Hw2 report			
學號: 109062233	姓名: 蘇裕恆		繳交時間:

Implementation :

1. How you implement each of requested versions, especially for the hybrid parallelism.
   在一般的版本中，我使用了pthread 並結合了sse2的方法來達成SIMD的要求。
   在hybrid 版本當中，我使用了omp與mpi的結合。基本上作法與在pthread 當中一樣，只不過每個的partition都是固定的。

kerwin	—	68	255.45	1.37	1.22	1.27	1.17	1.22	1.17	1.37	1.42	1.20	1.33
pp22s39	1	68	287.86	1.17	1.47	1.07	1.37	1.02	1.17	1.37	1.27	1.22	1.17
pp22s29	2	68	310.97	1.27	1.37	1.32	1.42	1.32	1.27	1.47	1.47	1.47	1.37

   沒有更進一步優化是因為在11/7保持rank 2 因此認為不需要太多的優化了。
2. How do you partition the task?
   兩種版本我都將 width 與 height 中比較長的那個做 split 之後再 assign 給不同的 thread。
3. What technique do you use to reduce execution time and increase scalability?
   我用的是很簡單的方法，就是將一個 row 或是 column 當作一個 Unit 並使用類似 omp dynamic 的方式來做 job assign，同時使用 sse2 來做 double 的 execution time
4. Other efforts you made in your program

```
double length_squared = 0;
/* this cause 440 when using sse2
while (cur_iter < iters && length_squared < 4) {
    double temp = x * x - y * y + x0;
    y = 2 * x * y + y0;
    x = temp;
    length_squared = x * x + y * y;
    ++cur_iter;
}
return cur_iter;*/
```

```
double x_sq = x * x ;
double y_sq = y * y ;
while (cur_iter < iters && length_squared < 4)
    y = 2 * x * y + y0;
    x =  x_sq - y_sq + x0;
    x_sq = x * x;
    y_sq = y * y;
    length_squared = x_sq + y_sq;
    ++cur_iter;
}
return cur_iter;
```

   我將左邊的方式改為右邊 這樣可以少宣告一個 temp 變數
   https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html
   https://stackoverflow.com/questions/14492436/g-optimization-beyond-o3-ofast
   我同時參考了幾個 CFLAGS 來達成最優解。

```
hw2a: CFLAGS += -pthread
# hw2a: CFLAGS += -fsanitize=address -g
hw2a: CFLAGS += -fexpensive-optimizations -ffp-contract=fast -fno-math-errno -ffinite-math-only -fno-rounding-math -fno-signaling-nans -fcx-limited-range -fno-signed-zeros -fno-trapping-math
```

   我有試過 Ofast，並會產生 wa 猜測是精準度問題使其無法正確被執行。

Experiment & Analysis & Discussion:

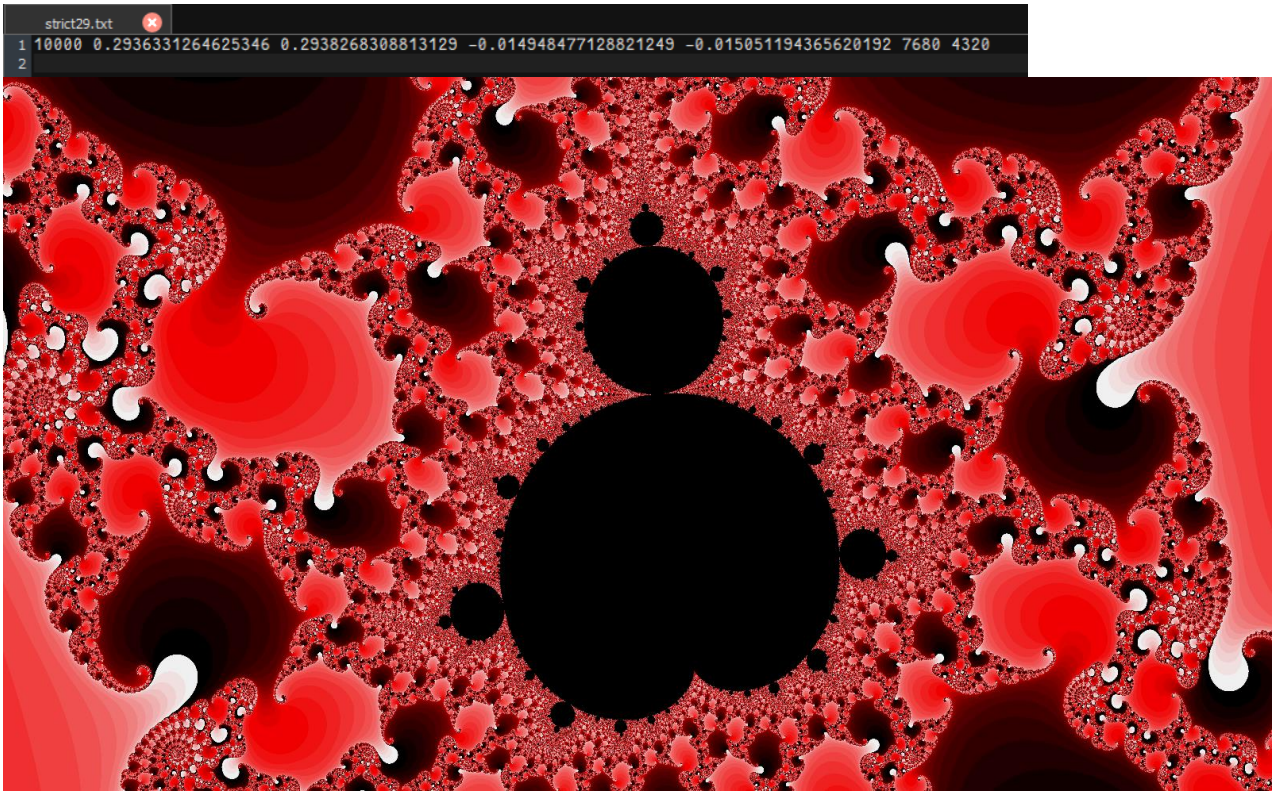
Explain how and why you do these experiments? Explain how you collect those measurements? Show the result of your experiments in plots, and explain your observations.

Methodology

1. System Spec (If you run your experiments on your own machine)
   Please specify the system spec by providing the CPU, RAM, storage and network (Ethernet / InfiniBand) information of the system.
   沒有，我使用的是 apollo
2. Performance Metrics
   How do you measure computing time of your programs? How do you compute the values in the plots?

在單個 node 的 pthread 裡面 我使用的是 <sys/time.h> 裡面的 gettimeofday()$
計算從抓完 width , height … 開始，到 write_png 前(我試過用<time.h>裡面的 clock() 來做計算 結果發現並不會跑出正確的答案) 。
至於多個 node 作法跟上次一樣，都是使用<Wtime>來計算，而計算的地方與 single node 一致。
(於 mpi init 後至 write_png 前)

ii、Plots: Scalability & Load Balancing

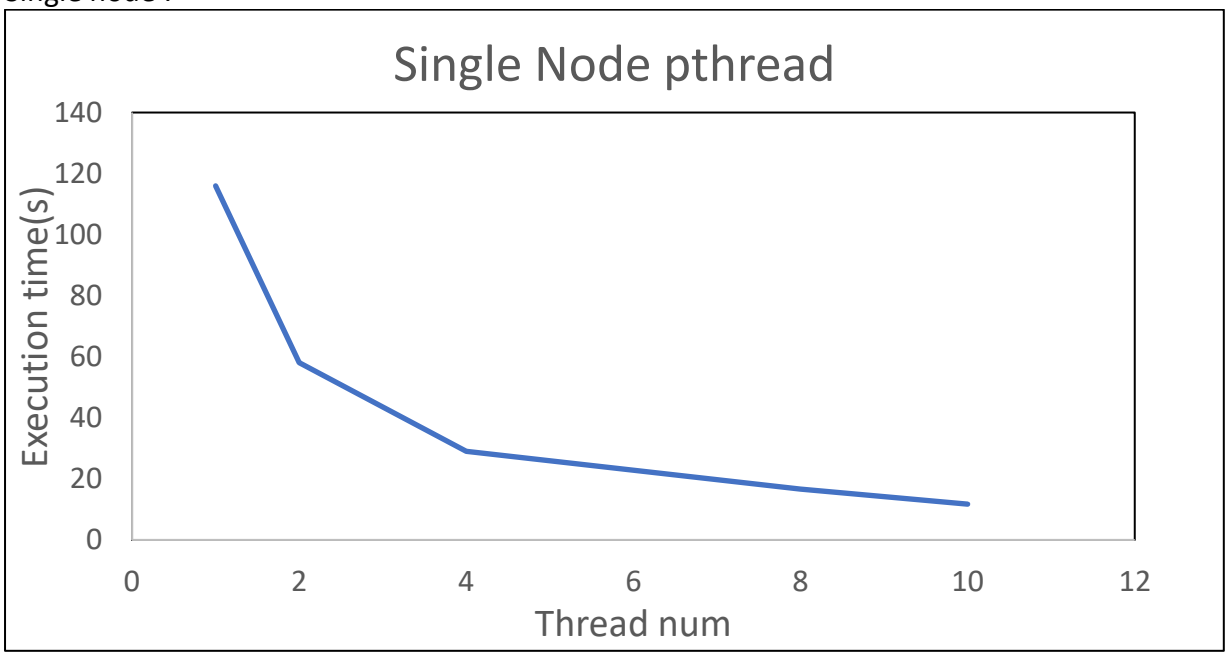


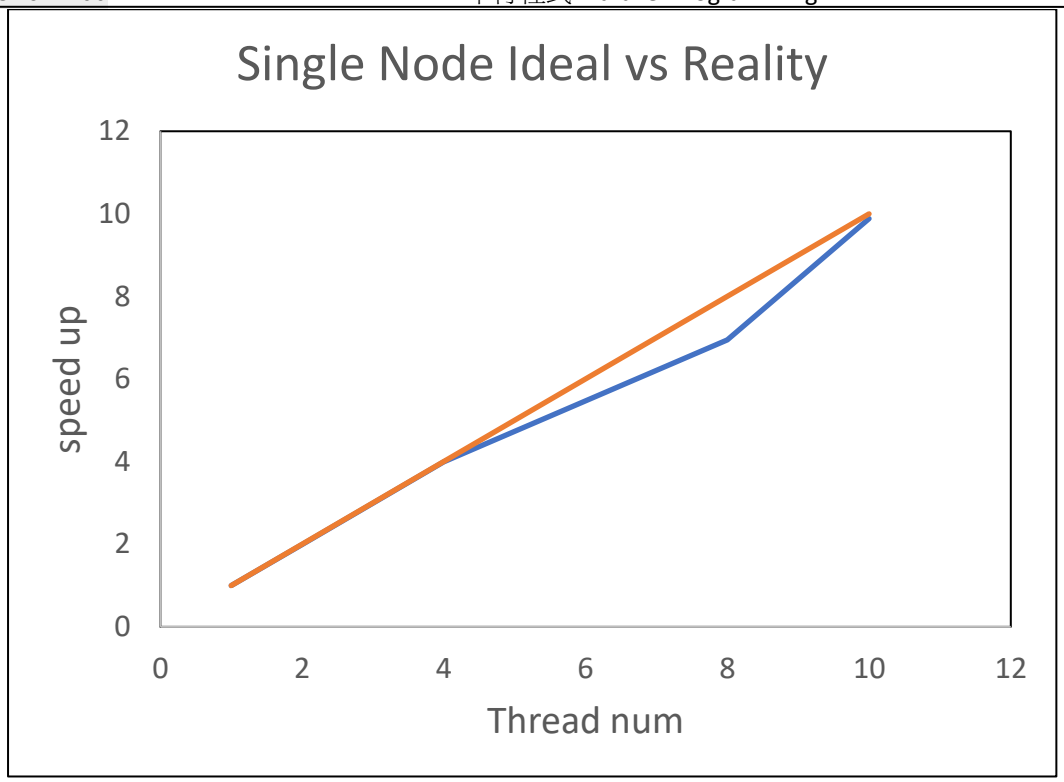
上述的圖為 strict 29 。由圖可見其複雜性與精密性。因此我選擇這張圖做為 baseline。

(a). Conduct strong scalability experiments, and plot the speedup. The plot must contain at least 4 different scales (number of processes, threads) for both single node and multi-node environments.

以下圖表 ： 橘色皆為 ideal 狀態 藍色為實際值

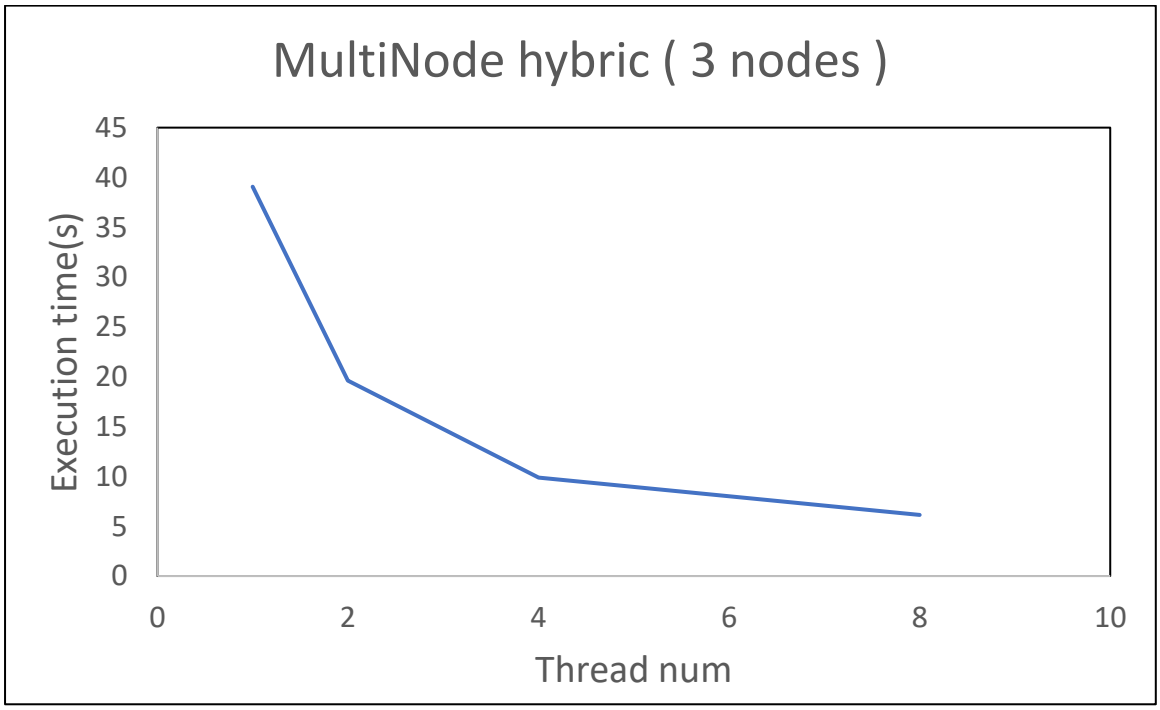
Single node :

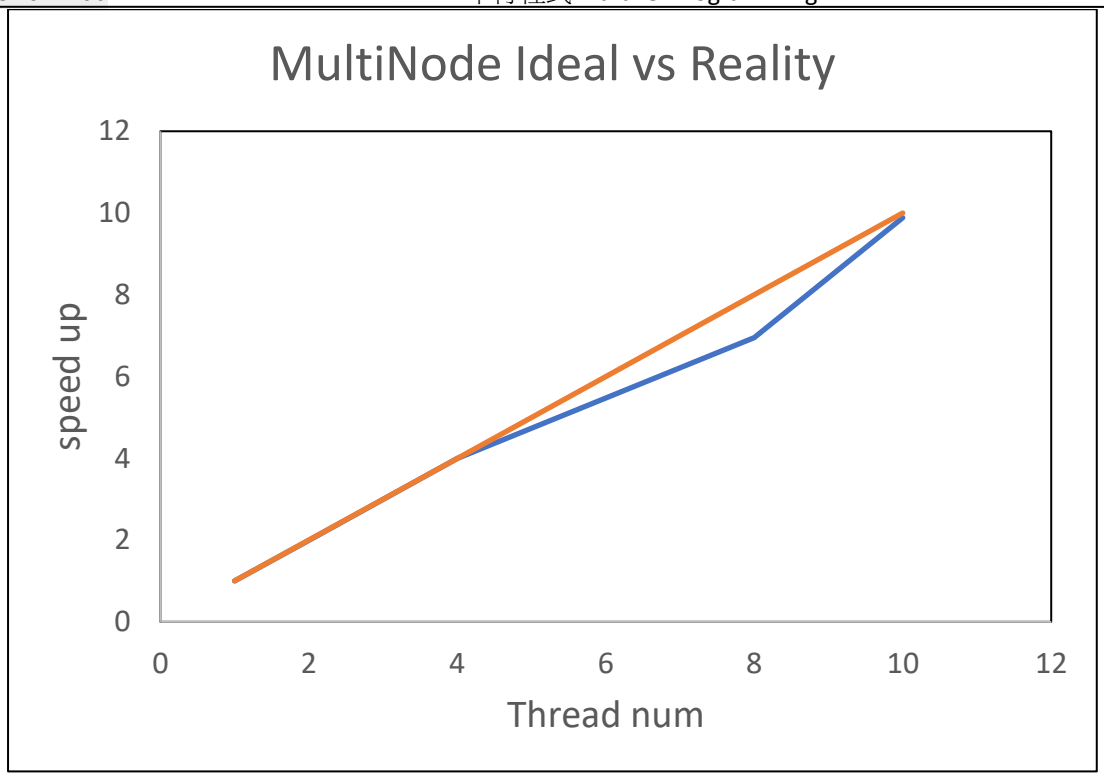




可以看到在 single node 裡面，我的 scalability 非常的好，我想撇除在 thread = 8 可能有一點差距以外基本上與 ideal 都十分接近。推測是因為 naïve 方式很不錯。而且本身這份 code 很適合做平行計算。

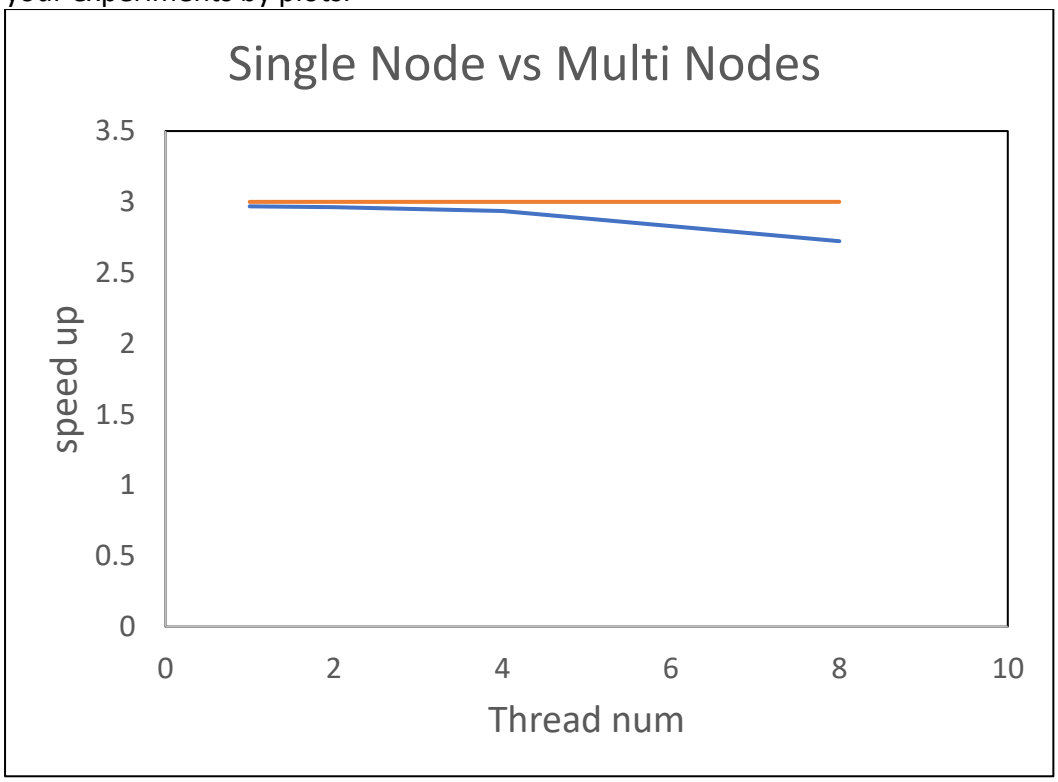
MultiNode :





在 multinode 這邊我的 scalability 也不錯，因為在一開始就已經 assign 好了每個 node 需要做哪些事情，因此我的 communication time 占比很少。猜測也是因此獲得不錯的 scalability。

(b). Design your own experiments to show how balanced it is in each of your experiments by plots.



我 combine 不一樣的 Node 數，發現基本上對於 scalability 這次是非常成功的。在最差的數據中只與理論差距不到 10%。

Others

You are strongly encouraged to conduct more experiments and analysis of your implementations.

在這次作業裡面，總共做了幾個優化。

首先我用最 naïve 的方式來做 for loop

```
if(flag){
    // printf("The height > width \n");
    // printf("The width is %d and the height is %d \n",  width ,height);
    /* the above can pass with the score 770 */
    /*double y0 = local_cnt * height_gap + lower;
    for (int i = 0; i < width; ++i){
        double x0 = i * width_gap + left;
        int repeats = 0;
        double x = 0;
        double y = 0;
        double length_squared = 0;
        while (repeats < iters && length_squared < 4) {
            double temp = x * x - y * y + x0;
            y = 2 * x * y + y0;
            x = temp;
            length_squared = x * x + y * y;
            ++repeats;
        }
        // printf("repeats : %d \n" , repeats);
        image[local_cnt * width + i] = repeats;
    }*/
```

並得到了 770 的成績，接著我使用了 sse2 將上述優化並變成了 450。

而底下是我的 final output

```
double x_sq = x * x ;
double y_sq = y * y ;
while (cur_iter < iters && length_squared < 4) {
    y = 2 * x * y + y0;
    x =  x_sq - y_sq + x0;
    x_sq = x * x;
    y_sq = y * y;
    length_squared = x_sq + y_sq;
    ++cur_iter;
}
return cur_iter;
```

我發現有一個 temp 會時時刻刻的在計算，以我們在計結的角度來看這樣會浪費掉指令。因此我將發放到外面。(sse2 的實作也有做更改) 並且調整 Cflags 進而得到了成績的 update 450 -> 416

至於 CFlag，我試了幾種以後發現其對於整個影響並不算大 (猜測主要是因為最快速的優化同時會將精準度降地進而造成 wa (-ffast-math) 而其餘的在不影響整個正確性底下能做得十分稀少。同時，沒有更進一步找尋 cflag 是因為在我前面的前提下，我無法判斷是因為 server 的奇蹟抑或是 flag 挑選的正確。

Experience & Conclusion

1. Your conclusion of this assignment、What have you learned from this assignment?
   這次作業我對於 openmp, pthread 有這深層的了解，同時也對 cflag 有個更多的想法，雖然在 flag 上無法對這次作業有多大的進步，但對於 cflag 的利弊我都有看到。
2. What difficulties did you encounter in this assignment?
   Server 過於老舊，讓每次跑出來的效果都有差距，整體會差到 20~30%進而造成很多誤判與無法真正體現差距。