多處理機平行程式設計 2021 Homework 2

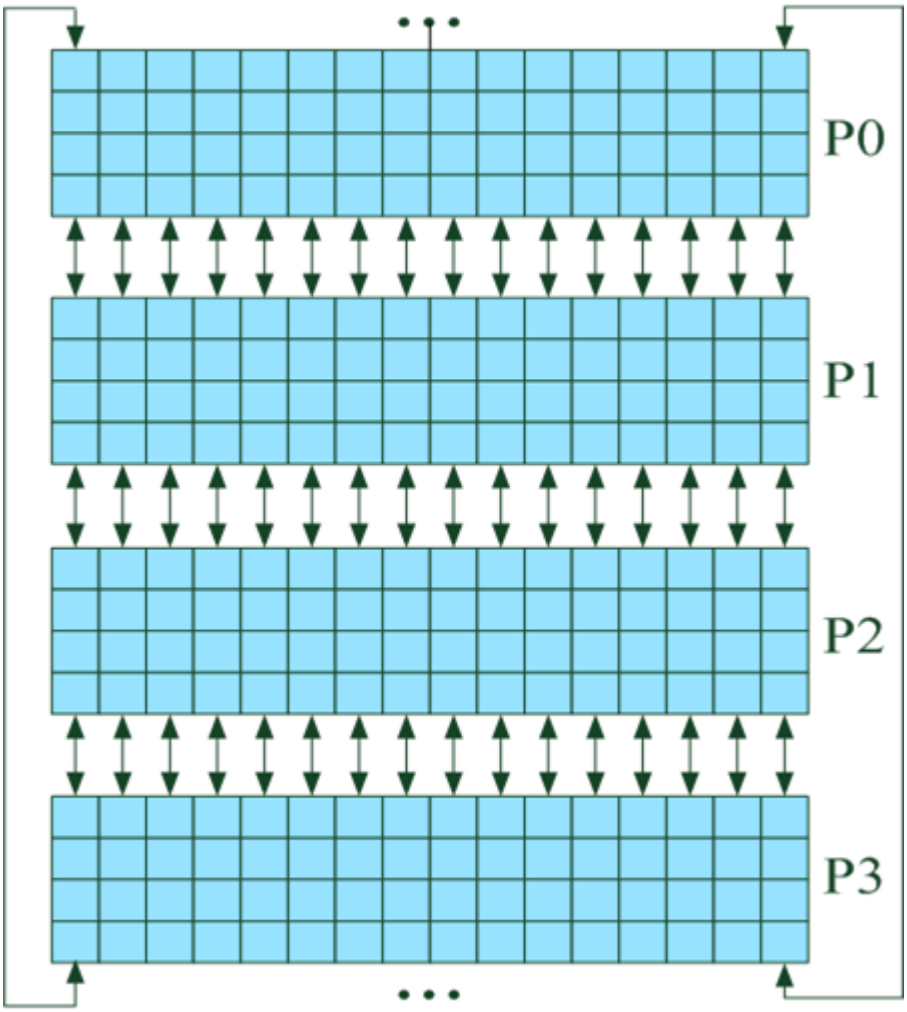
學號：F74076027
姓名：林政傑
系級：資訊 111

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

Problem 1

What have you done

按照助教提供的方法，將圖片平分若干個長方形，每一輪都透過 `MPI_Send` 與 `MPI_Recv` 向前後編號的 process 傳送邊界來進行平均。



Analysis on your result

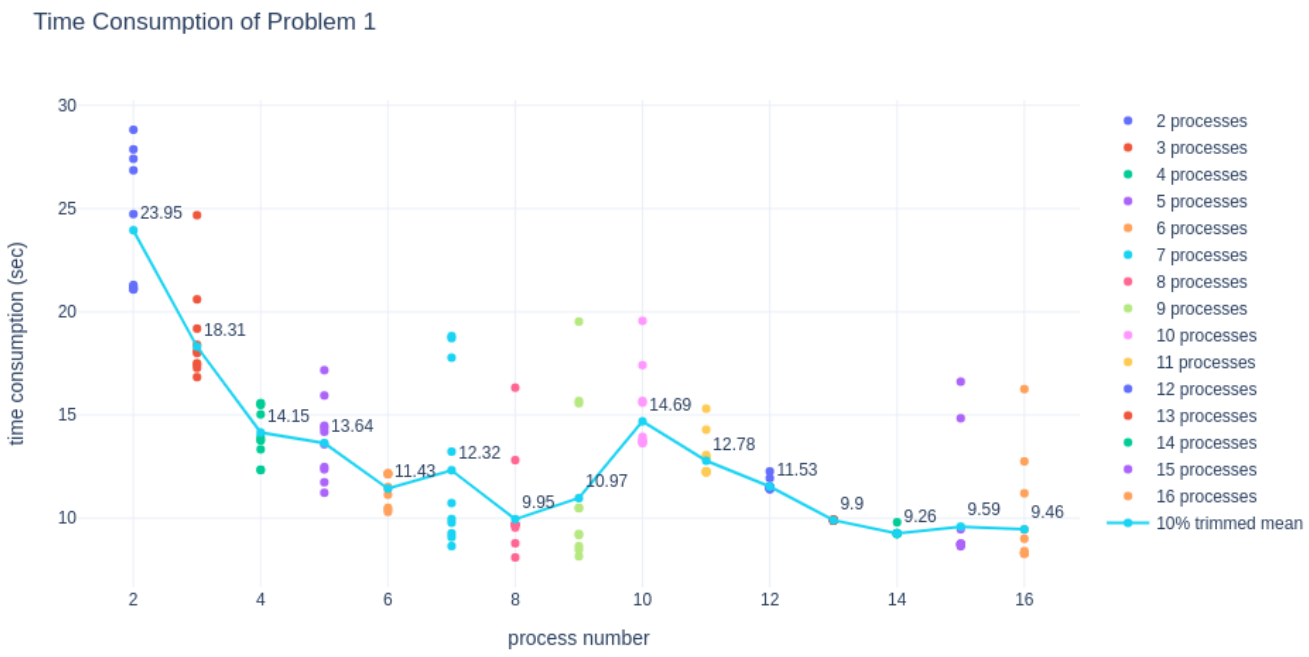
本程式必須使用兩個或以上的 process 執行。測試 **process 數量**對 wall-clock time 的影響，測試腳本如下：

```
# test_script.sh
mpiicpc h2_problem1.cpp -o problem1
rm -rf dump
mkdir dump

for (( t = 2; 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'
    done
done

rm problem1
```

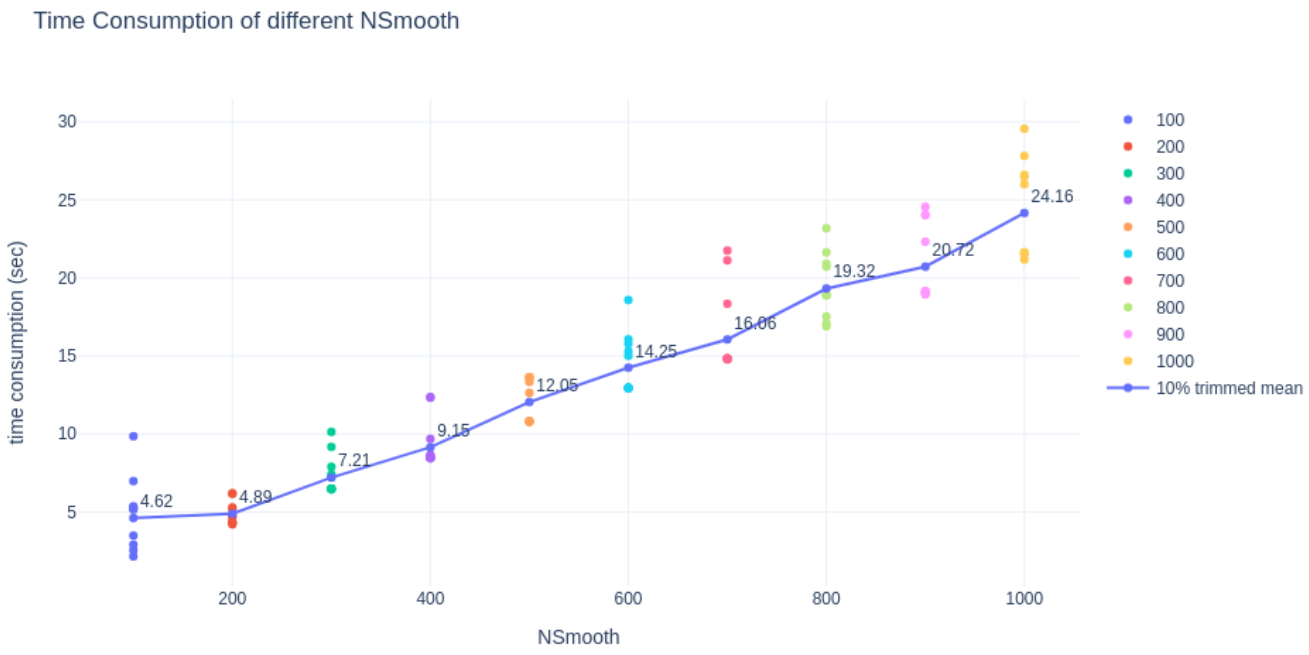
這個腳本會執行 `mpiexec -n 2 problem1` 至 `mpiexec -n 16 problem1` 各 10 次並顯示每次執行耗費的時間，接著將使用的程序數對應執行時間畫成圖表：



圖表的 x 軸為使用多少程序、y 軸為消耗的時間（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 個 process 來計算花費最少時間，當調用 9 到 12 個 process 所耗費的平均時間反而增加。我認為調用 9 到 12 個 process 時間比 8 個 process 多，是因為 process 越多，每一輪要等待所有 process 同步邊界的時間就越長，又或著 MPI 對這個區間的 process 數量的處理效率比較差。

接下來測試 NSmooth 的值對時間的影響，以下測試都是固定使用 2 個 process：



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

可以發現增加 NSmooth 所增加的時間幾乎是固定的。

Any difficulites?

一開始我想要只讓 process 0 讀取圖片，然後再把圖片資料 Scatter 到其他 process，但是不管怎麼試都會 runtime error，只好妥協，讓每個 process 都讀取圖片。

```
// WRONG
if (id == 0) {
    if (readBMP(infileName))
        cout << "Read file successfully!!" << endl;
    else
        cout << "Read file fails!!" << endl;
}

// CORRECT
if (readBMP(infileName))
    cout << "Read file successfully!!" << endl;
else
    cout << "Read file fails!!" << endl;
```

Problem 2

What have you done

按照投影片提供的邏輯，將步驟分為 even phase 和 odd phase，透過 MPI_Sendrecv 與另一個 process 共享資料。

Time	Process			
	0	1	2	3
Start	15, 11, 9, 16	3, 14, 8, 7	4, 6, 12, 10	5, 2, 13, 1
After Local Sort	(9, 11, 15, 16	3, 7, 8, 14)	(4, 6, 10, 12	1, 2, 5, 13)
After Phase 0	3, 7, 8, 9	(11, 14, 15, 16	1, 2, 4, 5)	6, 10, 12, 13
After Phase 1	(3, 7, 8, 9	1, 2, 4, 5)	(11, 14, 15, 16	6, 10, 12, 13)
After Phase 2	1, 2, 3, 4	(5, 7, 8, 9	6, 10, 11, 12)	13, 14, 15, 16
After Phase 3	1, 2, 3, 4	5, 6, 7, 8	9, 10, 11, 12	13, 14, 15, 16

程式碼參考了 https://github.com/ashantanu/Odd-Even-Sort-using-MPI

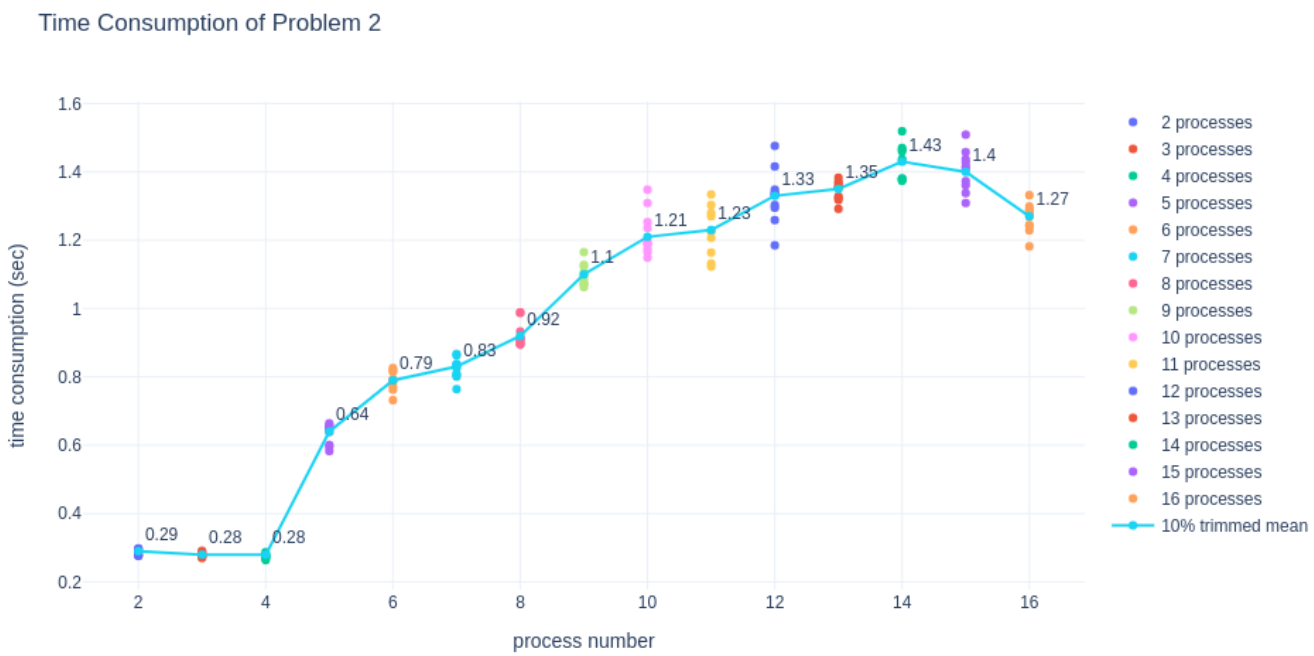
亂數的數量 `n` 預設為 100，可以透過參數改變，例如：

```
$ mpiicc h2_problem2.c -o h2p2
$ mpiexec -n 4 h2p2 500
```

即可將亂數改為 500 個

Analysis on your result

本程式必須使用兩個或以上的 process 執行。測試 **process 數量**對 wall-clock time 的影響，執行 `mpiexec -n 2 h2p2 1000000` 至 `mpiexec -n 16 h2p2 1000000` 各 10 次並顯示每次執行耗費的時間，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 process、y 軸為消耗的時間（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，調用 process 越多耗費時間反而越長，我認為是因為 process 越多，就必須要越多個 phase 來排序，反而降低效率。

Any difficulites?

無