Parallel Programming hw2 ----Mandelbrot Set

學號：109062639　姓名：葉哲欣

Implementation:

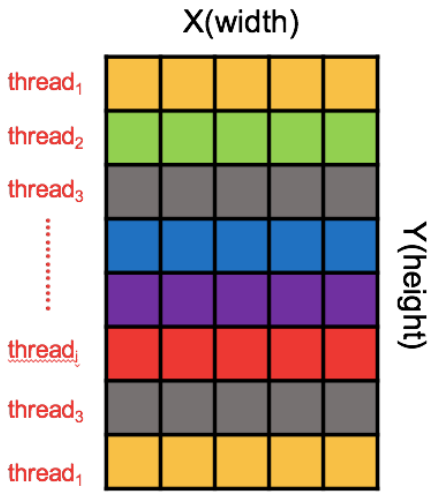
Pthread:

height: y 軸點數　width:X 軸點數　$thread_i$: 執行緒 I

count:目前已完成運算 row 數　image:存放運算結果

(a.) How do you implement each of requested versions ?

每個 $thread_i$ 負責一列(X,y_i),當運算完成後存入 image 並分配新的尚未運算的一列 data，直到每一列都運算完成。最終透過 main process 將結果輸出成 image。



(b.)How do you partition the task?

thread 分工圖如右:

讓每個 thread 負責一列運算

缺點：thread 彼此 context switch 與 mutex lock 次數過於頻繁，進而影響 cpu time

(c.) What technique do you use to reduce execution time and increase scalability?

若採用 Height/thread 來分配每個 thread 執行數量，會造成有些 thread 提早做完而有些 thread 仍在運算，無法達成 load balancing 。故採用每個 thread 執行一小部分的 task,當執行完時用 mutex lock 的方式來存取 count，若工作尚未分配完，則繼續分配工作。

Hybrid (open MP + MPI)

ntask :process 數　rank$_i$:process id　master:分配 task 給 process

slave:　執行 task　num_thread:每個 process thread 數

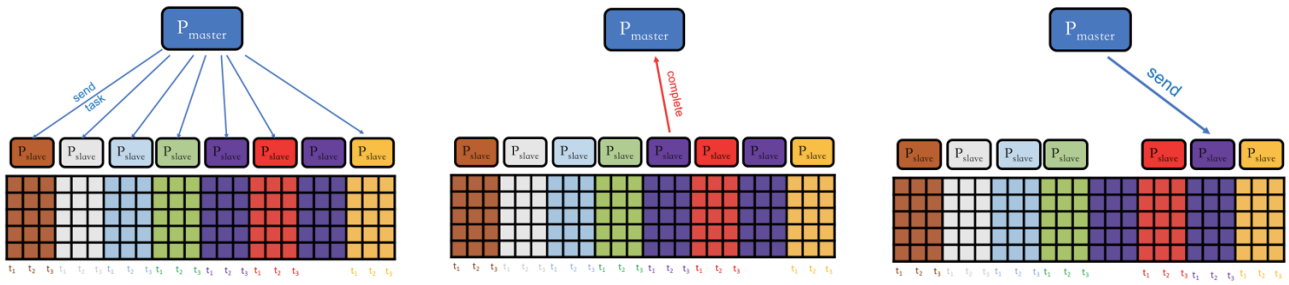
(a.) How do you implement each of requested versions ?

Master process 負責分配工作給 slave process,每個 process 分到 num_thread 個

task,當 slave 完成時將運算完結果回傳給 master，master 繼續分派任務，直到所有 task 完成。

(b.)How do you partition the task?

Process 與 thread 分配工作模式如下



(c.) What technique do you use to reduce execution time and increase scalability?

使用 master-slave 方式來分配 task，並且 openmp 使用 dynamic scheduler ，以及巢狀迴圈使用 collapse。每個 process 分配 c*num_thread 列數來平衡 context switch 與 process communication time ，降低 execution 時間

Experiment & Analysis:

I. Methodology

A. System Spec:

使用 apollo 機器

B. **Performance matric**

Time measure method:

使用 MPI_Wtime( )

使用 omp_get_time() 計算 thread 執行時間

Ex:

```
start = MPI_Wtime();
//compute
end = MPI_Wtime();
Time = end -start;
```

II. Plots: Scalability & Load Balancing

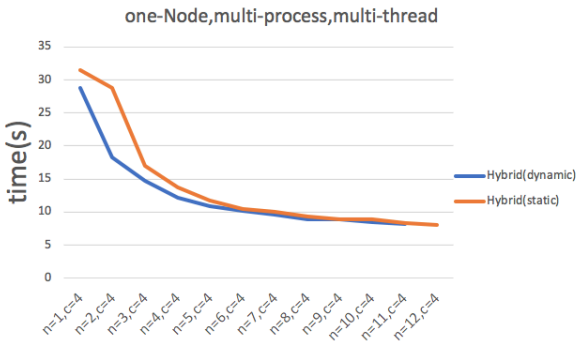
實驗採用 strong scalability

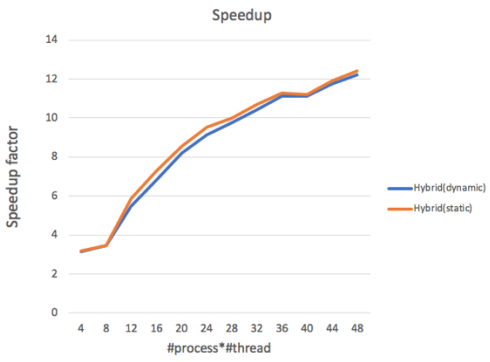
Coordinate range : X 軸----實數(-2,2) Y 軸----虛數(-2,2)

Iteration = 1000 Height = 10000 Width = 10000

(a.)Scalability & Speedup

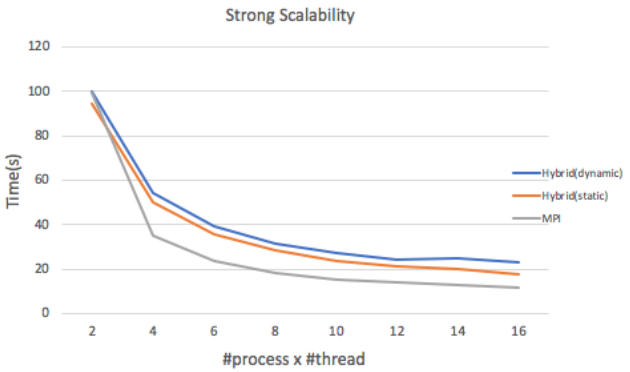
A. One Node,Multi-process,same-thread

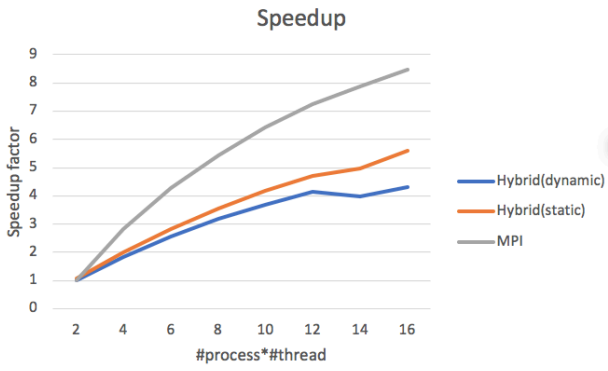




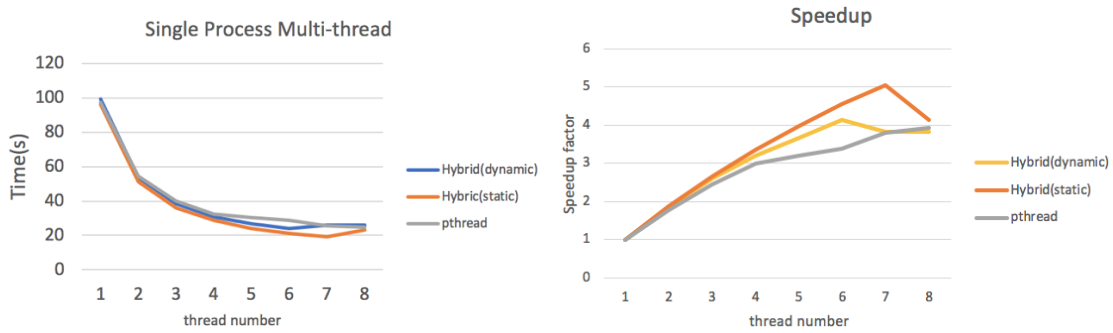
B. Same (#process＊ #thread)different approach.

#process:2.#thread:1~8 (MPI: #process 2~16)



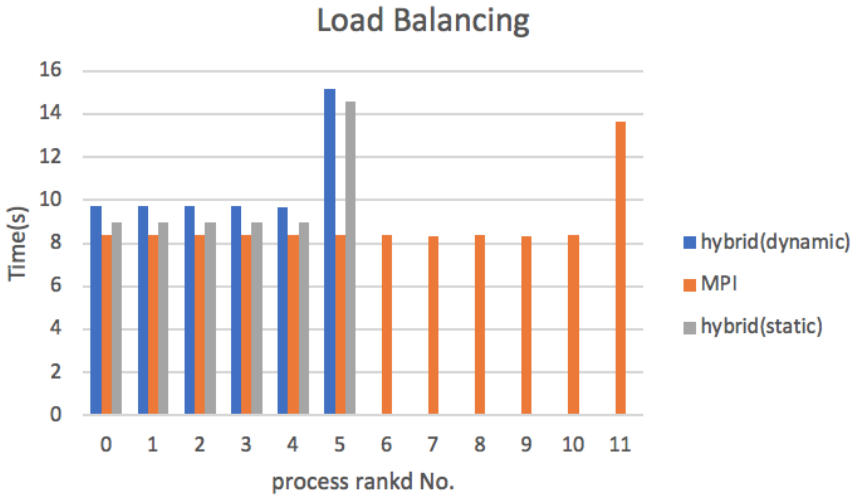


C. Single process Multi-thread



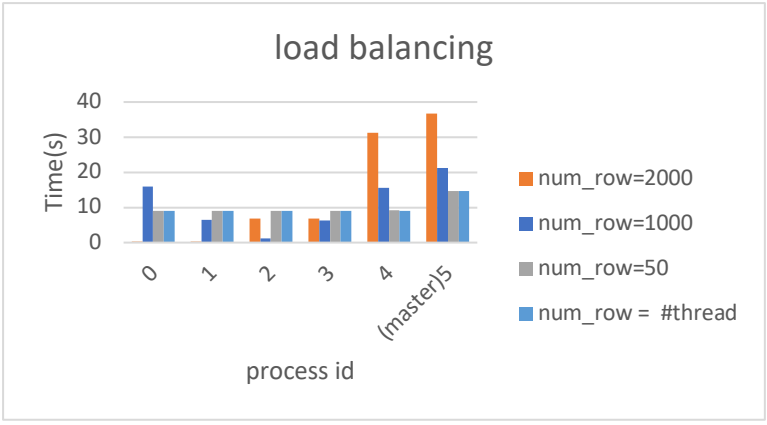
(b.)load balancing

1. Same #process*#thread different approach



2. dispatch number of row per process

Process 數:6 thread:2



III. Discussion

(a.) Scalability:

從 A,B,C 圖 我們可以看到在 hybrid 的情況下，scheduler 採用 static 方式較佳，但時間上的表現差距並不會太大，B 圖實驗我採用 process 數固定為 2（1 個 master,1 個 slave），而增

加 thread 數以及與 MPI 用多個 process(多個 slave 1 條 thread)做比較，我們可以看到 MPI 執行時間竟然比較低，代表 slave 的數量也會影響到執行的時間。從 C 圖表現可以看到由於每個 thread 只負責一 row，執行的時間比 hybrid 兩者都差，因此可以改進 pthread，一次一條 threa 負責多個 row，來降低執行時間。

(b.) 從圖 1.我們可以看到每個 slave 執行的時間幾乎差不多，由於寫入檔案最終交由 master 作處理，故執行時間會比其他 slave 多。圖 2.顯示，當我們給個 process(slave)分配的 row 數越小時，load balancing 狀況會越佳。

Experience and Conclusion

從實驗結果我們可以看到，工作平均分配的重要性，當一個 process 負責越多工作 ，執行時間會越長，也會造成其他完成工作 process 等待。不同的方式。

從這次作業，我學到如何透過 threading 的方式，增加運算單元，加速運算。

也學到該如何利用 open-mp 搭配 MPI 來執行多個 process 同時運算。MPI 的 Master-slave 機制可以讓 data 統一管理 ，由 master 分配 data,達到 load balancing。

作業中遇到最困難的地方是該如何加入 vectorization，一次運算兩個 pixel 來加速運算。

以及 master process 要如何分配工作給 slave process ，讓每個 process 都有工作可以做。