Parallel Programming hw2 ----Mandelbrot Set

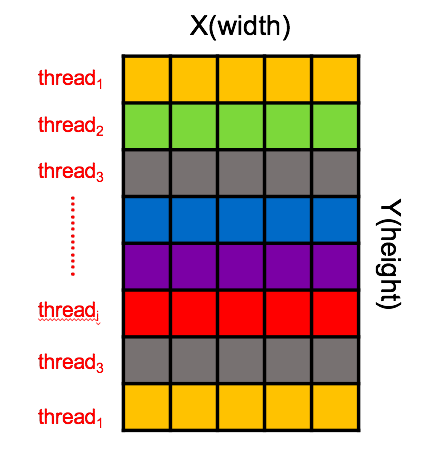
學號：109062639 姓名：葉哲欣

**Implementation:**

Pthread:

height: y軸點數 width:X軸點數 threadi: 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threadi 負責一列( X,yi),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 ranki:process id master:分配task給process

slave: 執行task num\_thread:每個process thread 數

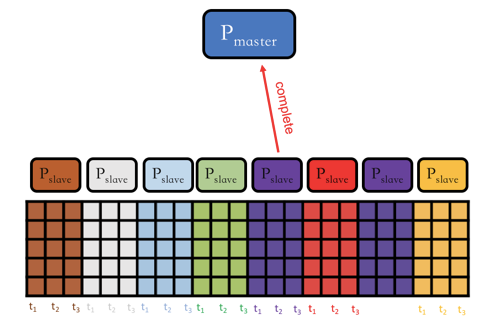
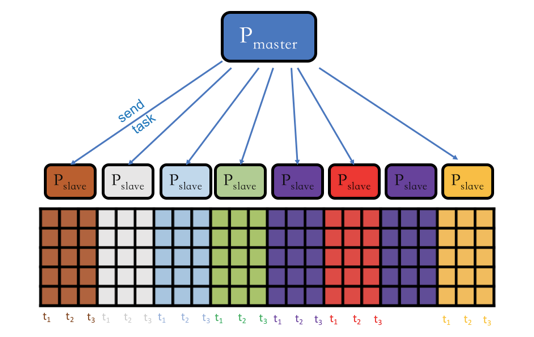
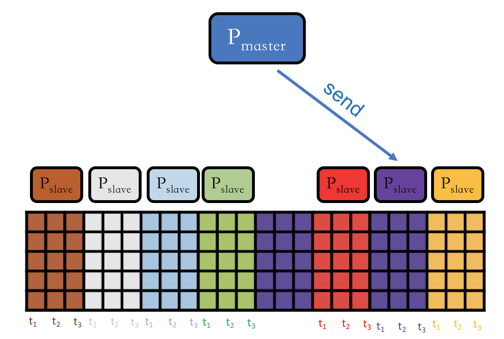
*(a.) How do you implement each of requested versions ?*

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:**

1. **Methodology**
2. System Spec:  
   使用apollo機器
3. **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;

1. **Plots: Scalability & Load Balancing**

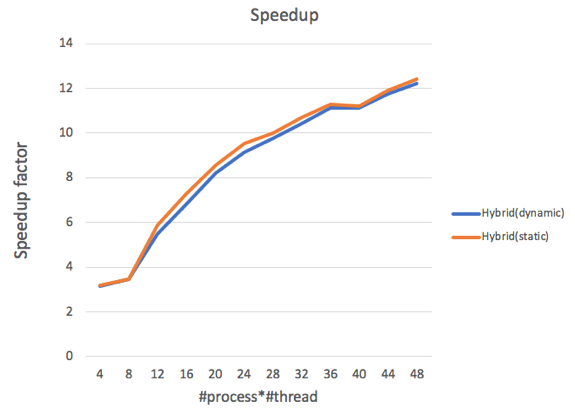
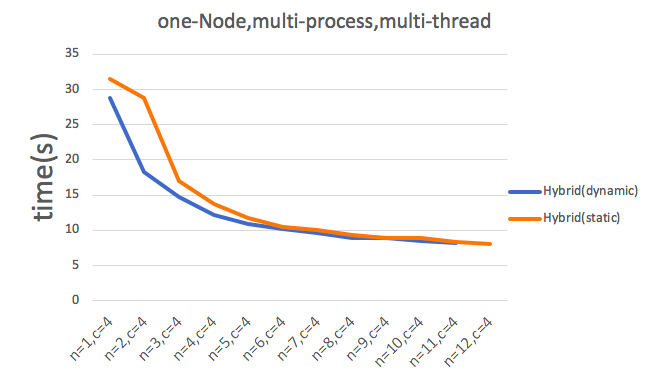
實驗採用strong scalability

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

Iteration = 1000 Height = 10000 Width = 10000

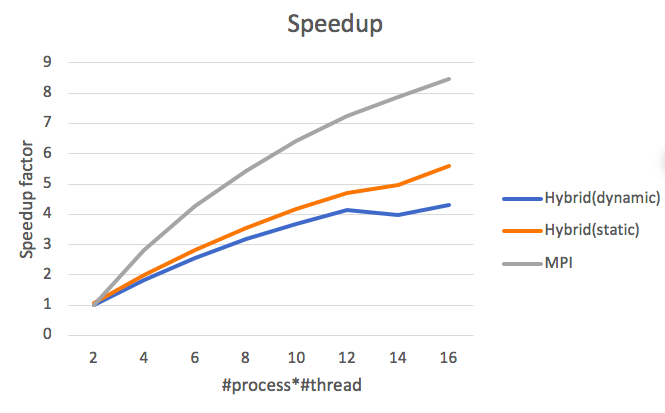
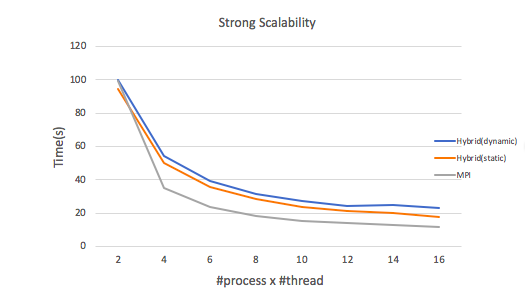
(a.)Scalability & Speedup

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

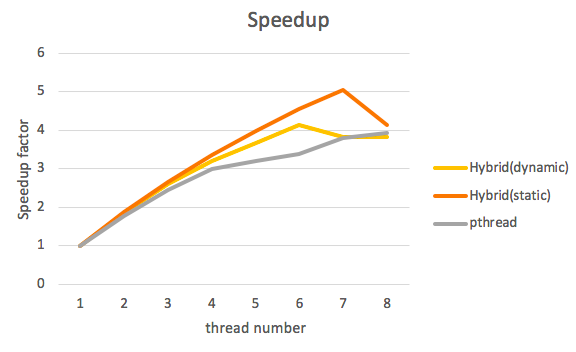
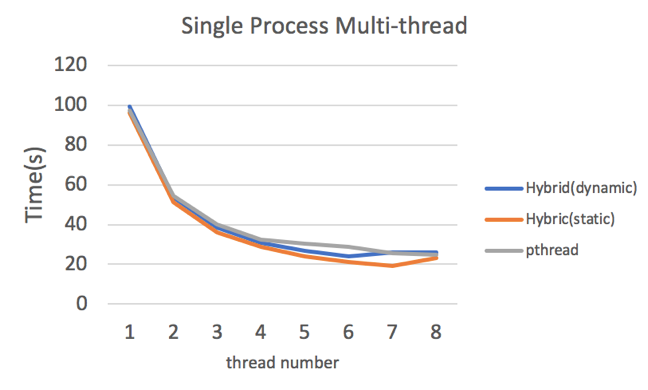


1. Same (#process＊ #thread )different approach.

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

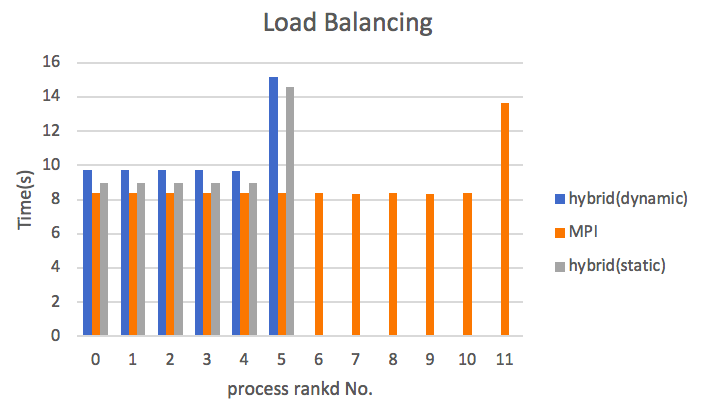


1. Single process Multi-thread



(b.)load balancing

1. Same #process\*#thread different approach



2. dispatch number of row per process

Process數:6 thread:2

1. Discussion
2. 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。

1. 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。