標題：HW2-report
姓名：葉怡君
學號：110062529

Implementation

Hw2a

先自定一個 structure，將讀取到的 filename、iterations、left…等變數存進去，以方便傳入 pthread 變數，接著根據 cpu 數目 create threads。

- 每個 thread 所要執行的 code：

(1,1)	(2,1)	(3,1)	(4,1)	Thread 1
(1,2)	(2,2)	(3,2)	(4,2)	
(1,3)	(2,3)	(3,3)	(4,3)	
(1,4)	(2,4)	(3,4)	(4,4)	Thread 2
(1,5)	(2,5)	(3,5)	(4,5)	

圖 1.(height = 5 ， width = 4)

採用 thread idle 就去要工作，直到所有工作都做完，以 y 軸需產生多少個點去切分(如圖 1.)，一次可計算 1~3 * width 個點。在要工作的時後，會用 mutex lock 住 j_cur 以防 race condition。

- 計算 Mandelbrot Set：

使用了 vectorization 的觀念，在 while loop 中，最多可一次計算 3 個 pixel，減少 loop 的次數及其所花的時間。

Hw2b

(1,1)	(2,1)	(3,1)	(4,1)	Process 1	Thread 1
(1,2)	(2,2)	(3,2)	(4,2)		Thread 2
(1,3)	(2,3)	(3,3)	(4,3)		Thread 3
(1,4)	(2,4)	(3,4)	(4,4)	Process 2	Thread 1
(1,5)	(2,5)	(3,5)	(4,5)		Thread 2
(1,6)	(2,6)	(3,6)	(4,6)	Process 3	Thread 1
(1,7)	(2,7)	(3,7)	(4,7)		Thread 2

圖 2.(height = 7 ， width = 4)

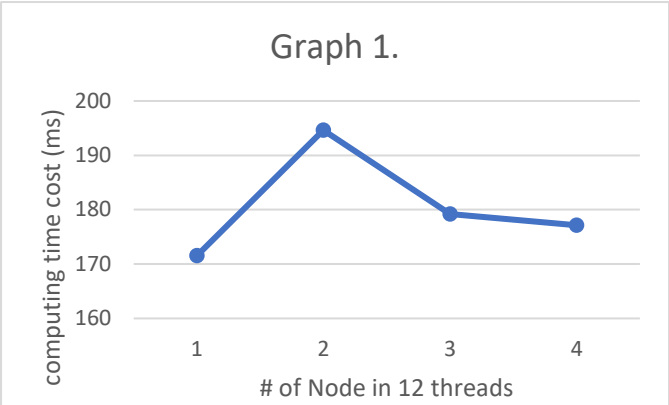
先均分 y 軸需要產生的點至每個 process，再與 hw2a 使用一樣的方法取得工作和計算 Mandelbrot set(如圖 2.)，最後透過 MPI 之間的溝通，將全部的結果回傳至 rank 0 的 process，統一由 rank 0 的 process 寫入 image。

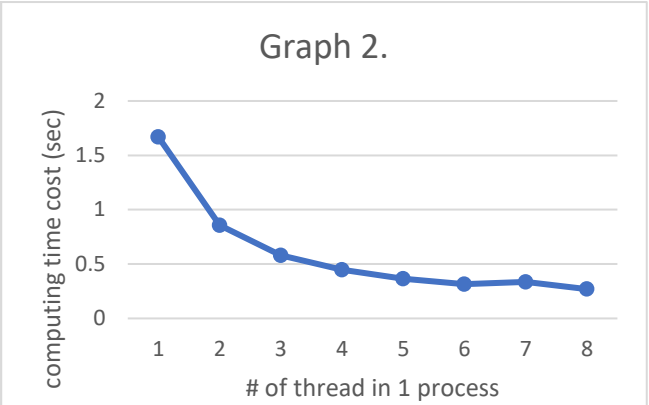
Experiment & Analysis

實驗設備：課堂提供的 apollo
測資：Fast01 → 2602 -3 0.2 -3 0.2 979 2355
時間計算方式： MPI_Wtime()
Computing time：定義為每個 thread 執行的 code 所花費的時間

Strong scalability

Graph 1.& 2. → Code：hw2a

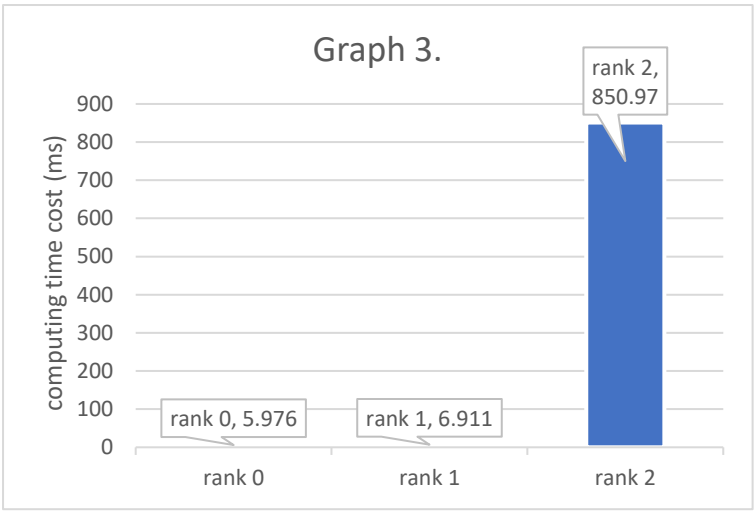




Discussion

在 Graph 1 中，我猜測是在 critical region 取得 global 變數 j_cur 所花的時間比較多，導致 scalability 不好。在 Graph 2 中，computing time 有隨著 thread 數的增加而變快，因為在單一個 node 跑，所以在 critical region 取得 global 變數 j_cur 所花的時間相對 Graph 1 會比較少。

Load balancing



Graph 3.

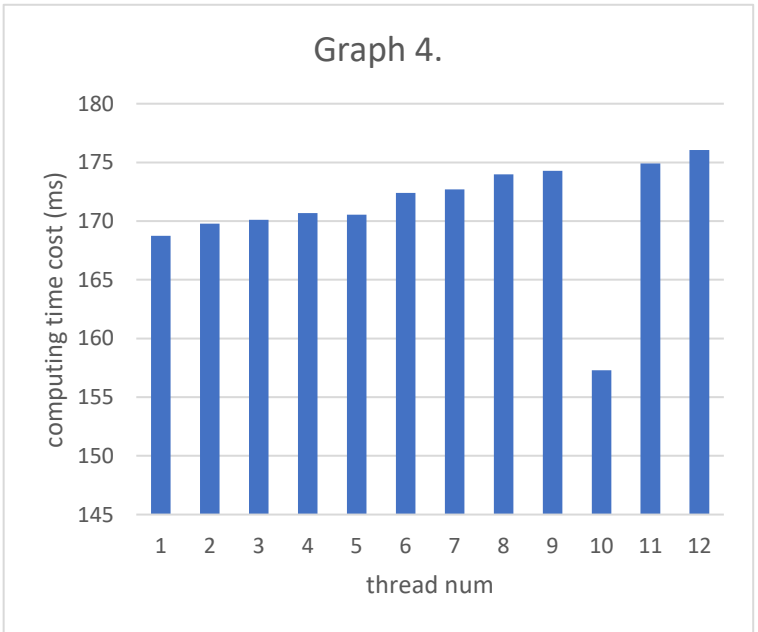
➔ 3 processes

➔ 2 threads in every process

➔ Code：hw2b

Discussion：

由於每個 process 所分配到要計算的點，沒有依計算量妥善分配，導致每個 process 所花的時間不一樣，有待改進。



Graph 4.

➔ 1 process

➔ 12 threads in every process

➔ Code：hw2a

Discussion：

因為 hw2b 平行的部分跟 hw2a 一樣，因此用 hw2a 來看每個 thread 的 load balancing 如何。從圖中看起來，因為採用不讓 worker 有 idle 的情況，所以基本上每個 thread 計算所花費的時間差不多，至於在 thread num = 10 的地方，我猜測是因為工作都被要完了而比其他 thread 早結束。

Conclusion

最初我是將固定列數個點平分給各 thread，但是發現各 thread 的執行時間會差太多，因此才改成誰 idle 誰就去要工作的寫法，但一開始一次只分配一對(i,j)給 thread，後來發現這樣太慢了，因此改成一次分配一個 row 給 thread，執行時間有變快，之後又改成一次分配 3 個 row 給 thread，並使用 vectorization 去 improve，這裡我學到了怎麼用 mutex 去 create critical region 來要工作，還有實作 vectorization，比較困難的點是如何抓一次要給 thread 做的工作量(batch)，還有如何得知每個點的計算量，並妥善分配給每個 process，這也是我沒有實做出來並且可以再去思考的地方。