多處理機平行程式設計 2021 Homework 1

學號：F74076027
姓名：林政傑
系級：資訊 111

測試環境：Linux 2.6.37.6-24-default
CPU：Intel(R) Core(TM) i7 CPU 870 @ 2.93GHz * 4: pn1, pn2, pn3(dead), pn4

Problem 1

What have you done

我將計算電路的迴圈由

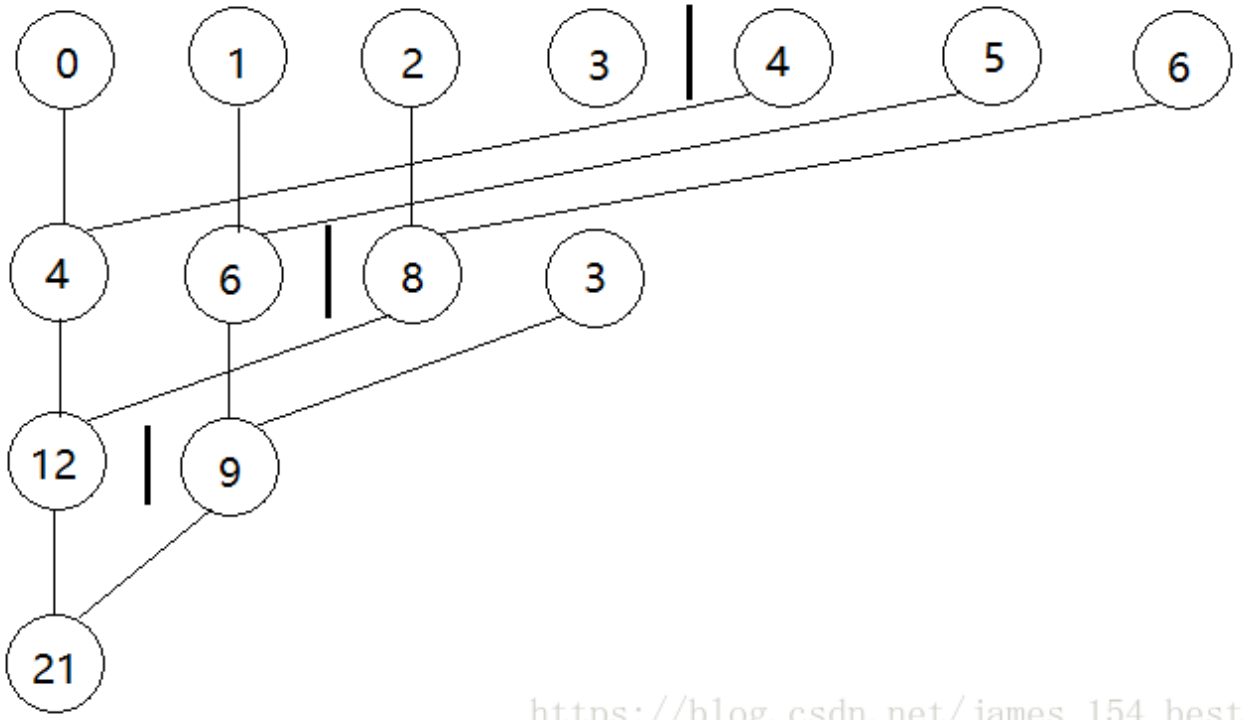
```
for (i = 0; i < UINT_MAX; i++)
    count += checkCircuit(id, i);
```

改為

```
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
for (i = my_rank; i <= USHRT_MAX; i += comm_sz)
    count += checkCircuit(id, i);
```

藉此頻均分配每個程序（process）負責的計算次數，其中 my_rank 代表該程序在 MPI 的編號。

當程序執行完 for 迴圈的計算後，會透過樹狀結構將所有 `count` 加總、傳回第 0 號程序，並印出執行時間與 `count`。樹狀結構的寫法是參考 MPI 树形和蝶形通信结构计算全局总和（reduce和all_reduce）的实现，邏輯如下圖：



假設程序數量為 n。當 n 為奇數時我將後半的程序的 count 一一傳給前半程序並且結束程序，最中間的程序則先不回傳，等待下一回合；當 n 為偶數時，就單純讓後半程序將 count 傳給前半程序加總。當最後只剩下第 0 號程序時，印出執行時間與 count。

Analysis on your result

測試**程序數量**對 wall-clock time 的影響，測試腳本如下：

```
# test_script.sh
mpicc problem1.c -o problem1
rm -rf dump
mkdir dump

for (( t = 1; t <= 16; t++ ))
do
    echo "testing for $t processes"
    touch dump/"$t".txt
    for (( i = 0; i < 10; i++))
    do
        mpiexec -n "$t" problem1 | grep 'time' >> dump/"$t".txt && sleep 0.5
    done
done

rm problem1
```

這個腳本會執行 `mpiexec -n 1 problem1` 至 `mpiexec -n 32 problem1` 各 10 次，每次間隔 0.5 秒等待 mpi 完全結束，並將執行時間記錄在 dump/*.txt 中。接著將使用的程序數對應執行時間畫成圖表：

圖表的 x 軸為使用多少程序、y 軸為消耗的時間（毫秒），分散的點為每次執行所消耗的時間，折線為使用每個 process 所消耗時間的 10% trimmed mean。

由上圖可以發現，平均調用 8 到 12 個程序來計算花費最少時間，當調用 13 個以上的程序時所耗費的平均時間反而增加。

Any difficulites?

第一次做樹狀結構時，不知道當 process 數量為奇數時該怎麼處理，後來看MPI 树形和蝶形通信结构计算全局总和（reduce和all_reduce）的实现才知道解決辦法。

Problem 2

What have you done

首先，可以讀取 argv[1] 作為 toss 的數量（預設為一億），接著利用 `mpi_Bcast` 將 toss 數量廣播給所有程序

```
MPI_Bcast(&toss, 1, MPI_LONG_LONG, 0, MPI_COMM_WORLD);
```

接著平均分配每個程序分攤的 toss 數量，並計算每個 toss 是否位於四分之一圓內：

```
MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
long long int in_circle = 0;
long long int i = my_rank;

srand(time(NULL) + my_rank);
#define RAND() ((double) rand() / RAND_MAX)
for (; i < toss; i += comm_sz) {
    double x = RAND();
```

```
        double y = RAND();
        if (((x * x) + (y * y)) <= (double)1.0) {
            in_circle++;
        }
    }
```

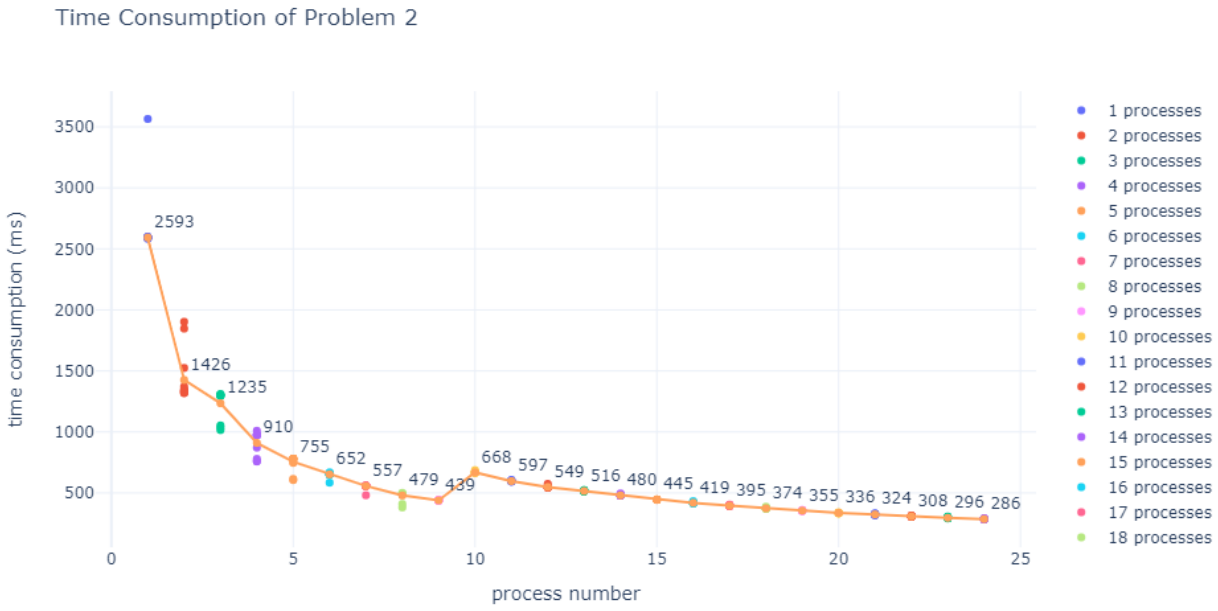
最後透過樹狀結構將所有 `in_circle` 加總、傳回第 0 號程序，透過以下：

```
    double pi = 4.0 * ((double)in_circle / (double)toss);
```

算出圓周率，並印出執行時間。

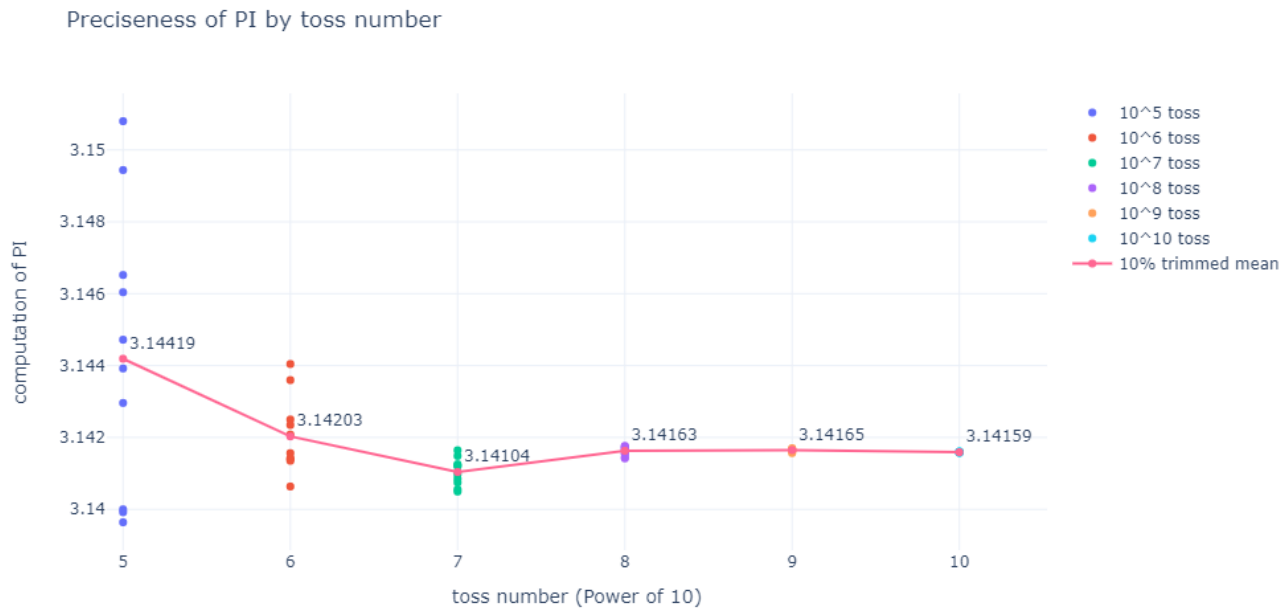
Analysis on your result

同 Problem 1，畫出執行時間的圖表：

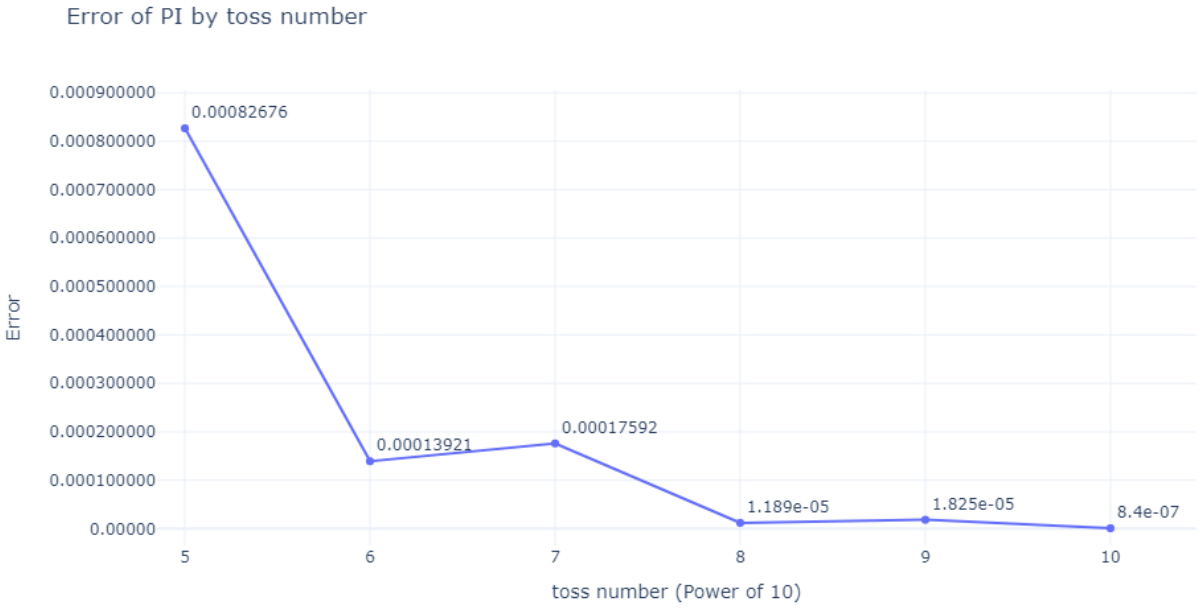


可以觀察到使用 9 個 process 執行時間的 10% trimmed mean 會減少到 439 毫秒，使用 10 個 process 又升至 668 毫秒，接下來緩慢下降。

分析 toss 數量對圓周率準確度的影響：



可以看出 toss 的次數越多，計算結果越接近圓周率。以下是 10% trimmed mean 與 3.14159265359 的誤差：



可以看到當 toss 次數為 10 的 6 次方時，計算出來的平均誤差為萬分之一點三；當 toss 次數為 10 的 10 次方時，計算出來的平均誤差為千萬分之八點四。

Any difficulites?

c 語言沒有內建的浮點亂數，爬文後找到方法製造浮點亂數：

```
#define RAND() ((double) rand() / RAND_MAX)
```