CS542200 Parallel Programming Homework 2: Mandelbrot Set

104062703 曾若淳

Implementation

三個版本由於memory 架構不同，造成實做時主要差異在於「溝通方式」。MPI 的不同process 的memory 是獨立的，因此當平行的工作做完時，需要藉由額外的 message來傳送與接收計算結果。相反的，OpenMP 平行工作做完時，計算結果可以直接存放於 shared 的空間裡。而Hybrid 版本，適合將不相關的計算使用MPI 平行分佈在不同node 上，而node裡使用OpenMP加速。

Distributed Memory - MPI

由於Mandelbrot set的計算需要做圖形輸出，圍了簡單起見，我固定讓一個node (rank=0) 做圖形輸出，其他node 做計算並將計算結果傳給那個node。而static 和 dynamic 差別主要在於圖形計算的分配。

static

static版本以列為單位做width 的partition，每個計算node 分配到的是 width / (node數目-1) + (1 if width%node > 0 else 0)列的圖片點。而計算node 每算完一條，會將結果用non-blocking 的MPI_Isend 傳給圖形node，讓圖形node知道是那一列的方式為，將列index 當作message 的tag。而圖形node做輸出時，只需要按照順序non-blocking 的接收MPI_Irecv 那一列的計算結果，按照計算遞迴數對應成不同顏色印在螢幕上。

dynamic

dynamic版本也是以列為單位做width 的partition，但每個計算node分配到的列數不固定，而是根據整列需要的計算時間。
詳細實做方法為：一開始，圖形node 分配一列給每個計算node，計算node收到後會進入計算迴圈，每次算完會跟圖形node回傳結果，圖形node 收到結果，會給出兩種回應：

一、假如還有需要計算的列，圖形node會繼續分配一列給這個計算node，計算node收到後會繼續做計算。

二、假如全部列都已經計算完畢，圖形node 需要告訴這個node 可以結束了。我的實做方式為用message 裡的tag 來表示中止，計算node收到中止message後便會跳脫計算迴圈結束。

直到所有計算node 都結束，才能將圖形node 結束。這部份實做方式為使用一個變數count 紀錄分派出但為完成的工作數目，也就是說圖形node 必須等到 count=0才能結束。
由於Mandelbrot set在不同位置的點計算時間不一致，所以dynamic 版本在分配計算量時，比起static 版本的load balance 效果更好，能減少有些node計算完閒置，有些node太忙的情況。

Shared Memory - OpenMP

由於OpenMP 必須在同個node 裡做平行計算，整個程式分成sequential 執行區域會由 main thread執行，與parallel 執行區域由設定的thread 數量平行執行。所以OpenMP版本的 Mandelbrot set 的圖形輸出會是在sequantial 執行區域，每個點是在parallel 區域，同

時會有多個thread 平行的將計算結果存放在shared 的color 陣列。static 和dynamic 版本同樣差異只在分配計算量的方法。

Static

在OpenMP static版本，每個計算node 分配到的點的個數為 (width * height) / thread 數目，詳細實做方式為使用OpenMP parallel for 將兩層 - width 和 height 的 loop 做collapse後，使用static schedule 分配給每個 thread 做平行計算，並將計算結果存到color 陣列對應位置裡。全部計算完後，再按照順序做圖形輸出。

Dynamic

和static 版本相當類似，同樣直接使用OpenMP 的parallel for construct，將width 和height 的loop 做collapse後，但是schedule 使用dynamic 的分配方法，之後同樣將計算結果存到color 陣列應位置裡，於parallel 區域結束，按照順序做圖形輸出。
由於將dynamic 交給OpenMP compiler，無法確認load balancing 效果如何，但預期至少不會比static 差。

Hybrid Memory- MPI + OpenMP

Hybrid 兩者的方式為，外層為MPI，每個MPI node裡使用OpenMP加速。也就是說，有一個MPI node專門做圖形輸出，其餘MPI node 做點的計算。對於計算的分配，我以列為單位將圖片分配給 MPI 計算node，而每個MPI 計算node 裡會再使用OpenMP 平行加速計算該列。至於static 和dynamic 的Hybrid 版本差異，大致上跟 MPI 的部份相同，也就是外部分配列的方法不同。

Static

因為每個MPI 計算node 要計算的圖形列是deterministic 決定的，例如rank是k的計算node被分配到的列是所有 k + (width / 計算node數)倍數的列，因此圖形node 並不需要用message 來分配計算點。而MPI 計算node在算每一列的時候會用 OpenMP 的parallel for 加速計算，算完後計算node 會asynchronously 傳給圖形node。圖形node 會按照順序做輸出。

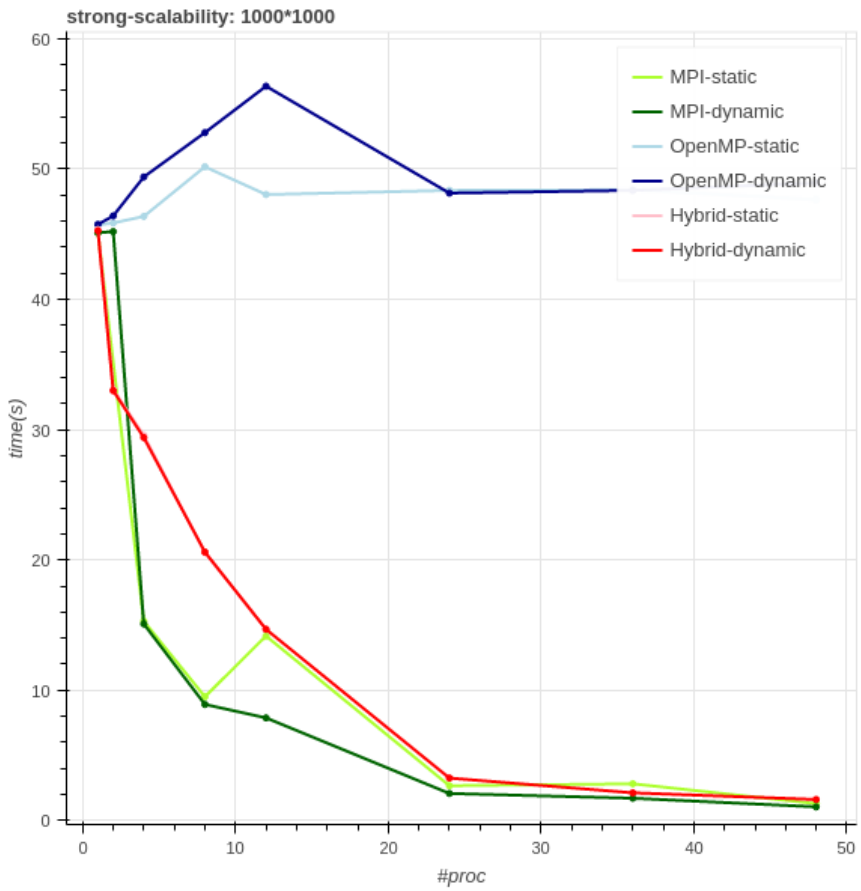
Dynamic

外層MPI，由圖形node 分配圖形列給 MPI 計算node，每個計算node 內部用 OpenMP 的parallel for 加速該列的計算，計算完後將結果傳給圖形node，圖形node收到計算結果會做輸出，並給這個計算node 一個message，假如還有為計算的部份便會分配一列，假如全部計算完了便會告知該node 可以結束了。和MPI 相同，直到所有計算node 都已結束，圖形node 才能結束。

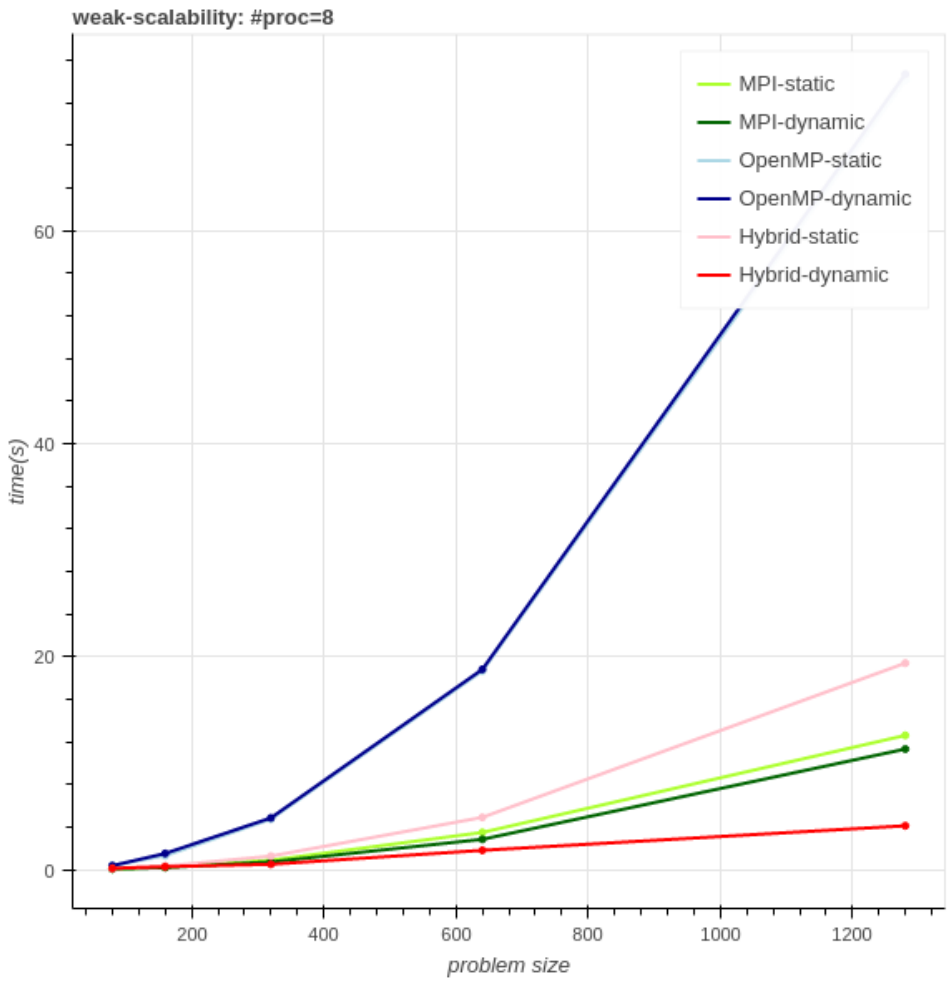
Performance Analysis

實驗環境為課程提供的 batch cluster：1 sequencer node + 10 worker node，每個 node 為2x6 Interl(R) Xeon(R) 的CPU 配備96GB 的memory 和 2TB HHD。

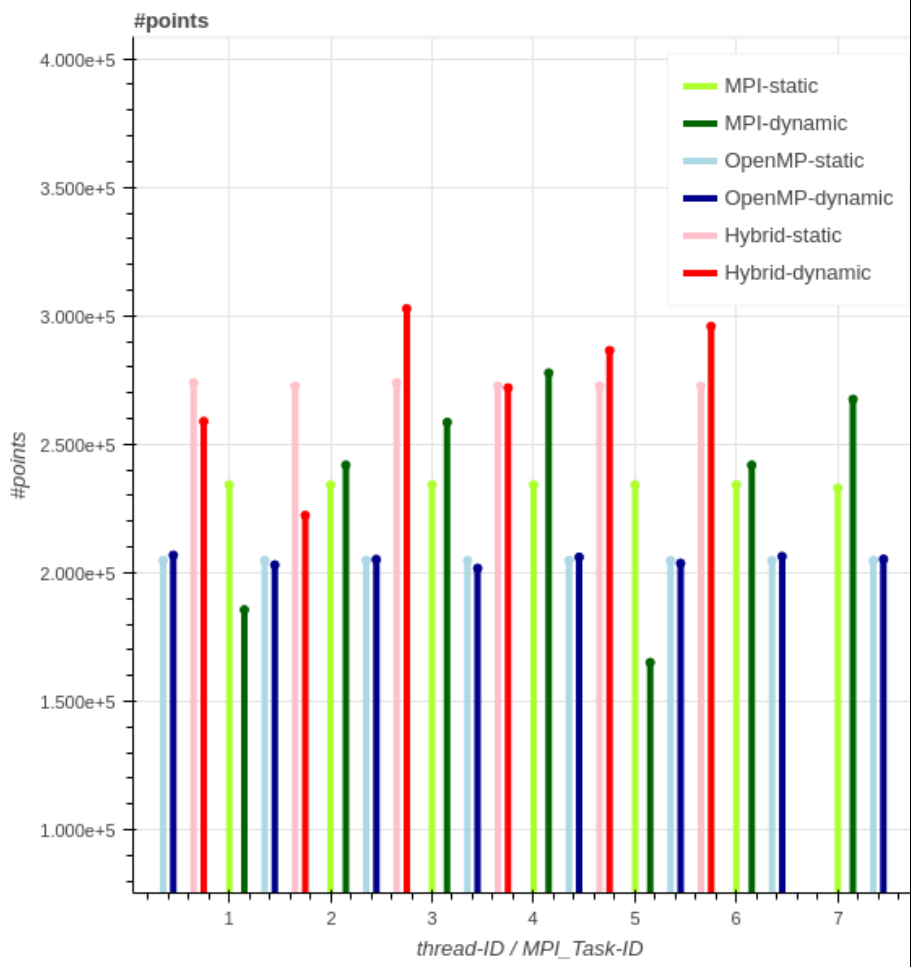
Strong Scalability

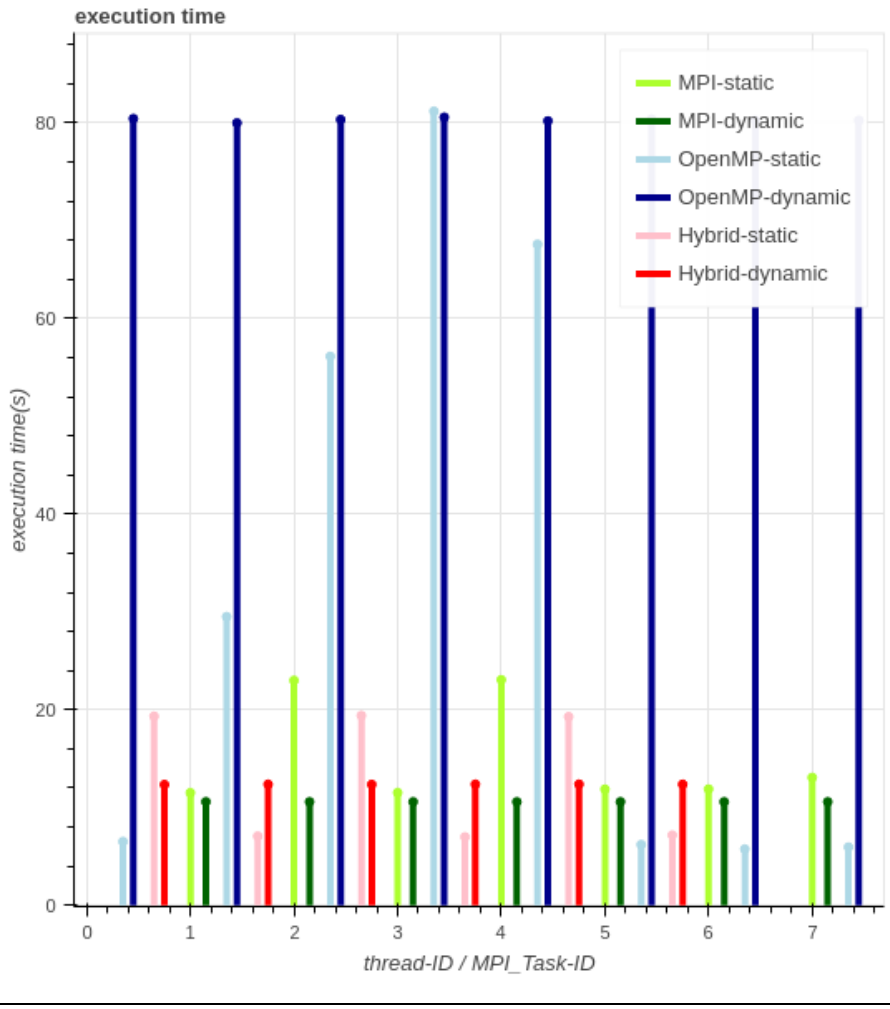
固定圖片大小為1000x1000，增加process unit時，OpenMP版本的總體時間很意外的不減反增，而MPI和Hybrid版本都能看見明顯加速，尤其又以MPI 版本scale效果最好，可以得到接近linear speedup。	

Weak Scalability

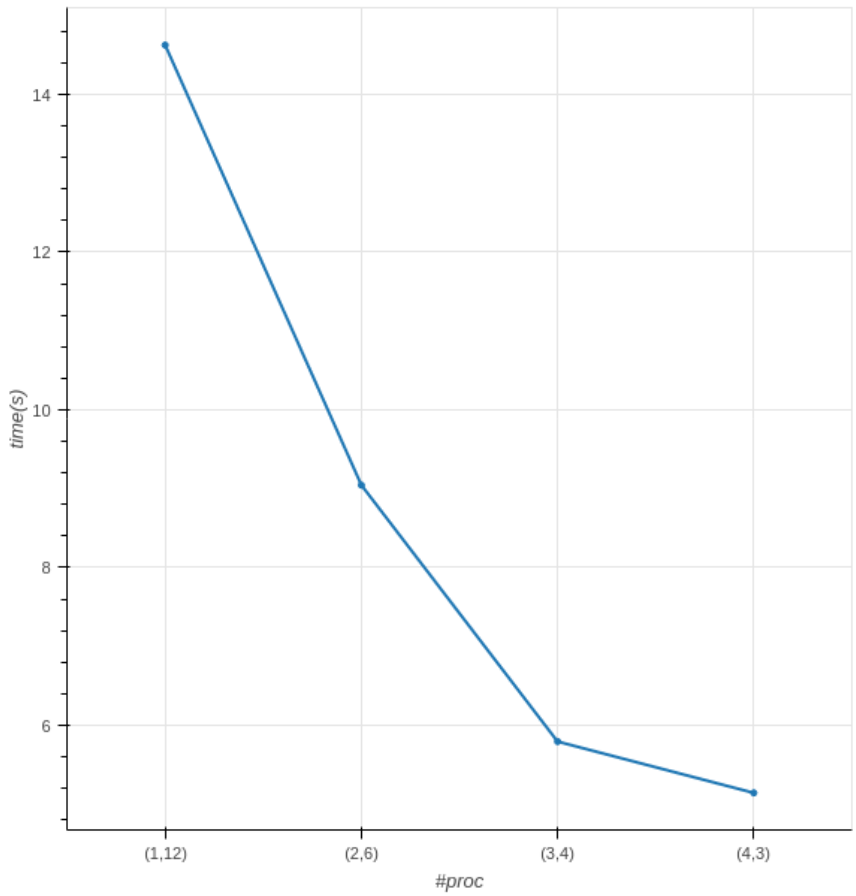
固定使用8個process unit，static和dynamic的schedule分法在MPI和Open版本差別不大，但是兩者一起使用的Hybrid版本，可以明顯看出dynamic 版本較好。	

Load Balance

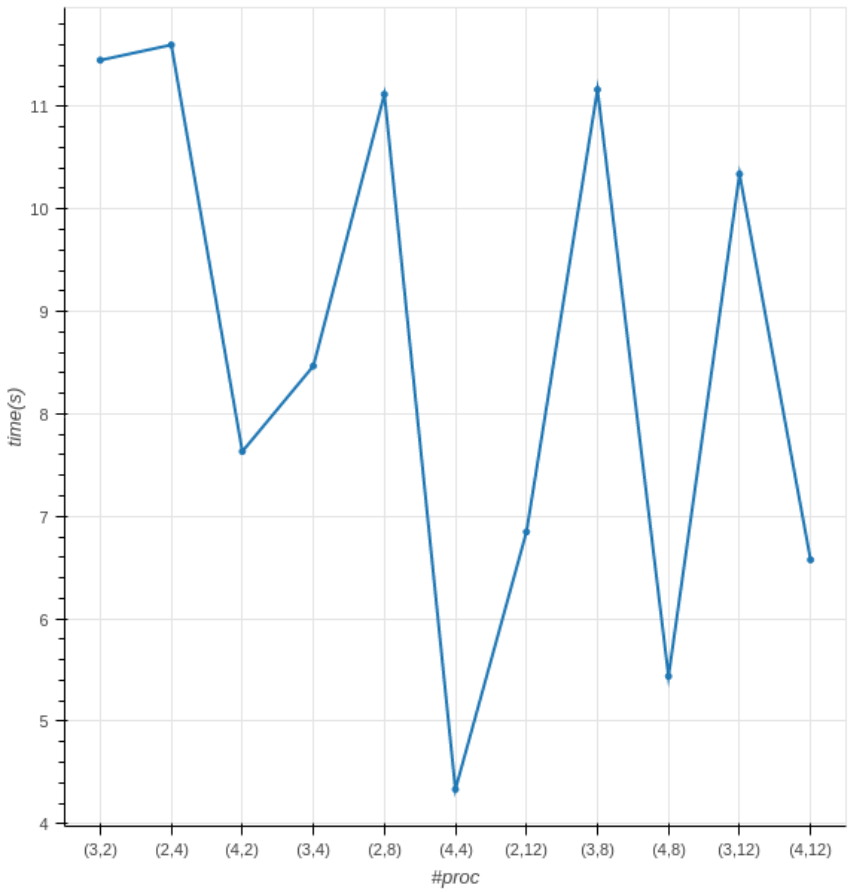
static schedul會平均分配每個process unit處理的point， 而dynamic schedule是動態分配的，其中又以MPI 差別最大因為process間溝通成本最高， 其次為Hybrid版本因為內部有用OpenMP加速，而溝通成本最低的OpenMP版本process unit處理point數差異最小。	

dynamic schedule 執行時間都差不多，static schedule差異很大，其中又以 OpenMP 版本差異最大，MPI其次，而Hybrid版本差異最小。表示OpenMP版本總體執行時間被最慢的thread dominate，獲得的加速最少，而 Hybrid 加速效果最好。	

Best Ratio when #proc = 12

在圖片大小固定為1000x1000時，固定使用12個process unit，對於Hybrid dynamic版本來說，最好的比例是4個process 每個process 內使用3個thread。	

Best Ratio

在圖片大小固定為1000x1000時，對於Hybrid dynamic版本來說，執行最快的不是使用最多process unit，而是16個，使用4個MPI node，每個node內部使用4個OpenMP thread。	

Conclusion

這次實驗最讓我意外的是 OpenMP，照理說OpenMP 在thread間的溝通成本最低，理論上會有最多的speedup，但是結果卻是MPI表現最好。而Hybrid 最後關於固定problem size下表現最好的process unit數目，不是最高的，在預期之中，因為process unit數目提高溝通成本會變高，而最好的是MPI task數對 thread數最balanced的組合。