Homework 2: Mandelbrot Set

110164508 黃柏惟

1. Implementation

Pthread version:

版本一: 直接將workload 平均分配到各個row

這個版本很直覺就是使用跟Lab2練習時候一樣的方法去partition data，把workload 均勻的切給各個thread去處理。

版本二: 透過Mutex 動態去分配workload

這個版本因為聽說Load Balance很重要因此在partition data的時候使用dynamic的方式去取，透過global的變數去紀錄當前算到哪一Row，並且透過Pthread內建的Mutex在要取這筆global變數的時候確保data不會被其他thread取用造成racing的情況發生。

版本三: 使用Mutex動態去分配workload並且使用vectorize加速重複運算。

這個版本是基於版本二後續的延伸，因為這次作業Mandelbrot Set的計算在每個pixel都是相同的並且是dependent的所以可以使用Lab2有提到過的Vectorize去進行優化。 Vectorize顧名思義就是將多個data可以透過一個指令的方式就去將她計算完成。 這次我使用SSE2的指令，使用__mm128d將data load到high以及Low去計算。我使用Union的方式去把宣告vector以及 local的data，這樣子宣告的好處就是我可以很方便在vector端算完後在local這邊去使用(相較於debug也較友善)。再來就是由於這次是一次計算兩個pixel的data，因此兩邊不一定會同時結束，因此我的作法是當今天有一方算完後，我會break出去準備下一筆的data，這樣子的好處是vector不會因為一端計算完成後就IDLE在那邊，而是可以一直處理之後的運算。

Hybrid version:

Hybrid version: 在hybrid版本中使用到了MPI + OMP，除了pthread版本使用到的vectorize可以直接的移到Hybrid version中。 最重要的就是如何讓data load balance，Mandelbrot Set 有個特性就是由於是圖片的關係因此相鄰幾個row col的值不會差到太多(工作量相當)，因此在切的時候就要均勻的分給各個process去處理，可以相比整塊整塊分出去得到更佳的scalability。因此每個Process會平均拿到差不多的工作量與工作區塊，之後透過OMP的lock去讓內部的cpu可以動態的去拿取工作就跟Pthread版本的做法一樣。 這邊我的MPI彼此之間不會溝通，因為我對於Communication的優化並不是做得太好因此這邊就直接把任務平均分下去最後透過Reduce把算完的data收回來。

2. Experiment & Analysis

i、 Methodology

a. System Spec

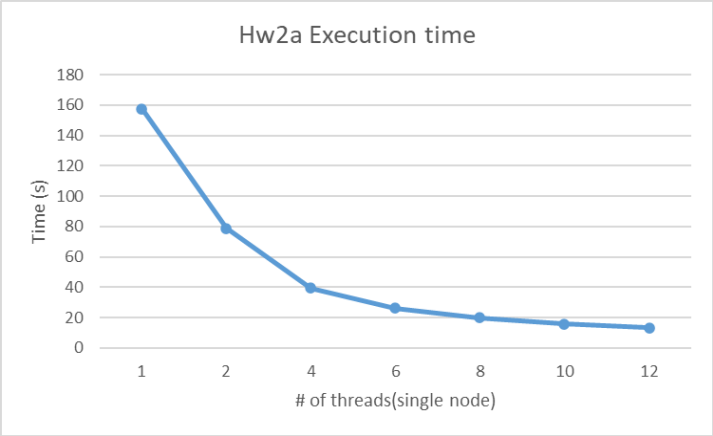
使用課程提供的appollo.cs.nthu.edu.tw server進行實驗

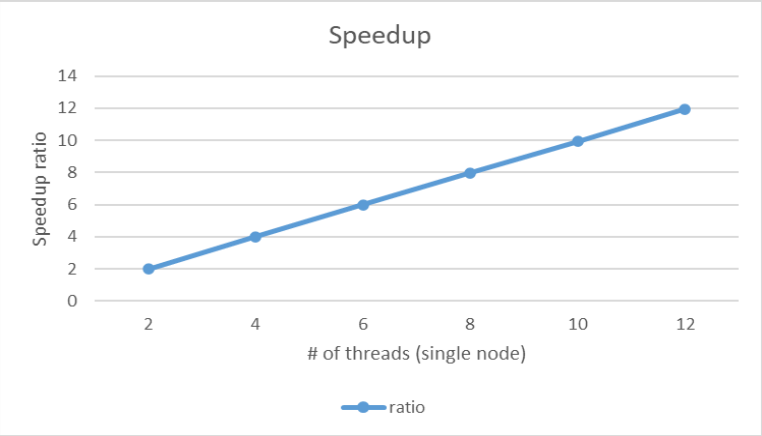
b. Performance Metrics

這次使用 Clock_getTime()的function去計算執行時間，我計算的區間是程式開始到寫入檔案前的時間計算。

ii. Plots: Scalability & Load Balancing

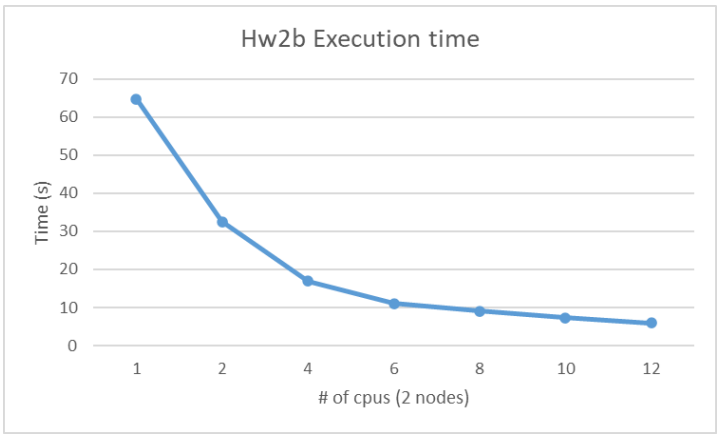
Hw2a:

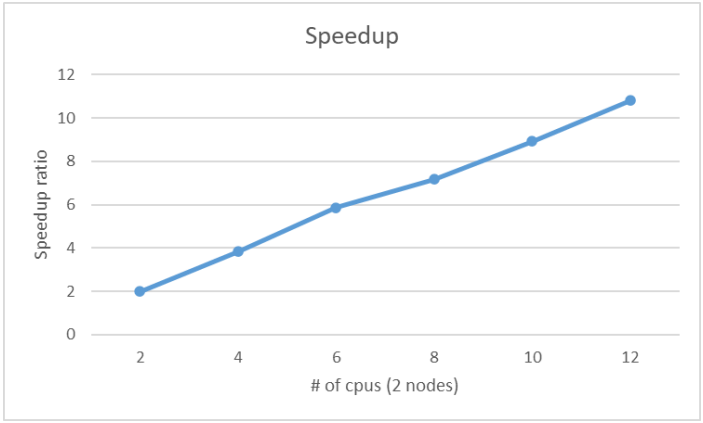




Hw2a的scalability相當的好，是因為在hw2a的實驗中我們有限制只能使用single process因此程式之間不需要溝通，做完了就可以馬上去取下一個要處理的data，因此我們多給多少的硬體資源，speed up ratio就可以提升幾倍。

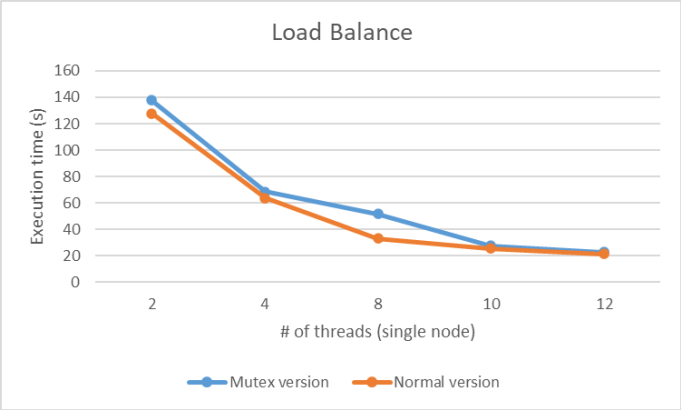
Hw2b:





Hw2b的實驗中scalability的效果就稍稍差了一些，不過還是很接近理想的ideal值，我想原因是因為有數個processes可以使用因此在reduce收回來的時候一點communication time，進而造成scalability稍差的原因。

Load balance分析:



由於這次load balance很重要，所以特地把Hw2a實作兩種load balance的方法拉出來比較，比較讓人意外的是均勻分配工作量的normal version反而表現得比Mutex去取任務的版本還要好。 個人推測是因為這次作業的特性造成相鄰row之間的工作量不會差到太多，因此使用這種簡單的分配就可以有很好的效果，反而mutex因為在取得row col的時候其他的cpu需要等待他lock後才可以繼續工作因此造成效能反而稍微差一點的結果。

3. Experience & Conclusion

這次我在一開始的寫法就意外的把load balance處理得很好，以至於我將程式改成使用mutex去取的時候反而並沒有得到太多的優化。不過這次對於vectorize學到很多，因為之前一直都在寫Verilog所以有時候會很不習慣在寫C/C++的時候對於計算並沒有辦法有並行的概念，直到這次學習到使用vertorize一次去執行多個data的運算。並且對於scalability的數據其實也在意料之內， 因為這次沒有使用到太多的communication的溝通，因此可以有很強的scalability。