Parallel Programming Homework 2: Mandelbrot Set

107060002 曹立元

Part I、Implementation

1. How I implement each of requested versions, especially for the hybrid parallelism.

(1) Pthread

實作 Pthread 時我先 create 出多個 thread 並透過自己設計的 compute 函式去做計算，compute 的實作方式等等會再提到，再來對每個 thread 呼叫 pthread_join，並在大家完成計算工作之後呼叫 write_png 把圖片做出來。共用的變數的部分我把他們都放到 global，以利大家使用 share memory 的方式去 access。

```
pthread_t threads[ncpus];
int rc;
int ID[ncpus];

for (int t = 0; t < ncpus; t++) {
    ID[t] = t;
    rc = pthread_create(&threads[t], NULL, compute, (void*)&ID[t]);
}

for(int i=0; i<ncpus; i++){
    pthread_join(threads[i], NULL);
}

/* draw and cleanup */
write_png(filename, iters, width, height, image);
free(image);
pthread_exit(NULL);
```

(2) Hybrid (OpenMP + MPI)

實作 hybrid 版本方面，若是 scale 比較小的 task（我定義為 width < 100 && height < 100），我會只使用單一 process 並利用 OpenMP 平行化的技巧去進行運算。若是 scale 較大的 task，我就會把 task 分給各 process，再利用 OpenMP 讓各 process 之間的任務也可以分給各 thread 達到平行化。

2. How I partition the task?

在 task partitioning 的部分，在 Pthread 或 hybrid 中我都是一次以一個 row 為單位來送給各 thread 做計算，其餘運算的部分在 Pthread 或 hybrid 中大同小異，後面會再介紹計算的方式。

(1) Pthread

在 Pthread 的部分，我設了一個 global 變數 count 來讓所有 thread 知道現在算到哪一個 row，並各自存在 local 變數 j 中，j 就代表他現在要做第幾個 row，每 assign 一次 j 的值的時候代表又有一個 row 被人拿去算了，所以 count 的值也要加一。而因為怕有 race condition 發生，我把

更改 count 的值的動作放在 critical section 中去操作。剩下的就是單純針對每個 pixel 做計算的部分。

```
int j;

while(count < height){
    pthread_mutex_lock(&mutex);
    j = count++;
    pthread_mutex_unlock(&mutex);
// for (int j = *tid; j < height; j += ncpus) {
    double y0 = j * ((upper - lower) / height) + lower;
```

(2) Hybrid

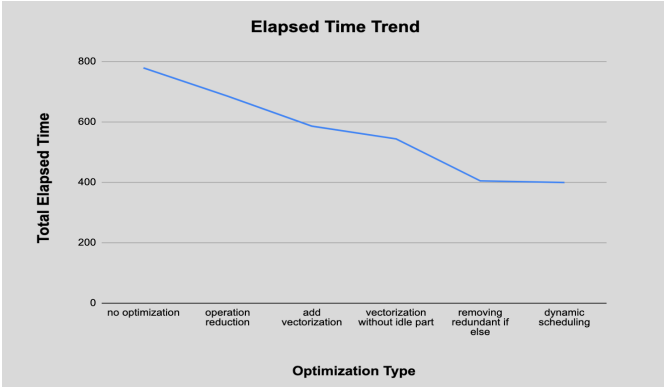
在 Hybrid 的 task partition 部分，我則是先 assign 固定的 rows 給某個 process（第 i 個 process 就處理第 i、i+size、i+2*size、……個 row），再對於每個 process 內採用 dynamic scheduling 的方式去做平行化。

```
#pragma omp parallel for schedule(dynamic, CHUNK)
    for (int j = rank; j < height; j += size) {
        double y0 = j * ((upper - lower) / height) + lower;
```

3. What technique do you use to reduce execution time and increase scalability?

我先對於原本 sequential code 的算式進行化簡，光做完一點化簡跑出來的時間就比沒化簡時快了約一百秒，我再利用了 vectorization 的方式進一步做優化，type 為 __m128d 的變數可以一次將兩個 double 存在裡面，所以等於一次可以計算兩個 pixel，整體速度便提高許多，而原本我在實作 vectorization 時是讓兩個 pixel 為一個 pair 去做，但因為某個 pixel 可能成為 bottleneck。所以我後來改成只要當其中一個 pixel 做完時，便會馬上 assign 另外一個 pixel 進到 128bit 的變數中取代掉原本的變數並進行計算，又省下了不少時間。

再來是原先我的 code 寫的比較爛，在 while 迴圈裡面還有很多判斷式，讓整體速度變得很慢，後來我就優化了一下我的 code，儘可能在 while 裡面只要做純計算的動作就好，速度也獲得明顯提升，下圖為我在 Pthread 中加上各種優化之後的 total elapsed time 趨勢圖。



在 hybrid 的實作部分基本上和 Pthread 使用的方法相同，在 scheduling 的部分我原先是套用 guided scheduling 的方法，但發現效果沒有很好，後來想到 Mandelbrot Set 這種一個一個 pixel / row 去處理的方法應該直接用 dynamic scheduling 的方法，並把 chunk size 設為 1 最好，我在 hybrid 實驗中當我把 scheduling 方法從 guided 改成 dynamic 後速度變快了約一百秒。

4. Other efforts I've made in my program

我的優化方式和過程大概都在上個部分講完了，只是不知道要怎麼在 report 中分割說哪些是減少 execution time 的方法哪些是 other effort，所以就都寫在剛剛的那部分了。

Part II、Experiment & Analysis

1. Methodology (Performance Metrics)

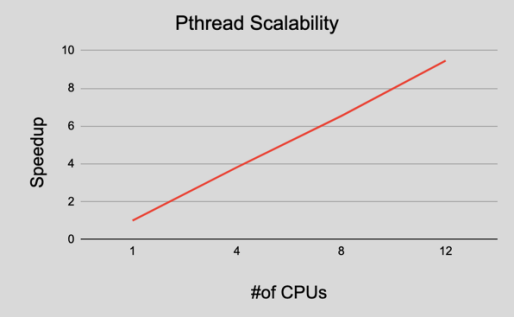
在計算時間的部分我選擇的是使用 clock_gettime 這個函式去計算，實作的 code 就和助教在 lab1 時給的相同，把要計算的區域用一個 start time 跟 end time 包起來再相減便可以得到待測區域的計算時間。測資的部分我是拿 slow01 來測，因為他的 iteration 數值設的特大，算是有參考價值的一筆測資，實驗結果我是跑三次取平均來得到最後的數值。

2. Plots: Scalability & Load Balancing && 3. Discussion (Must base on the results in your plots)

a. Scalibility

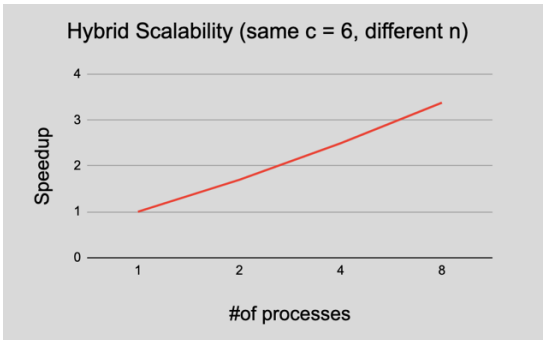
(1) Pthread Scalibility

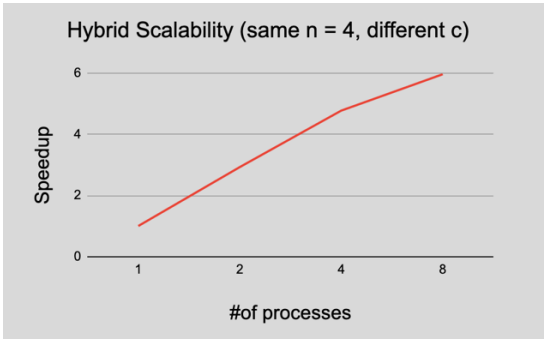
由圖可見使用 Pthread 實作時隨著 scale 增加，表現是呈現線性的成長，倍率約為(3/4)*(# of CPUs)，跟其他實驗比起來算是成長幅度很大的。



(2) Hybrid Scability

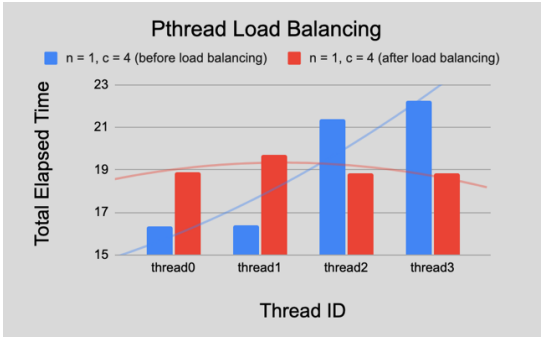
在使用 OpenMP + MPI 時，我進行了兩種實驗，第一種是固定 core 數量，並操作 process 數量，由左圖可看到隨著 process 數增加，Speedup 的倍率也隨之增加。第二種實驗是固定 process 數量，並操作各 process 擁有的 core 數量，由右圖可見隨著 core 數量增加，speedup 的倍率也隨之增加，且倍率比操作 process 數時還大，也可以發現兩種實驗皆展現了 hybrid 方法的 scalibility。

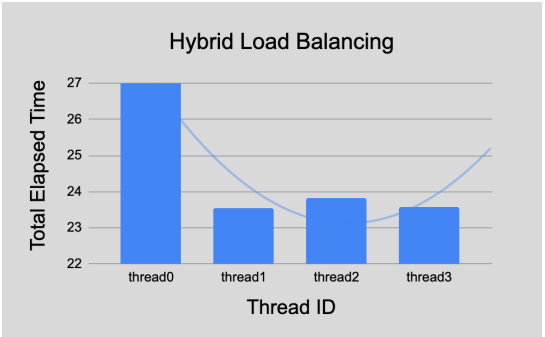




b. Load Balancing

下方左圖是我在 Pthread 的實作，在實作 Pthread 時我有利用 counter 來實作 load balancing，避免有 idle thread 的產生。藍色柱子為沒做 load balancing 時跑出來的時間，紅色則是有做 load balancing，可以發現我做了 load balancing 之後確實各個 thread 跑出來的時間都差不多，代表確實有做到 load balancing。而在 hybrid 實驗中我是選用 strict34 這筆測資來測試，由於我在 hybrid 的實作中並沒有做好 load balancing，所以可以由圖中發現 thread0 成為了 bottleneck，拖到了大家的時間，這方面還需要再改進。





Part III、Experiences / Conclusion

這次作業讓我對 Pthread 和 OpenMP 更加熟悉，也成功的實作了 vectorization，這是我在之前寫程式從未有過的經驗，同時也了解到了 load balancing 的重要性及效果。只是感覺應該還有不少地方可以再做優化，尤其是 load balancing 的部分，如果之後有時間我會想再來嘗試看看！