壹、Motivation

本次作業是藉由 Mandelbort Set 來探討 workload balance 對於平行化效能的影響，以及觀察在 Distribution memory、Shared memory、Hybrid memory 三種記憶體架構下的執行效能。

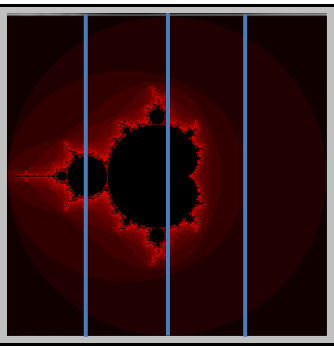
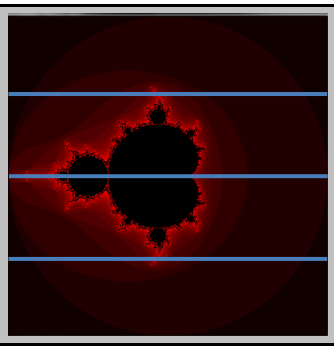
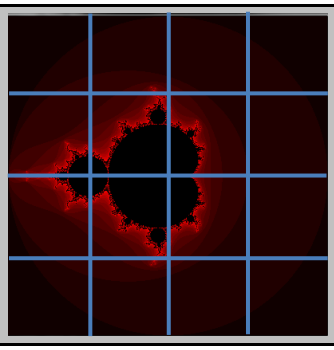
研究方法主要是對於上面的目標提出下列問題，以實驗的結果來回應這些問題，並於最後做出結論。

Q1. Mandelbor set 是否為 CPU bound？(是否適合做平行化？)
Q2. 何種切割方式能達到較好的效能？
Q3. Hybrid 架構在何種 distribution 下是較好的？
Q4. Hybrid 架構相對來說是否為最好的解？
Q5. 根據 Mandelbor set 性質進行優化後，workload balance 的影響是否減少？
Q6. Workload balance 是否與效能呈正相關？

貳、Implementation

一、Job split

本次個別實作 3 種不同類型的切割方式，如下：

Column split	Row split	Area split
		

二、Optimization

我們可以知道若某個點屬於 Mandelbor set，那它所要做的運算數必然最多，所以可以根據以下定理來進行優化：

$$If\ |c| \leq {}^{1}/_{4}\ ,then\ c \in M$$

在單機版本進行測試，可以發現運算次數會有約 13%的優化。

$$(1 - {}^{824355115}/_{947165115}) * 100\% \cong 12.966\%$$

參、Compile information

Type	Compiler & flags
MPI	mpiCC –o **{FILENAME}**.exe **{FILENAME}**.cpp –lX11 –O2
HyBrid	mpiCC –o **{FILENAME}**.exe **{FILENAME}**.cpp –fopenmp –lX11 –O2
OpenMP	g++ -o **{FILENAME}**.exe **{FILENAME}**.cpp –fopenmp –lX11 –O2

肆、Experiment & Analyze

一、Platform environment

Quanta Cluster Hardware Spec	
CPU	2 Intel Xeon X5670 CPUs, each one has 6 cores = 12 physical cores, 24 ones in HT mode
MEMORY	96GB
NETWORK	Ethernet/Infiniband
DISK	100GB SSD*1, 2TB HDD *2

Single queue ： 16 nodes，each node has 1 core
Multi queue ： 3 nodes，each node has 16 cores
Core16 queue ： 1 node，each node has 16 cores

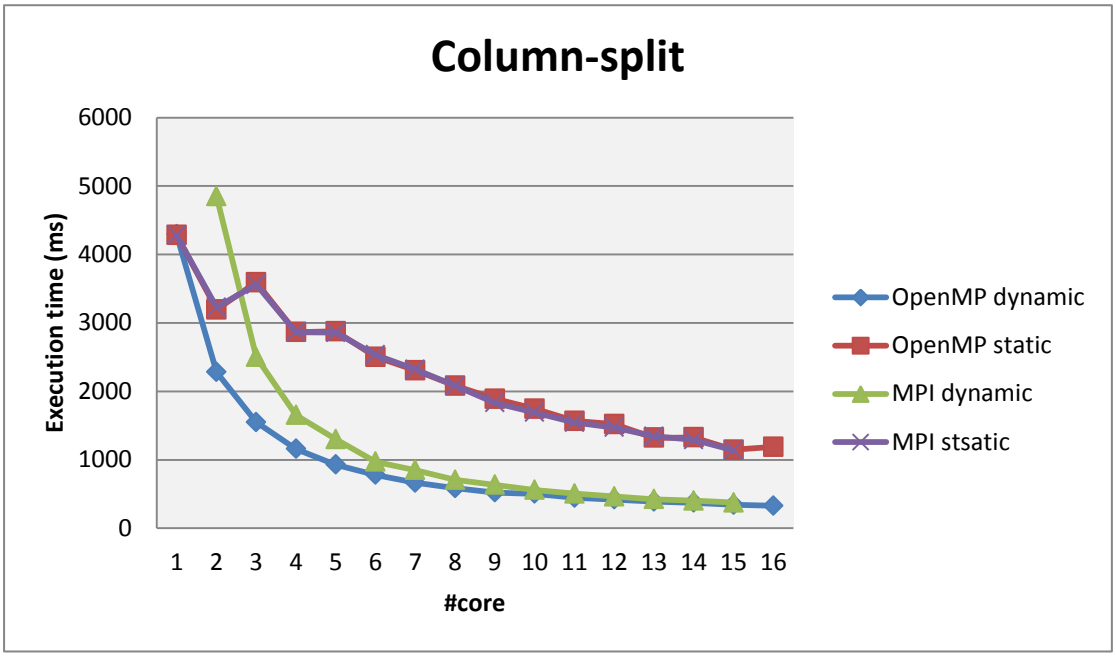
二、Time measure function

	Time Measure
1	`unsigned long long start_utime, end_utime;`
2	`struct timeval tv1, tv2;`
3	`gettimeofday(&tv1, NULL);`
4	`start_utime = tv1.tv_sec * 1000000 + tv1.tv_usec;`
5	
6	`// do something`
7	
8	`gettimeofday(&tv2, NULL);`
9	`end_utime = tv2.tv_sec * 1000000 + tv2.tv_usec;`
10	`printf("thread %d's runtime = %llu\n", tid, end_utime - start_utime);`

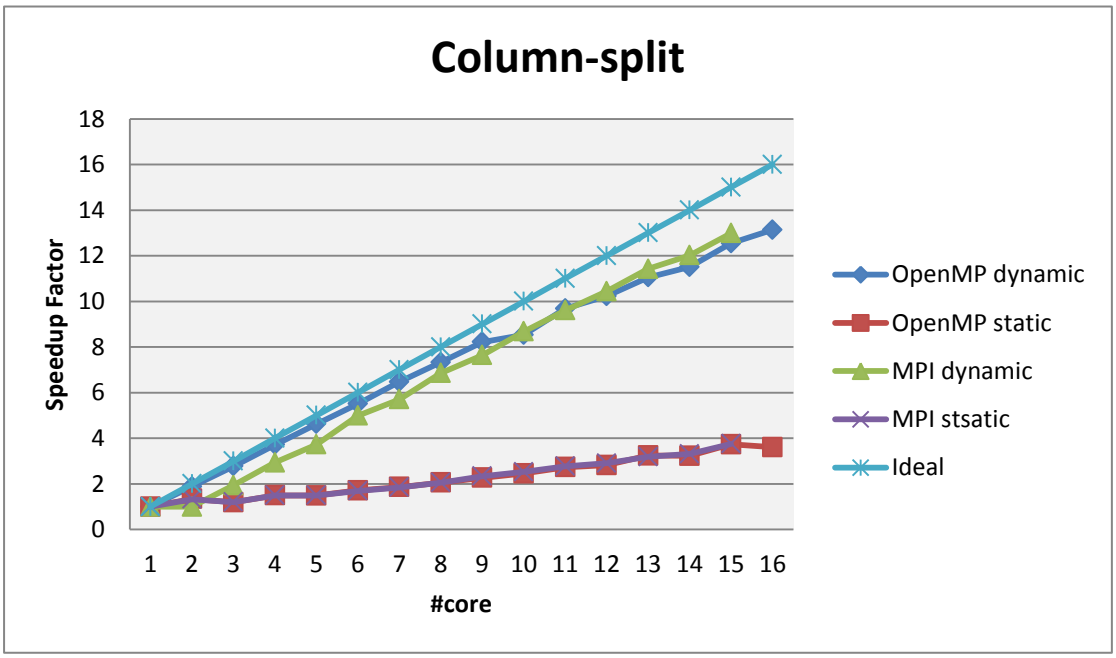
三、Experiment

(1) Strong Scaliabity

各版本執行平台　：OpenMP @core16、MPI @single。
實驗數據　　：跑五次，去頭去尾取平均。
測資範圍　　：Range = [-2,-2] ~ [2,2]，#Point = 1000*1000



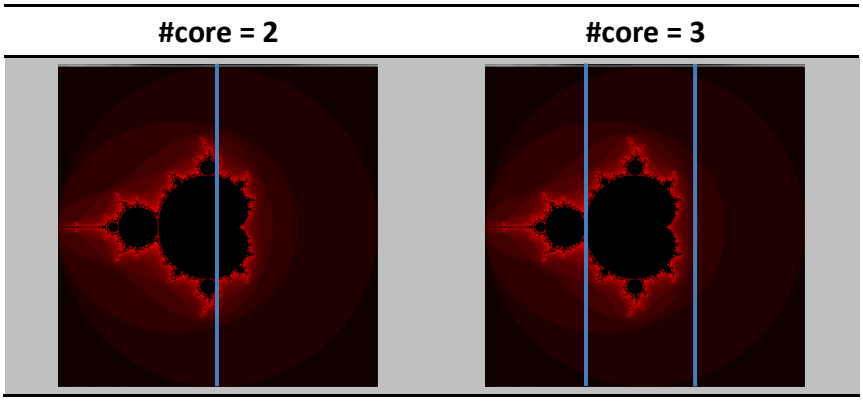
Figure(1)



Figure(2)

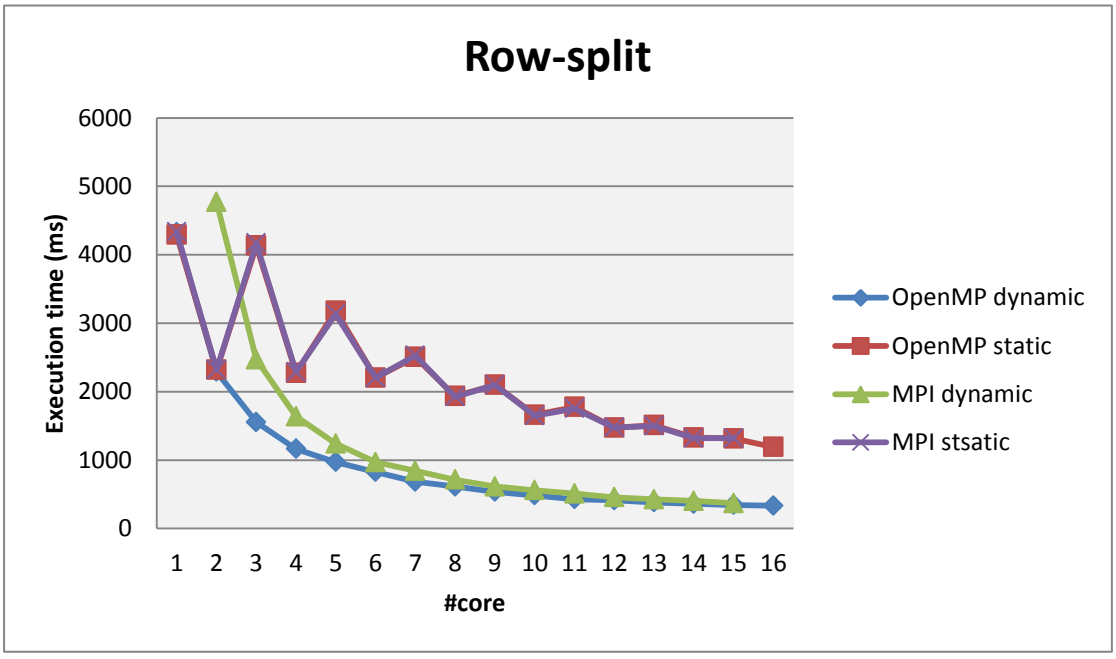
在 Figure(1) 中，看 static 的版本，大致走向可以看出具備 scaliabity 性質，其中有幾種 #core 跑出來的情況較特別，例如明顯可以看出 #core = 3 的效能比 #core = 2 還要糟糕。

觀察這兩種情況所切割出來的 Job 可以知道，在 #core = 3 時，會剛好把 workload 最重的部分集中在一個 core，而相對的 #core = 2 時的就會比較平均，故會有 #core 數較少但卻跑的快的情況。

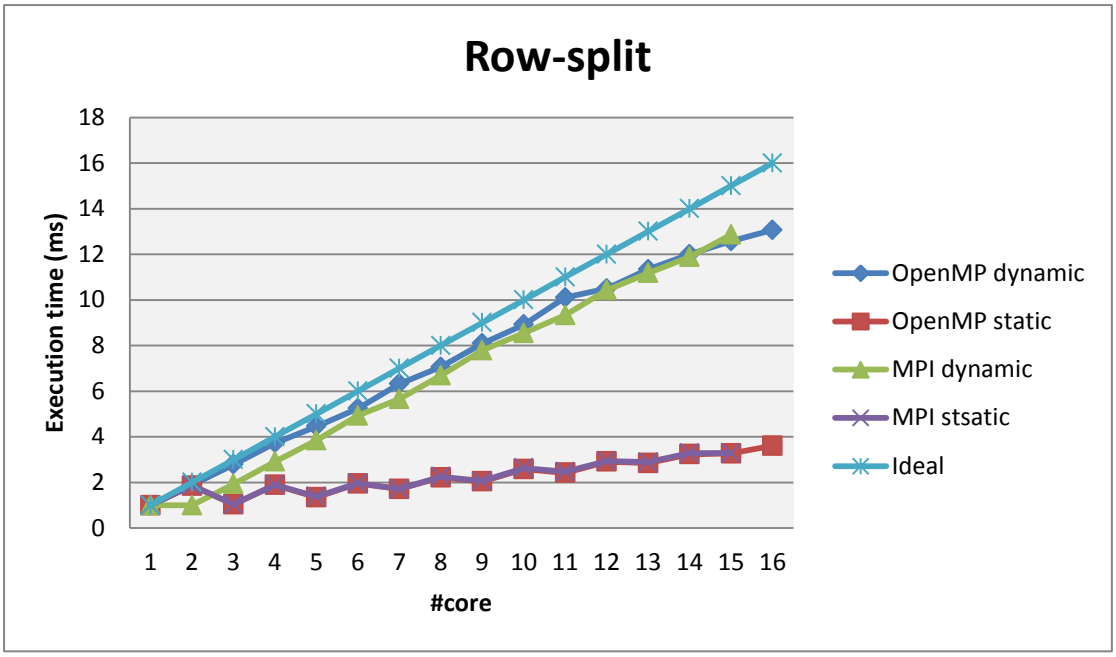


而 MPI dynamic 版本因為要有一個 process 當 master，故從 #core = 2 開始測試。

在 Figure(2) 中，可以看到加上 load balance 的機制後，效能提升顯著，已經幾乎逼近理想的 Speedup factor 線，而最後與理想曲線差異的原因，分別是因為 OpenMP 要幫忙做 Scheduling、MPI 需要 Network overhead 所造成的。

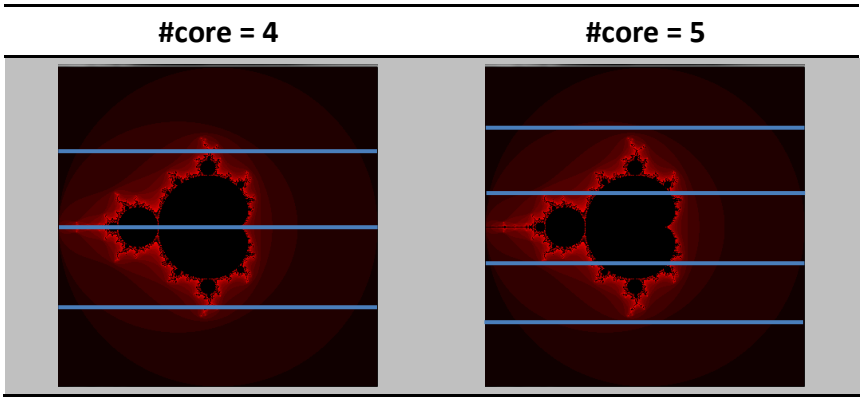


Figure(3)



Figure(4)

Row-split 跟 Column-split 的情況類似，但從 Figure(3) 可以看出再 Row-split 的情形下，Static 版本的震盪現象更明顯了。因為在這種切割下，圖形會對稱，所以在#core 是奇數的時候，中間的那 Process 總是負責整體 workload 最重的區塊，而影響到整體效能。



(2) Weak Scaliabity

各版本執行平台　：OpenMP @core16、MPI @single、Hybrid @multi。
實驗數據　　：跑五次，去頭去尾取平均。
測資範圍　　：Range = [-2,-2] ~ [2,2]，total #core = 9

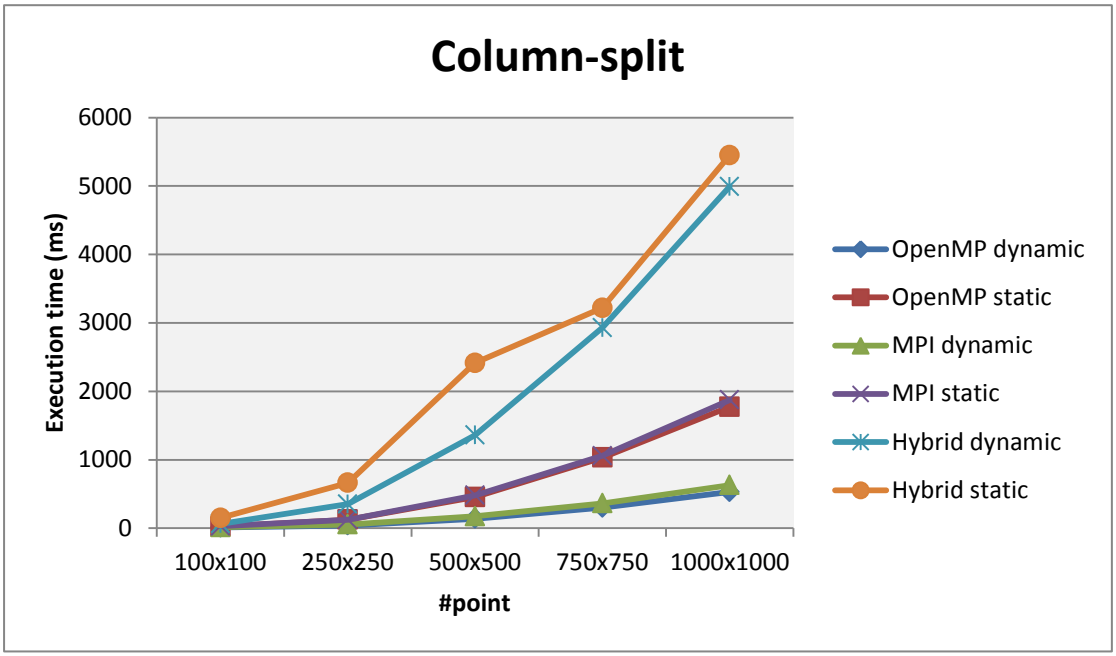
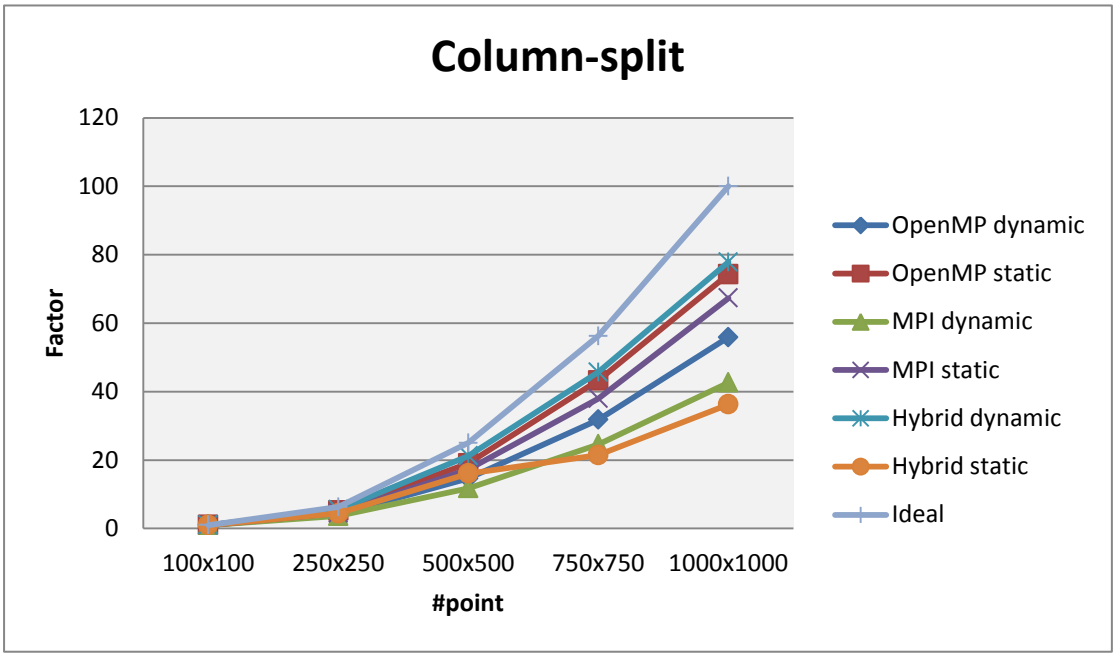


Figure (5)

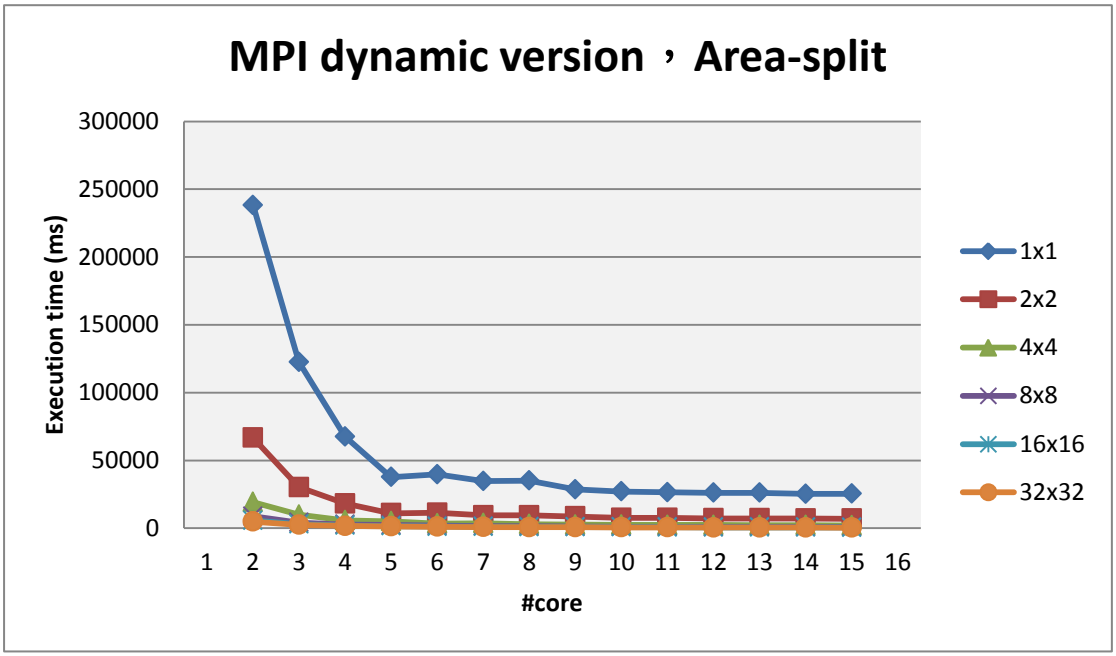


Figure(6)

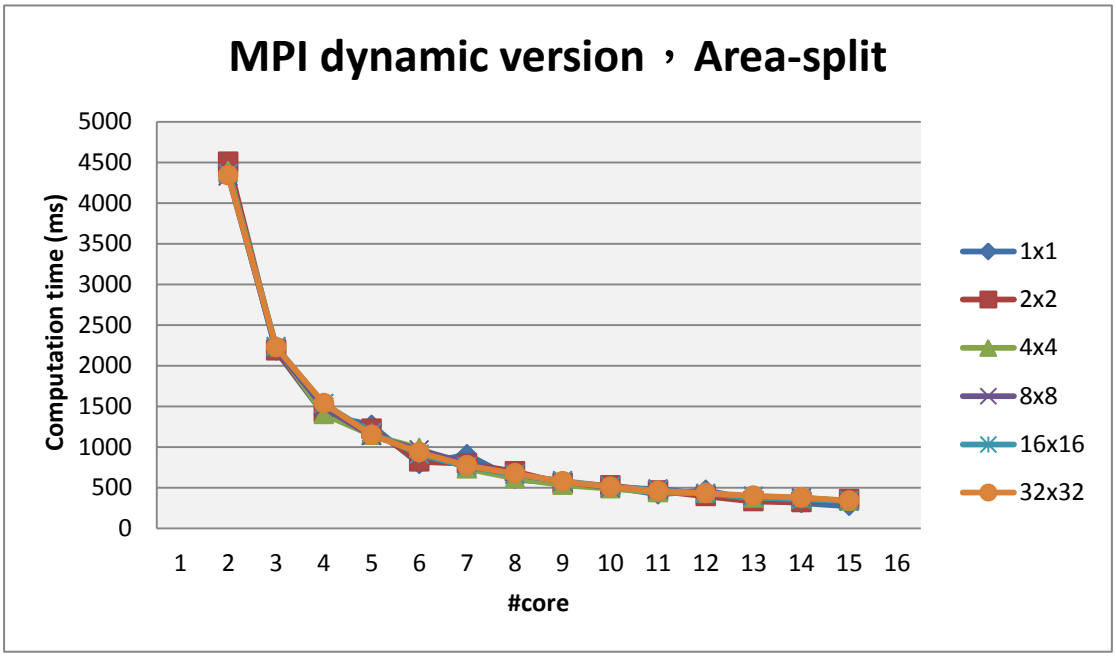
Hybrid 版本的效能理應要介於 MPI 跟 OpenMP 之間，但從 Figure(5) 中看到 Hybrid 的效能奇差無比。計算每筆數據之間的標準差，發現很多同樣測資之間的標準差高達 40%。而且 MPI 的效能在同樣的環境下跑，比前面測 Scaliabity 的實驗結果還要慢上二至三倍，所以應是實驗環境不穩所造成的結果，因此我們無法得到任何結果。

(3) Relation between workload balance and performance

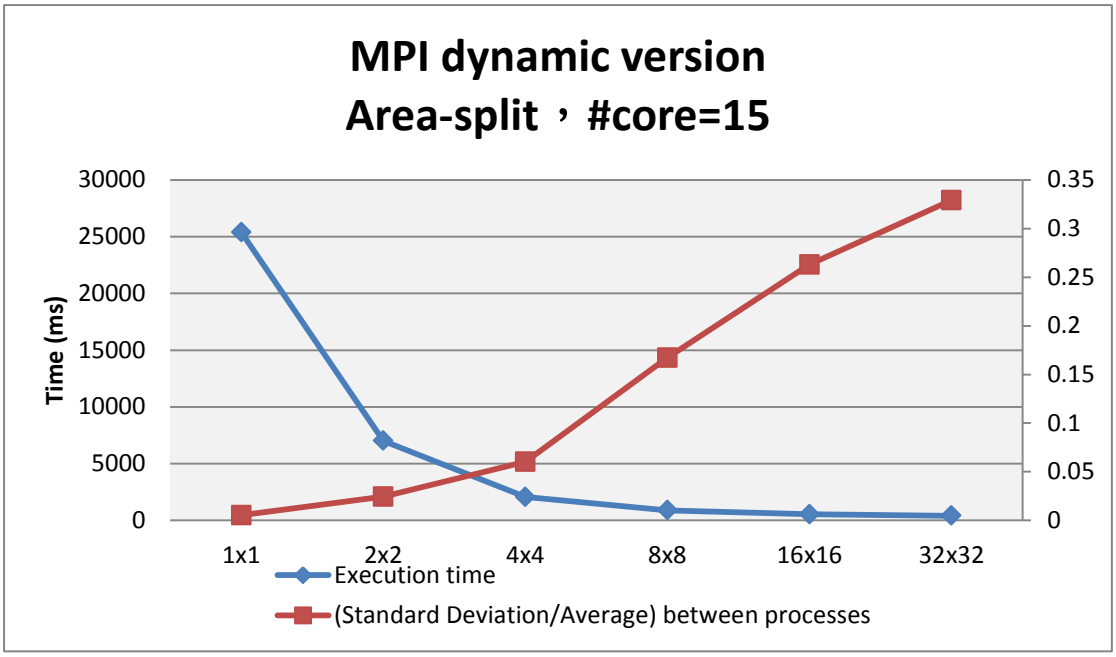
各版本執行平台　：MPI @single。
實驗數據　　：跑五次，去頭去尾取平均。
測資範圍　　：Range = [-2,-2] ~ [2,2]，#point =1000x1000



Figure(7)



Figure(8)



Figure(9)

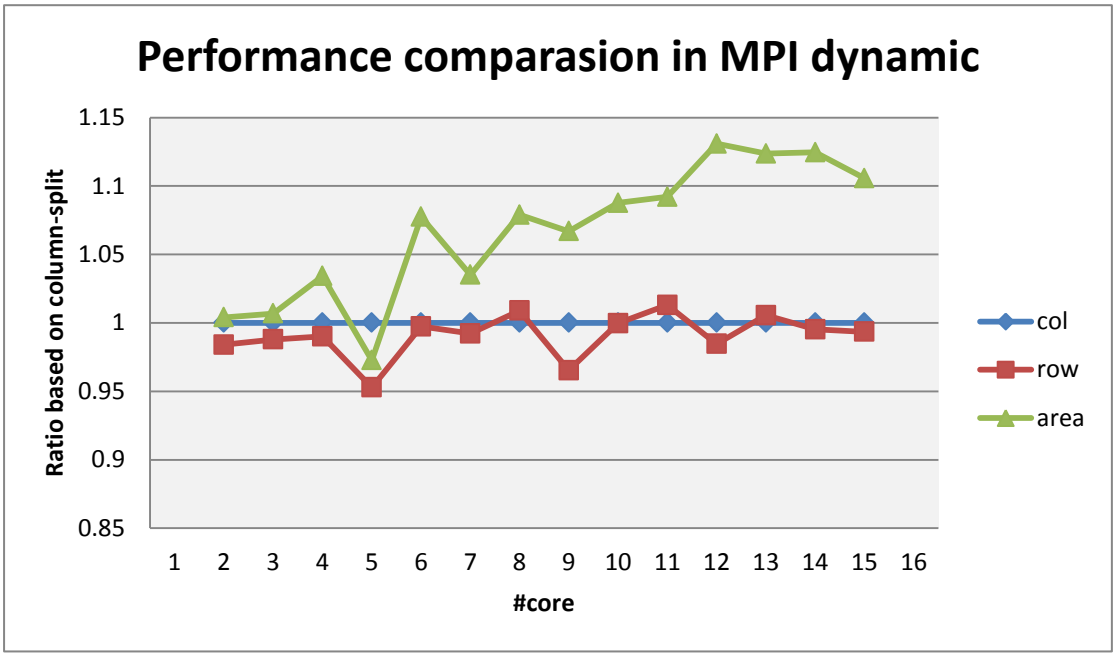
Workload 越 balance，performance 理應越好。但是在考慮到網路傳輸的影響下，越細緻的切割，所需要的溝通就要越多。所以用 MPI dynamic，area-split 的版本來觀察他們之間的影響。

先看 Figure(8)，我們可以看到單純的 Computation time 在不同細緻度的切割下幾乎沒有差別，所以在 Figure(7) 中的差異就取決於網路的影響。而明顯地，我們可以在 Figure(7) 看出網路傳輸的成本非常的高！

接著我們從 Figure(9) 可以看出，雖然差異極小，但是在每個 process 間的 workload 的確是越均勻。

(4) Which kind of partition is better?

各版本執行平台　：MPI @single。
實驗數據　　：跑五次，去頭去尾取平均。
測資範圍　　：Range = [-2,-2] ~ [2,2]，#point =1000x1000

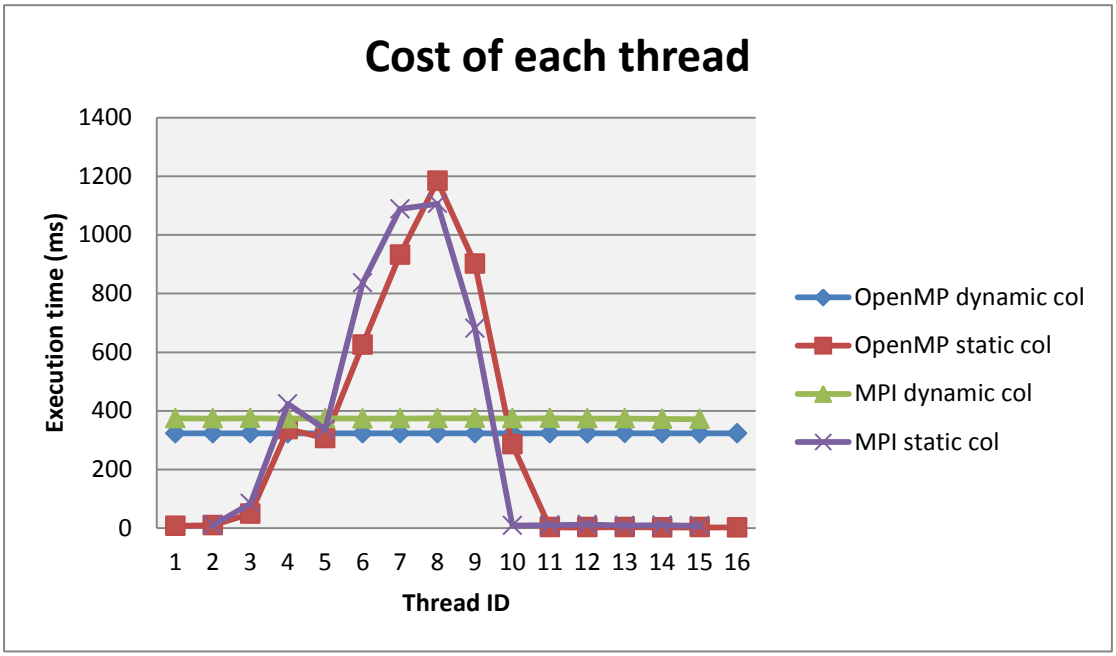


Figure(10)

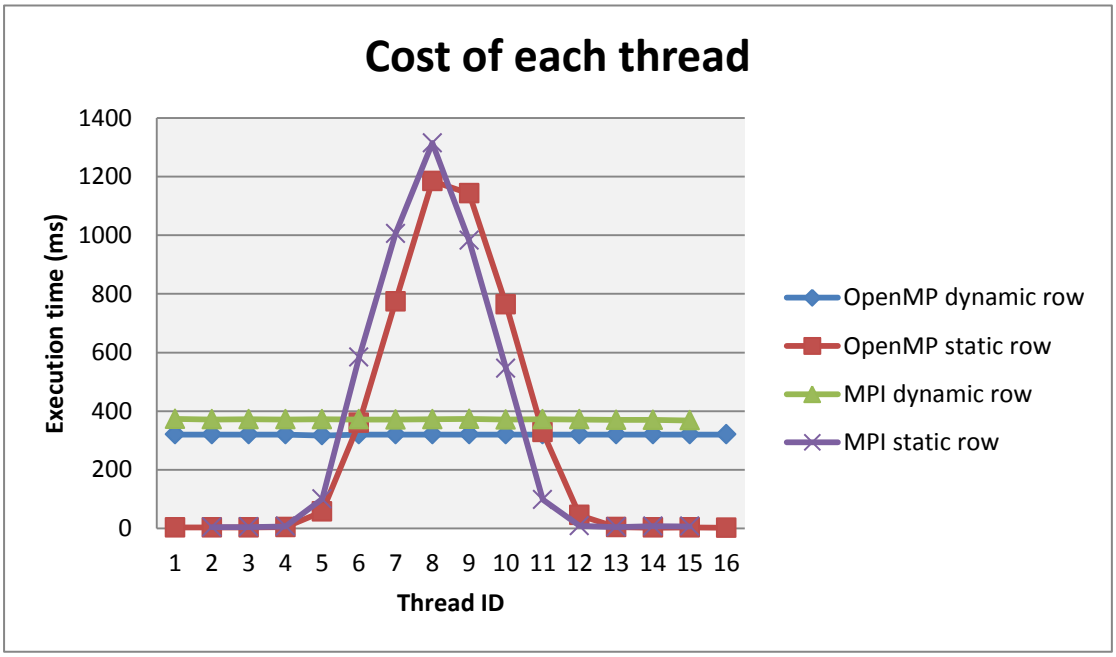
從 Figure(10)可以看出，整體來說 row-split 的效能較好！而圖表中的震盪現象原因如前面所述。

(4) Workload between threads

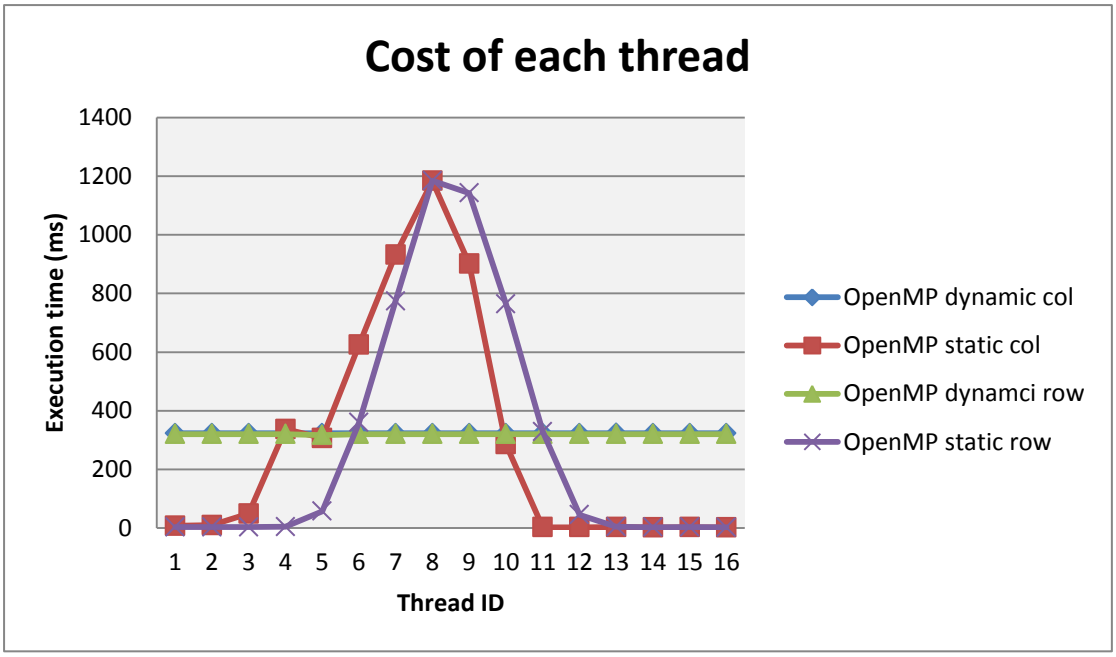
各版本執行平台　：OpenMP@core16，MPI @single。
實驗數據　　：跑五次，去頭去尾取平均。
測資範圍　　：Range = [-2,-2] ~ [2,2]，#point =1000x1000



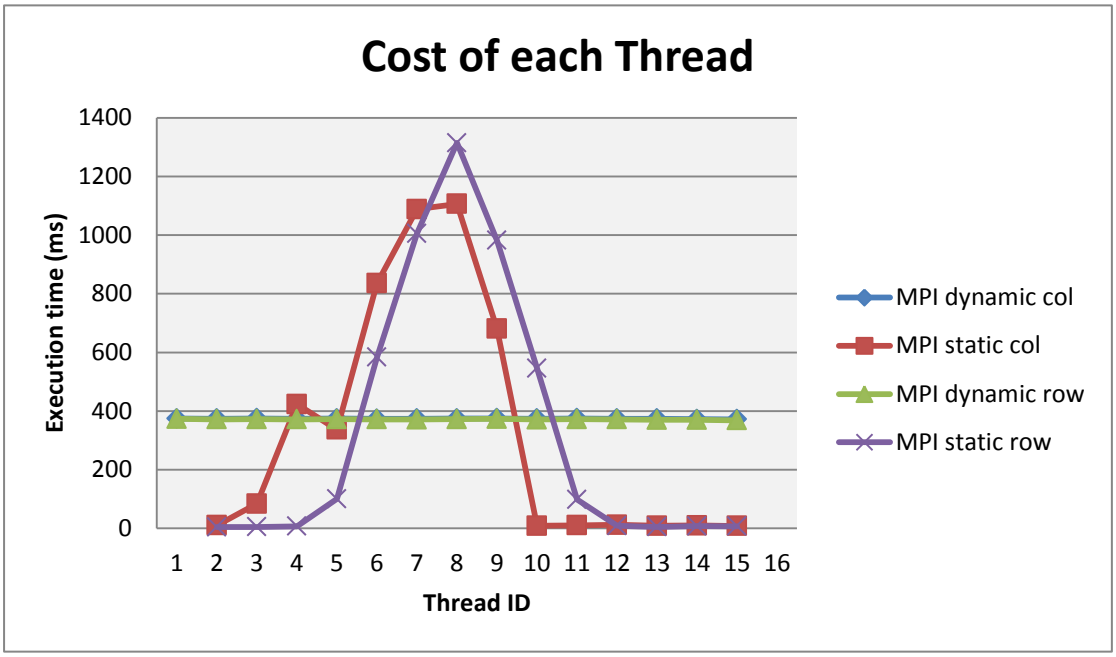
Figure(11)



Figure(12)



Figure(13)



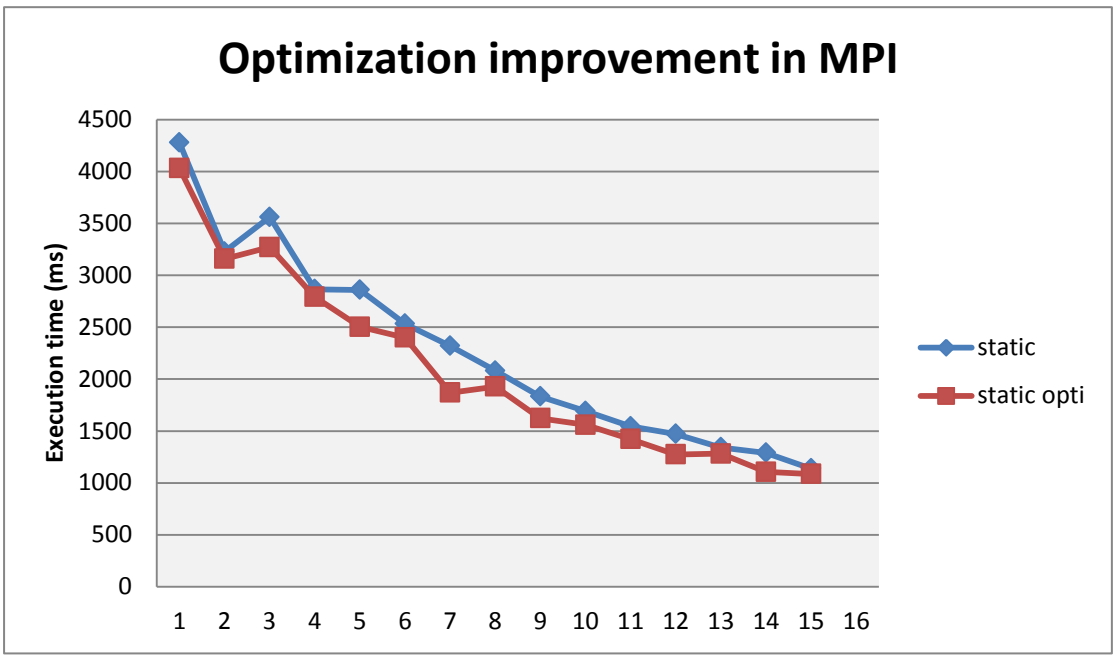
Figure(14)

(5) Optimization

各版本執行平台 ： MPI @single。

實驗數據 ：跑五次，去頭去尾取平均。

測資範圍 ：Range = [-2,-2] ~ [2,2]，#point =1000x1000



Figure(15)

伍、Conclusion

整體來說 Mandelbort set 這個問題在考慮 work loadbalance 的情形下很適合做平行化。我們也從實驗的結果看出，儘管用面積切割可以達到較好的 workload balance 但是所造成的 Network overhead 更大，所以並不值得我們採用，相比之下使用列切割會是較好的結果。

可惜的是，由於實驗環境不穩，所以無法確切看出 Hybrid 版本的效能。或許在穩定的結果可能依舊比較差，但是 Hybrid 架構的優點在於我們可以輕易地擴充機器來提升效能，而非像 OpenMP 所需要單台電腦提高 CPU 數那們困難，所以依舊值得我們去採用！

陸、Experience & Feedback

本次作業最困難的是 Hybrid 如何進行實驗。

柒、Reference

http://zh.wikipedia.org/wiki/%E6%9B%BC%E5%BE%B7%E5%8D%9A%E9%9B%86%E5%90%88