

Parallel Programming HW2 REPORT

109062671 吳竣宇

1 Implementation

a. Pthread version:

我主要是對圖片中的多個pixels做Plot the Mandelbrot set的多次iteration 實做平行計算，首先我將tasks partition為 height * width pixels，令變數 pixel_count 做為shared variable後，利用pixel_count獲得該pixels在圖片中的座標(i, j)後讓threads一次取得 2 個pixels進行plotting，並使用 vectorization的技巧增加了平行度，最後將image存進png檔結束程式執行。

在threads取得pixels時因為是以先完成工作先取的方式進行，所以相對於在程式一開始就分配好每條thread去執行哪些pixels來說，可以有效減少執行時間，並增加scalability。

b. Hybrid version:

我對圖片中的多個pixels做Plot the Mandelbrot set的多次iteration 實做平行計算，首先我將resources提供的processes分成Master process和 Slave processes，Master process負責將tasks partition成 height rows，令變數 row_count 作為shared variable後，Master process會把row傳給 Slave processes進行plotting，plotting的過程中，除了一樣使用 vectorization的技巧增加了平行度以外，再利用openmp對for迴圈的平行化處理做dynamic schedule，讓threads可以對for迴圈做切割後做平行計算，Slave processes plot完一行row後，即將計算結果利用row buffer傳回Master process，把row buffer複製到image後，讓Master process send 尚未完成plotting的row給剛完成一個row task的Slave process繼續做下一輪的plotting，同時Master process利用openmp的section也會分配row，一邊傳送接收row buffer，一邊進行plotting，當所有row都被分配到不同的processes後，用變數execution_slave_processes去確認是否所有 Slave process都已完成plotting的工作，並回傳row buffer到Master process，當execution_slave_processes為0時，Master process會傳送變數flag給Slave processes讓他們結束執行，最後在Master process將 image寫入png檔後結束程式執行。

在threads取得row時因為是以先完成工作先取的方式進行，所以相對於在程式一開始就分配好每條thread去執行哪些rows來說，可以有效減少執行時間，並增加scalability。

c. Other efforts:

- 我嘗試過一開始平均分配rows給不同的 threads，當有thread完成工作後即去平均分攤還沒做完工作的thread的工作量，不過這個方法最大的好處只有每個 thread/process 各自plotting的pixels幾乎是相鄰的，其他像是locality之類的好處在先搶先拿的code上也有，因此中途放棄了implement這個方法。

2 Experiment & Analysis

i. Methodology:

(a). System Spec:

All the experiments were executed in Apollo.

(b). Performance Metrics:

- pthread version:

gettimeofday() in <sys/time.h>.

- hybrid version:

MPI_Wtime() in <mpi.h>.

- Measure method:

- CPU time: total execution time - IO time - Communication time.

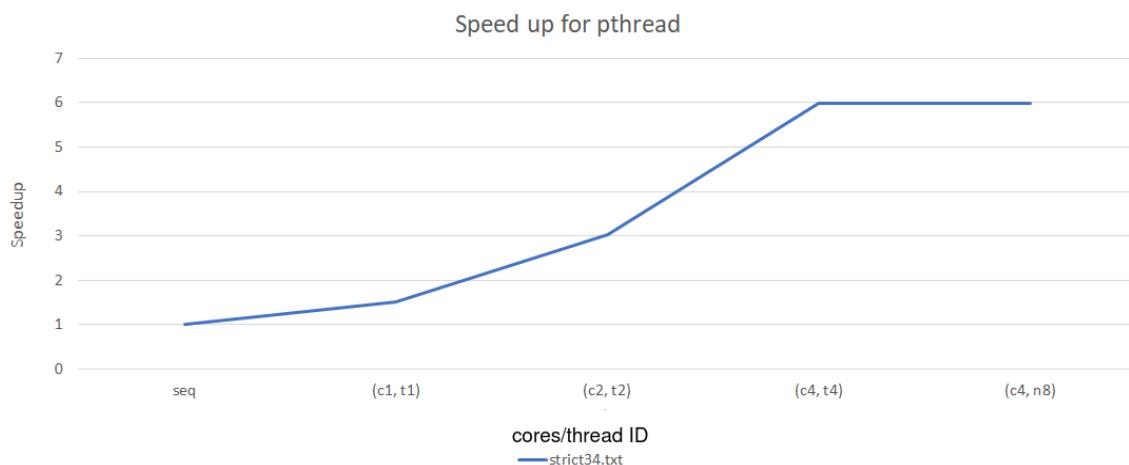
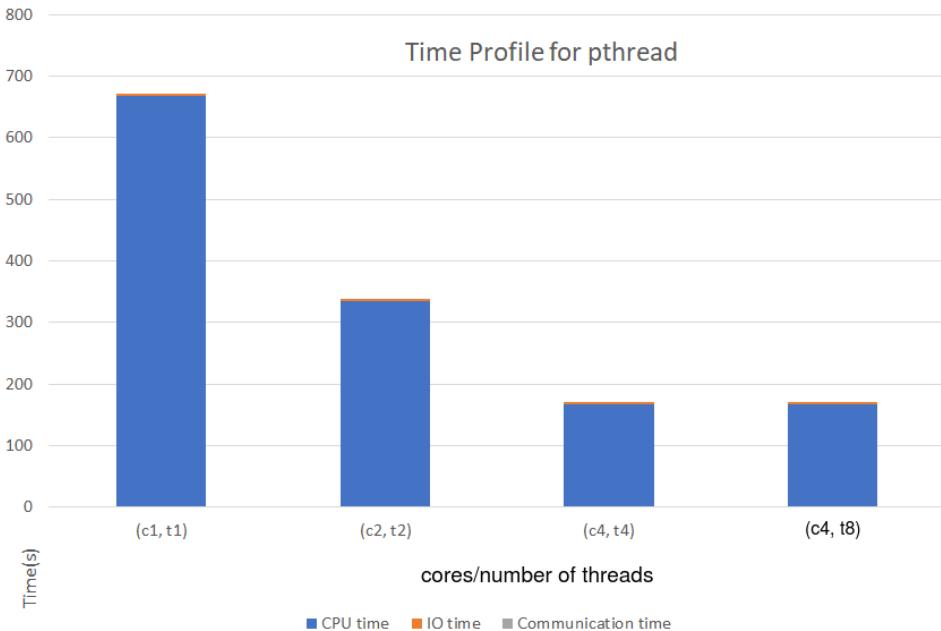
- IO time: write_png() time.

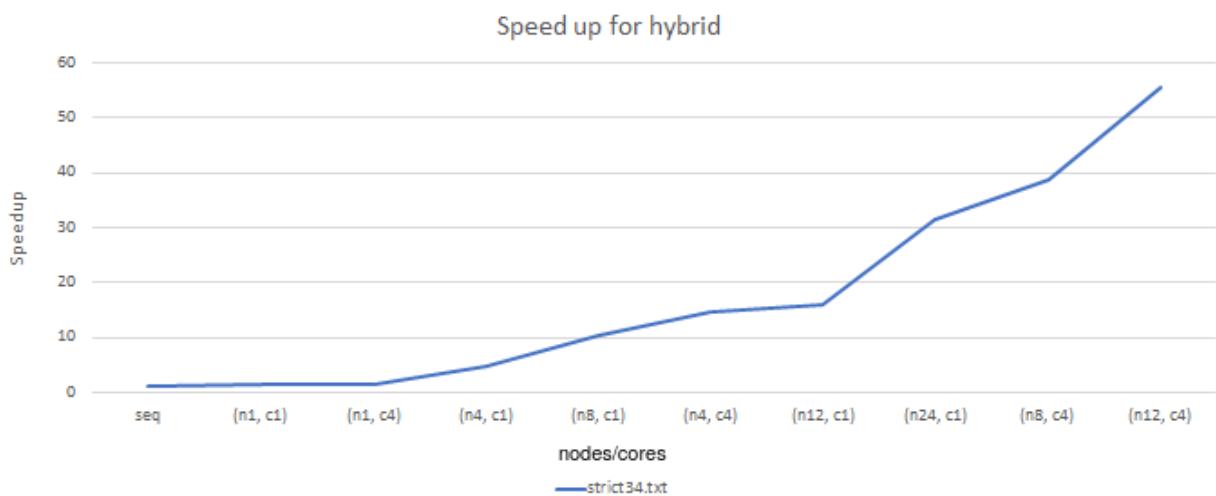
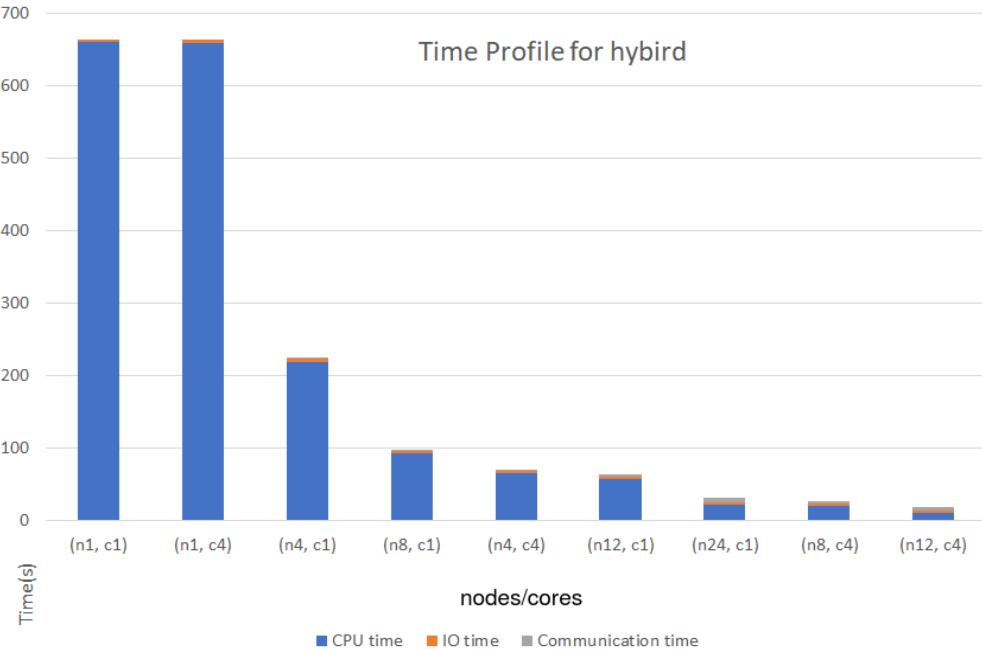
- Communication time: the sending/receiving tasks time in hw2b.

ii. Plots: Scalability & Load Balancing

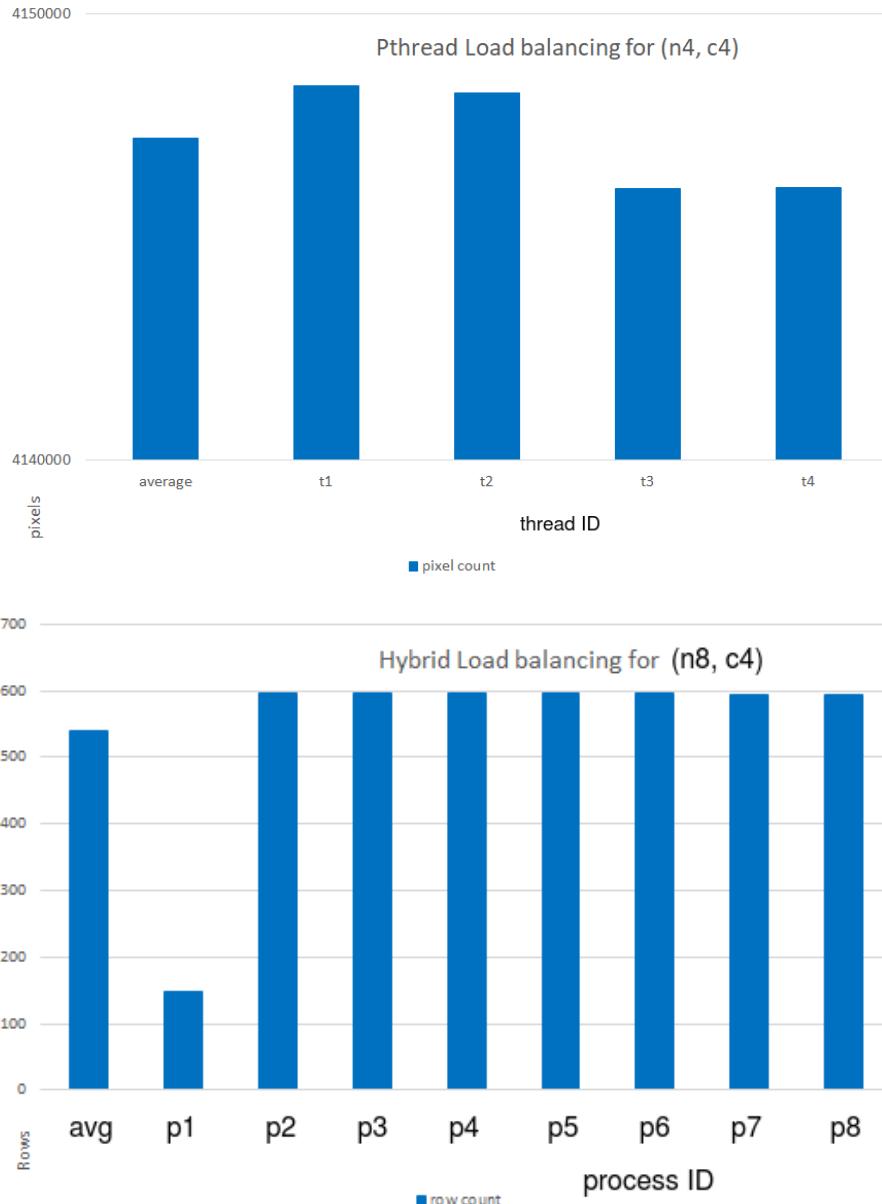
(a). strong scalability experiments:

- I made the time profile for pthread version and hybrid version of hw2, and speedup for pthread version and hybrid version of hw2:





(b). Since all threads/processes terminate in the same time nearly, I assume they have a good load balance for their tasks. However, I choose to count the number of tasks (pixels or rows) they have finished, and find that the throughput of the master process in hw2b is lower than other slave processes.



iii. Discussion:

(a). Scalability:

- 藉由vectorization的輔助之下, 即使使用1 node 1 core跟sequential版本的code相比, 也能加速到1.5倍, 在pthread版本的code中沒有communication time, 在hybrid版本的code中當master process是做一對多的communication, 所以若使用很多processes, communication time / total time的比例就會很高, hybrid版本的code我覺得scalability很好, speedup跟nodes/cores的乘積比斜率很接近1, 若是有更多時間會嘗試看看其他種vectorization的方法。

(b). Load balance:

- 基於先做完task的thread/process優先搶下一份task的原則, 即使每個thread/process完成task的時間差異極大, 還是不會讓processes和threads有idle的時間, 因此達到很好的throughput, 不過load balance還是會

因為實現的strategy不同，而對總體效能上有不同程度的improvement，根據上3 ii.(b)的實驗我們可以發現，在hybrid版本的plotting因為有openmp的dynamic schedule輔助，每個process完成的task數量明顯較pthread版本的threads分配的更為平均，不過實驗後才發現我的master process的throughput為其他processes的1/4，我猜測原因是master process應該在使用兩條threads分別執行section時，原本應支援plotting計算的threads就沒有slave processes來的多，因此解決的方法應該是要額外再create thread讓我們有足夠的threads去支援分配工作給其他processes及執行plotting的計算。

4 Experience & Conclusion

(a). Experience:

- 對於load imbalance的解決方法有了新的認知
- 第一次使用registers做加速還滿新鮮的
- 查詢vectorization的指令花了不少時間
- 在pthread版本的code上卡了一個測資TLE卡很久，最後把share variable從row改pixel就過了，目前還沒想出理由。