某段區域的計算量特多或特少)，想讓每個thread均分mandelbrot set的工作量，我以很簡單的跳row方式去partition data，將`numOfThread`設為`cpu_per_process`數量，讓所有thread每間隔`numOfThread`個row就拿一條row去計算，直到height條row都被拿完，如此一來每個thread拿到的工作量會差不多，可以不需使用mutex lock維護整個image，而每個thread能同時計算且互不衝突。而這樣的作法同時也能保證在scale變大的狀況下，更能夠避免出現某個thread的loading a很重導致speed up效果不好的問題。

optimization的部分我實作做了vectorization去加速，我使用SSE4的`__m128d`，對每個thread而言，拿到一個row時我會讓他兩個兩個pixel一起做，並且如果有一個pixel先算完，則繼續取出下一個pixel放入`__m128d`中一起vectorization。如若其中一個pixel在拿下個位置的時候發現已經做完該row了，則跳出迴圈把剩下的pixel做完。另外，在實作vectorization時發現，如果可以避免在很常走到的地方放置if-else，則可以大幅減少時間，像是在做完整條row的pixel要跳出迴圈時，可以不用把條件放置在每次的最外圈while，而是在檢查要拿下個pixel的人時判斷現在是否已經做完即可，還有可以減少一些重複計算的數值(像`x*x`)等等的reduce execution time。

Hybrid

沿用Pthread的直觀想法，我也是跳row partition data，這邊將換成是每隔`numOfProcess`個row取一行給一個process，在每個Process中，我將`numOfThread`設為`cpu_per_process`，使用openmp的dynamic schedule加速計算每個process所拿到的那行row的pixel，如此一來只要有thread做完就會馬上去拿下一個pixel來計算，加速整體運算速度。此外，我在每個process中紀錄一張完整大小的圖片去儲存每個process跳row取的結果，最後使用`MPI_Reduce`的`MPI_SUM`得出整張image，而這邊由於每個process一定不會重複取到row，因此使用`MPI_SUM`將獲得所有計算後的數值並不會被任何覆寫。

optimization的部分和pthread部分相同，我實作做了vectorization去加速執行。另外，我在較不常走進入的if-else上加入了unlikely，讓compiler 幫忙將assemble code順序調動而優化速度。

Experiment & Analysis

Methodology

1. System Spec

   使用課程所提供的 apollo server。

2. Performance Metrics

   a. Pthread: 使用 `clock_gettime(CLOCK_MONOTONIC)` 計算時間。

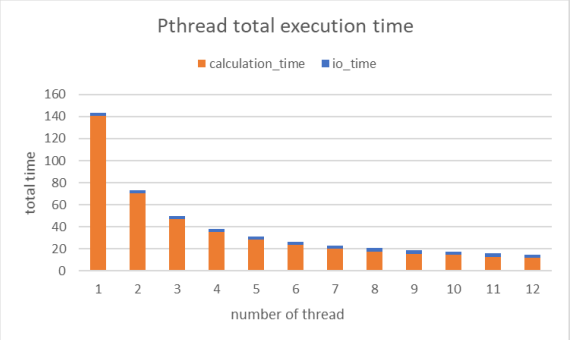
   b. OpenMP+MPI: 使用 `MPI_Wtime()` 來計算時間。

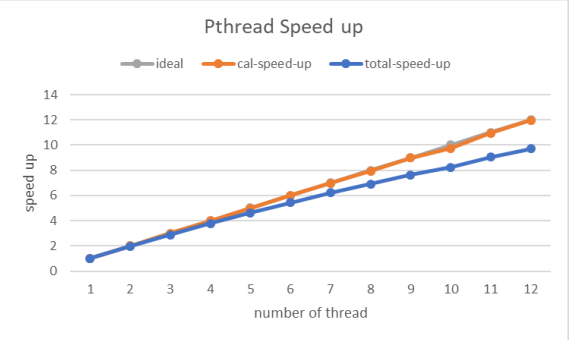
Plots: Scalability & Load Balancing & Profile

用strict21.txt測試

Pthread

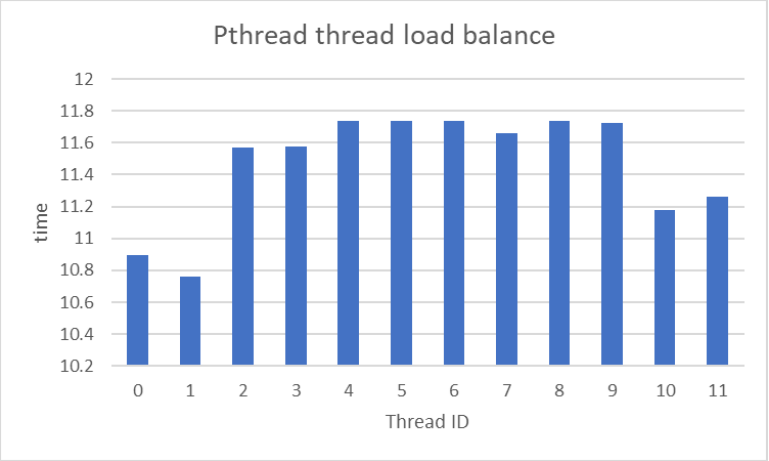
以下是在process=1, cpu_per_task(thread)=1~12的狀況下測試的。





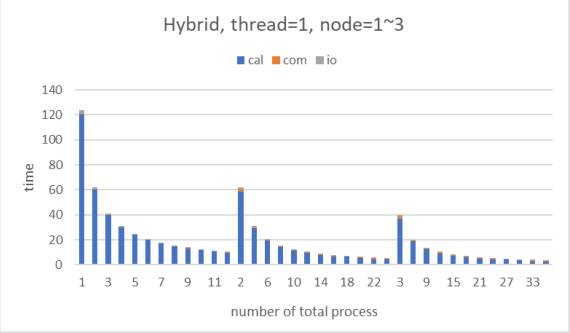
首先，左邊這張圖是Pthread版本的profile，可以看到calculation time幾乎占滿多數執行時間(是主要的bottle neck)，在thread=1時的total time會是140秒左右，但經過parrallel 加速後，隨著thread數量上升，execution time有非常顯著的下降，而透過右邊speed up圖也可以看出scalability非常好，calculation speed up近乎是貼著ideal speed up線在向上增長。

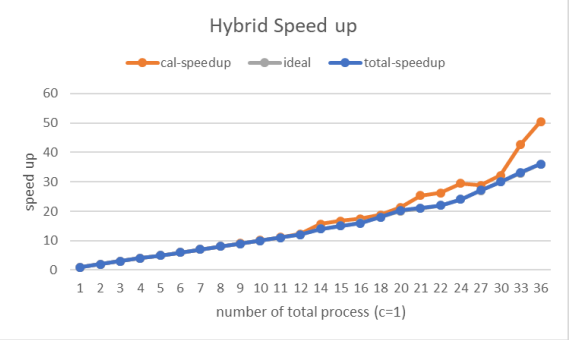
接著，以下這張圖是Pthread load balance的測試，我將cpu_per_task設為12，觀察每個thread的運算時間，發現以跳row partition data的方法能有效讓work load很平均，每個thread的執行時間差距不會超過0.1秒。



Hybrid

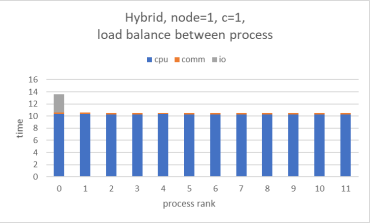
以下將實驗將會分成`cpu_per_task(thread)`數量固定，提高node數及process數量測量scalability，以及探討load balance 的結果。

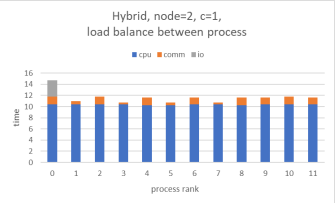


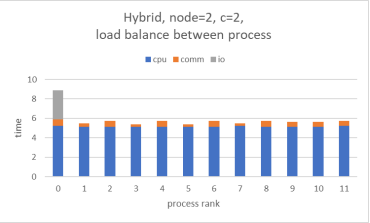


首先左邊這張是固定`cpu_per_task(thread)`=1，將整體的node跟process數量提高，也可以很明顯地看到執行的時間會大幅縮減，右邊的speed up 可以看連total speed up都是貼著ideal line前進，calculation speed up甚至會super speed up。

而以下的圖分別探討在固定node數、process數及`cpu_per_task`的情況下Hybrid版本的load balance會如何。可以看到以下三張圖的rank0都有多一塊灰色的IO時間，是因為image就是在rank0寫入的。接著先看左邊兩張圖，可以發現不管有無跨node，所有Process的calculation time 差距不超過1秒; 而觀察右邊兩張圖可以發現，在node=2的狀況下即使process的thread數量不同，每個process2都還保有著良好的load balance。而觀察第一跟第三張可以發現，縱座標的執行時間，在node=2, cpu_per_task=2時會大概是node=1,cpu_per_task=1的0.5倍，也相當符合效能提升的ideal構想。







Experience & Conclusion

這次的作業讓我學到很多，對openMP的不熟悉，要直接混用openMP跟MPI時在是有些複雜。此外vectorization的部份我也花了蠻多時間去研究怎麼寫code才會比較快速，這次作業中有好多地方都是只改一兩行code，我覺得”可能應該差異沒這麼大吧”結果跑出來成果會差超多的XD 這也讓我重新審視平行程式的的觀念及撰寫邏輯。另外想提的是，再加上vectoriation後有時候過了hw2x-judge時以為應該都沒問題，殊不知居然會有一些微小的錯誤導致圖片精準率不會是100%而是99.9%或是99.8%，然後要非常仔細去檢查圖片輸出及程式碼，會發現一些導致精準度下降的問題。

總之，非常感謝助教及教授用心準備這次的作業，真的獲益良多，也讓我發現自己的不足並努力去改善。