**Homework 2: Mandelbrot Set**

108062313 黃允暘

**Implementation**

以下會根據pthread及hybrid版本來講解：

**Pthread**

1. **partition the task:**

為了要有效利用pthread完成平行話，首先要定義出執行工作的基本單位。我將一個row的計算當作是一個計算的基本單位。定義出基本單位之後，接下來就可以分配給threads去執行了。考慮到每個row的計算量都不一樣，為了避免threads互相等待的情況，我在這邊採用dynamic load balancing的策略。

實際作法是設置一個變數finished\_row，這個變數可以記錄目前執行到哪個row。當thread要拿一個row來計算時，他會拿finished\_row的下一個row，並把finished\_row++。這樣一來，如果有thread比較快完成工作的話，就可以立刻再拿一個row來執行。不過，因為可能有多個threads同時要讀取並更改finished\_row的值，所以在這邊會用mutex區隔出一個critical section，並在裡面對finished\_row做讀寫。

一張含有 文字 的圖片

自動產生的描述

1. **calculate pixel**
2. **vectorization**

當每個thread都拿到一個row時，接下來就是要計算在row中的每個pixel的值。我在這邊一開沒有做任何的優化，就是直接for迴圈遍歷整個row，結果好時非常久。因此我後來利用了vectorization的技術去實作。

因為SSE2支援128bits的vectorize，而一個double為64bits，所以在這邊可以一次執行兩個數字的計算，也就是可以同時計算兩個pixel。為了要能夠利用SSE2座計算，要將變數設為\_\_m128d的型態。

1. **load pixel into vector**

不過每個pixel所需要的計算時間也不同，我在這邊為了節省時間，將這兩個數字的計算完全獨立開來。也就是說，在vector中任一個pixel計算完後，會立刻將下一個pixel丟進來，如下圖。

一張含有 文字, 白板 的圖片

自動產生的描述

而將一個新的pixel丟進vector時，要將一些計算pixel的參數值做更新，也要更新pixel的座標(row, col)。一張含有 文字 的圖片

自動產生的描述

不過需要注意的是，因為vector裡面一定要有兩個pixel，且不能是同一個pixel，因為會有dependency的問題。所以當剩下最後一個pixel得時候，就不能用vector的方式去計算了，需要用原本double的方式計算最後一個pixel的值。

1. **computing vector**

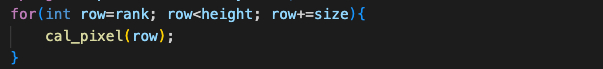
這部分就只是單純的把範例code轉換成vector的版本。一張含有 文字 的圖片

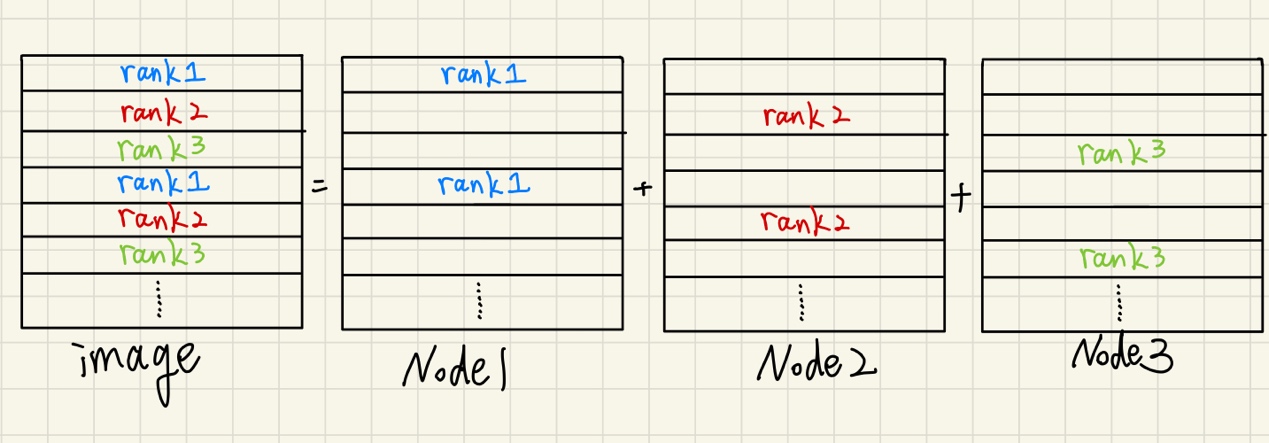
自動產生的描述

**Hybrid**

我在Hybrid部分的實作，計算方面基本是跟pthread部分是一樣的，因此在這邊會著重講解如何以MPI及OpenMP達到平行化。

1. **MPI:**

在這個部分一樣是以一個row為計算單位。不過就是用static loading的方式去做loading，會依照rank的值去分配需要計算的row。而考慮到不同的row的計算量可能會相差很多，因此在這邊會讓每個Node用穿插的方式去計算每個row，盡量讓每個Node的計算工作相同。另外，每個Node都會有自己的local\_image，而Node只需要計算他們被分配到的row就可以了，如下圖。



而當每個local\_image都計算完之後，就可以利用MPI\_Reduce()，將每個Node的local\_image加起來，這樣就會是一張完整的image了。



1. **OpenMP:**

這部分相對簡單，我主要就是將程式當中可以用for迴圈執行，且迴圈內執行的內容沒有dependency的地方，用OpenMP平行化。至於schedule的部分，考慮到每個row的季算量不同，採用dynamic的方式，並將chunk設為5。而num\_threads就設為processors的數量。我在這邊發現chunk的值對Performance有蠻大的影響，因此在這邊是了蠻多值，最後定在5。一張含有 文字 的圖片

自動產生的描述

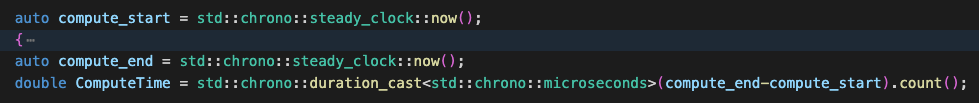
**Experiment & Analysis**

1. **Methodology**
2. System spec: 學校的系統環境

一張含有 文字 的圖片

自動產生的描述

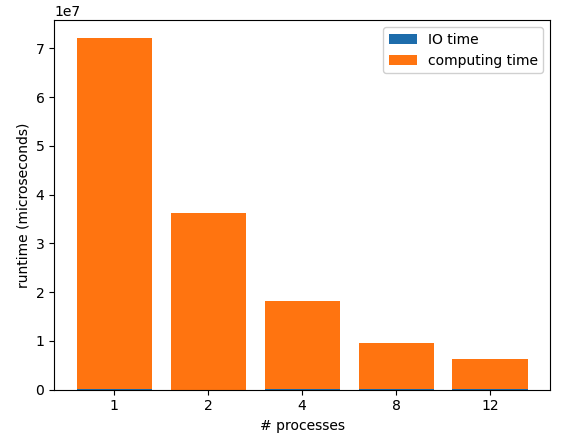
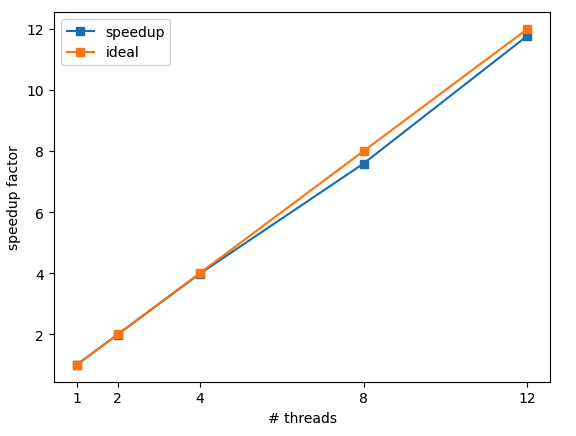
1. Performance Metrics:

****我利用std::chrono::steady\_clock去計算程式執行的時間，計算的方式如下圖。

1. **Strong Scalability & Load Balancing**

**Pthread**

1. **Speedup Factor & Time Profile**



Input parameters

從speedup factor的圖表上可以看到，當thread的數量增多的時候，speedup factor也有隨之上升。且隨著threads持續增加，speedup的程度也幾乎與ideal的線條重疊。因此可以了解到，程式的scalability非常好。我另外也把time profile給畫出來。可以看到IO幾乎沒有站任何時間。因此程式的bottleneck及為computing time。而computing time，也隨著threads的數量成比例下降，代表我的程式有很好利用threads來做運算。

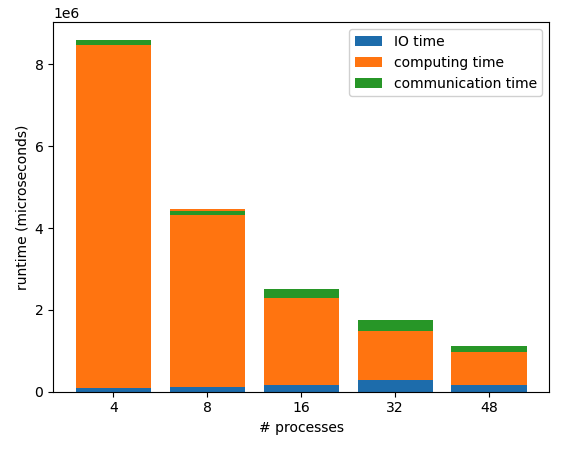
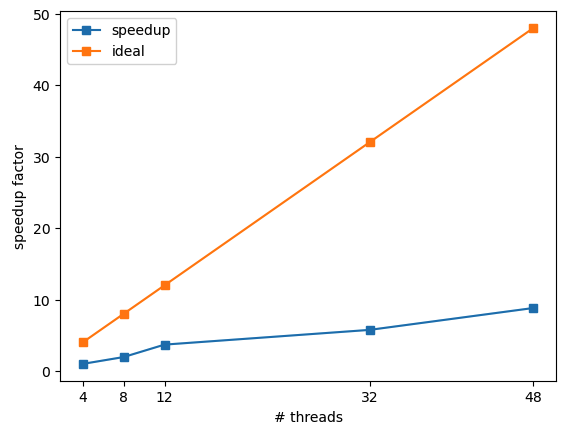
1. **Load Balancing**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | p=1 | P=2 | p=4 | p=8 | p=12 |
| Average time  (microseconds) | 72132763 | 36136066 | 18063279 | 9194396 | 6044488 |
| Diff  (microseconds) |  | 57694 | 70456 | 25836 | 51667 |
| ratio |  | 0.15% | 0.39% | 0.28% | 0.85% |

圖表中的diff代表該次任務中，執行速度最快的threads與執行速度最慢的thread的差距。可以看到diff在平均執行時間的佔比都不到0.1%，代表程式有很好的load balancing。因此我認為dynamic load balancing在這項任務中的效果很好。(Average time代表全部threads的平均執行時間，ratio代表Average time / Diff)

**Hybrid**

1. **Speedup Factor & Time Profile**

固定為4個Node，並調整threads的數量。



Input parameters

可以看到在hybrid中，speedup的程度並沒有像pthread版本一樣那麼好，我認為這是因為多了communication的tradeoff緣故。因此我也將Time Profile畫處來，並發現communication time確實有隨著processes上升而上升的趨勢。

1. **Load Balancing**

固定為4個threads，並調整Node的數量，以測試Node之間的load balancing。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | N=1  p=4 | N=2  p=4 | N=3  p=4 | N=4  p=4 |
| Average time  (microseconds) | 8524488 | 4428995 | 3002319 | 2293828 |
| diff |  | 64577 | 20845 | 97833 |
| ratio |  | 1.45% | 0.69% | 4.26% |

我在hybrid用的是static load balancing，會讓每個Node用穿插的方式去計算圖片的每個row，盡量讓每個Node的計算工作相同。不過ratio的值普遍比thread版本的還要高，代表thread版本的load balancing做得比較好。推次是因為在這邊採取static load balancing，儘管有設計一些做法希望能讓每個Node的工作量相同，不過還是比不過dynamic load balancing。

**Conclusion**

我在這一次作業花了很多時間在vectorization上面，有部分原因是因為不熟悉SSE的寫法，另一個部分是想要將vector內的pixel獨立處理，因此花了蠻多時間，不過vectorization確實使程式的表現大幅度提升。相較於Hw2a寫的很久，Hw2b一下就寫完了，因為基本的架構都在Hw2a完成了。不過Hw2b的scoreboard表現卻比Hw2a在scoreboard上的表現好很多。這次的作業使我更了解pthread、 MPI 及OpenMP。