多處理機平行程式設計 2021 Homework 4

學號：F74076027
姓名：林政傑
系級：資訊 111

測試環境：Linux ubuntu 5.11.0-40-generic
CPU：Intel i7-10750H (6 core 12 thread) @ 2.6GHz
GCC version：9.3.0

What have you done

按照助教提供的方法，將圖片平分若干個長方形，分發給 child thread 來模糊。首先主程式透過 `argv` 讀取 n_thread （預設為 4）與 NSmooth（預設為 1000），接下來平均分配長方形區域給 child thread，然後進入模糊處理的迴圈。

在迴圈的一開始，主程式將當前的圖片資料 swap 到暫存陣列，接著將 `ready_child` 陣列中的每個元素設為 `false`，然後發送 `sem_post()` 給每個 child thread 進行模糊，當 child thread 處理完畢，會將屬於自己的 `ready_child` 設為 `true`，主程式在確認每個 child thread 都執行完後，就會進行下一輪迴圈，直到 `NSmooth` 執行完次模糊。大致 pseudo code 如下：

```
for (int count = 0; count < NSmooth; count++) {
    // 把像素資料與暫存指標做交換
    swap(BMPSaveData, BMPData);

    // send semaphore to all child threads
    for (int i = 0; i < n_thread; i++) {
        ready_child[i] = false;
        sem_post(&semaphore[i]);
    }

    // wait for all child threads complete
    bool busy = true;
    while (busy) {
        busy = false;
        for (int i = 0; i < n_thread; i++) {
            if (ready_child[i] == false) {
                busy = true;
                break;
            }
        }
    }
}
```

Analysis on your result

首先測試 **child thread** 數量對 wall-clock time 的影響，測試腳本如下：

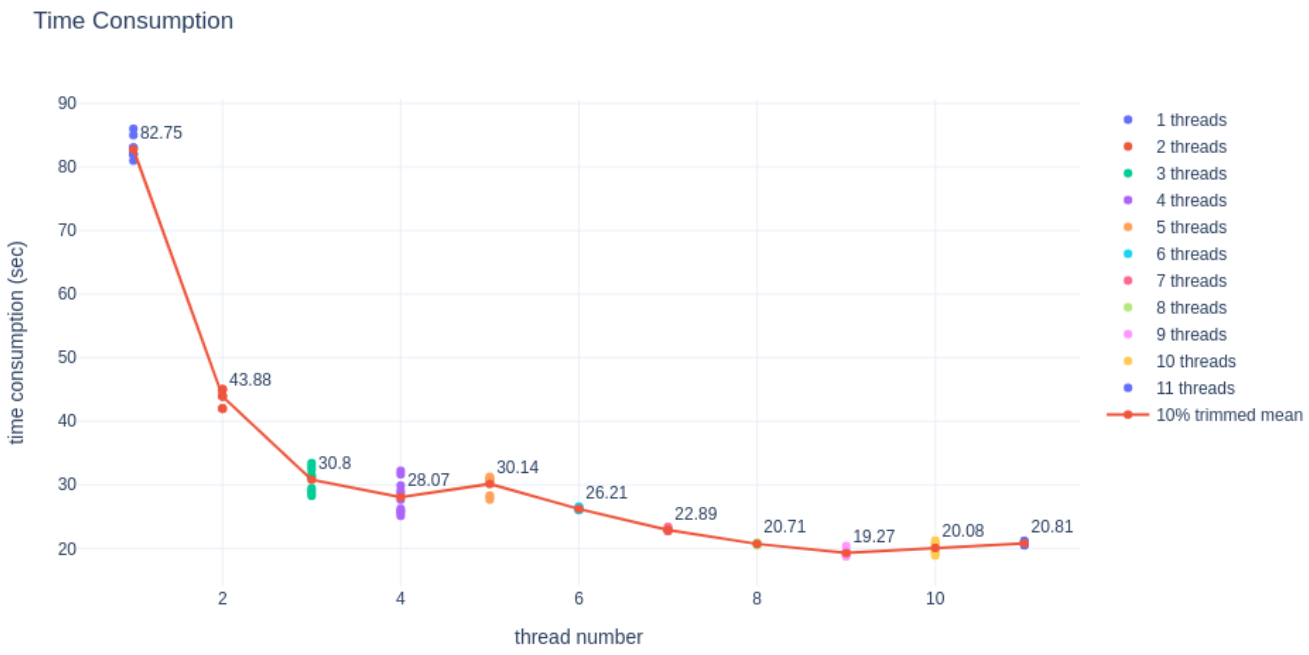
```
# test_script.sh
g++ h4_problem1.cpp -o a -lpthread

for (( t = 1; t <= 11; t++ ))
do
    echo "testing for $t threads"
    for (( i = 0; i < 10; i++))
    do
        ./a "$t" | grep 'time'
    done
done

rm a
```

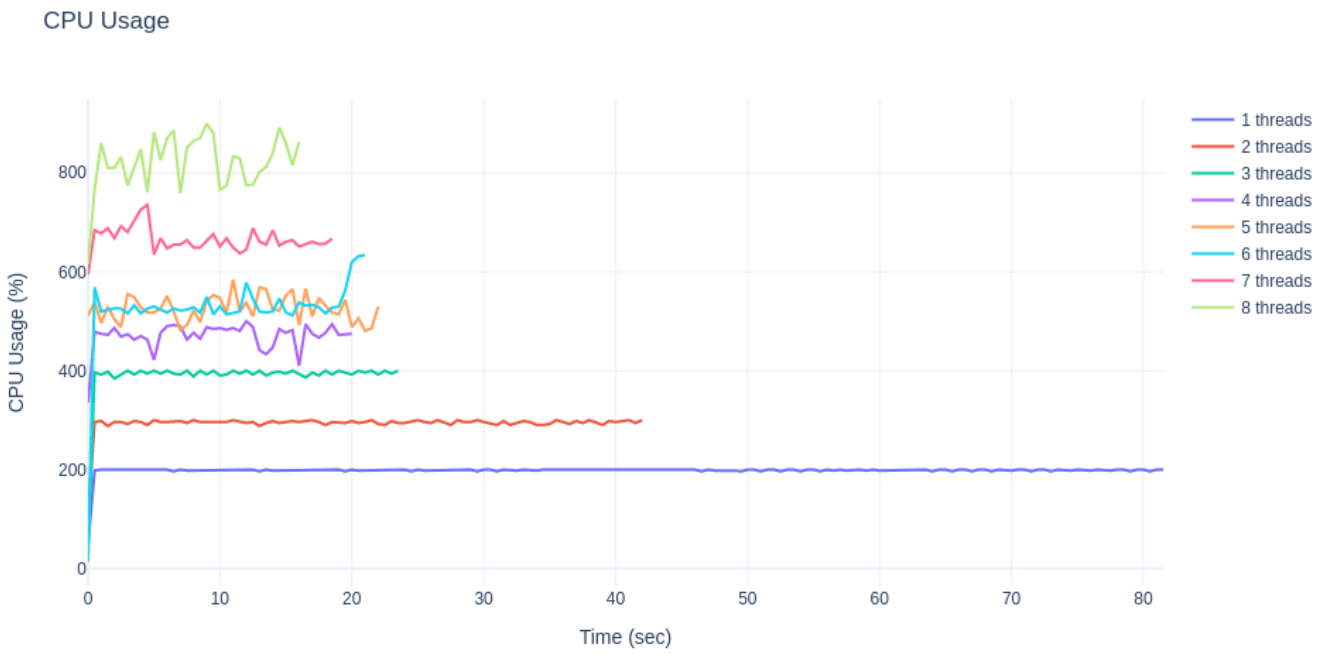
這個腳本會執行 `./a 1` 至 `./a 11` 各 10 次並顯示每次執行耗費的時間，接著將使用的 child thread 數量對應執行時間畫成圖表：

圖表的 x 軸為使用多少個 child thread、y 軸為消耗的時間（秒）、分散的點為每次執行所消耗的時間、折線為使用不同數量 child thread 所消耗時間的 10% trimmed mean。



可以發現用一個 thread 需要耗費 80 秒左右完成運算，用 9 個 thread 則能降低至 19.2 秒左右，再使用更多的處理器則可能因為過多 context switch 反而增加計算時間。我還觀察到一件很有趣的事，當 thread 越多，執行時間的 jitter 就越小，也就是每次執行所消耗的時間不會有太大的差異，這也許意味著把工作劃分的，小一點對於 Linux 系統能更有效的排程。

測試 thread 數量與 cpu 使用率的關係，固定 NSmooth 為 1000。首先在一個終端機執行 `top -d 0.5 -b | grep h4_problem1` 每 0.5 秒在 top 中抓取程式佔用的資源，然後在另一個終端機執行 `./h4_problem1 1` 至 `./h4_problem1 8` 然後將 cpu 使用率畫成折線圖：



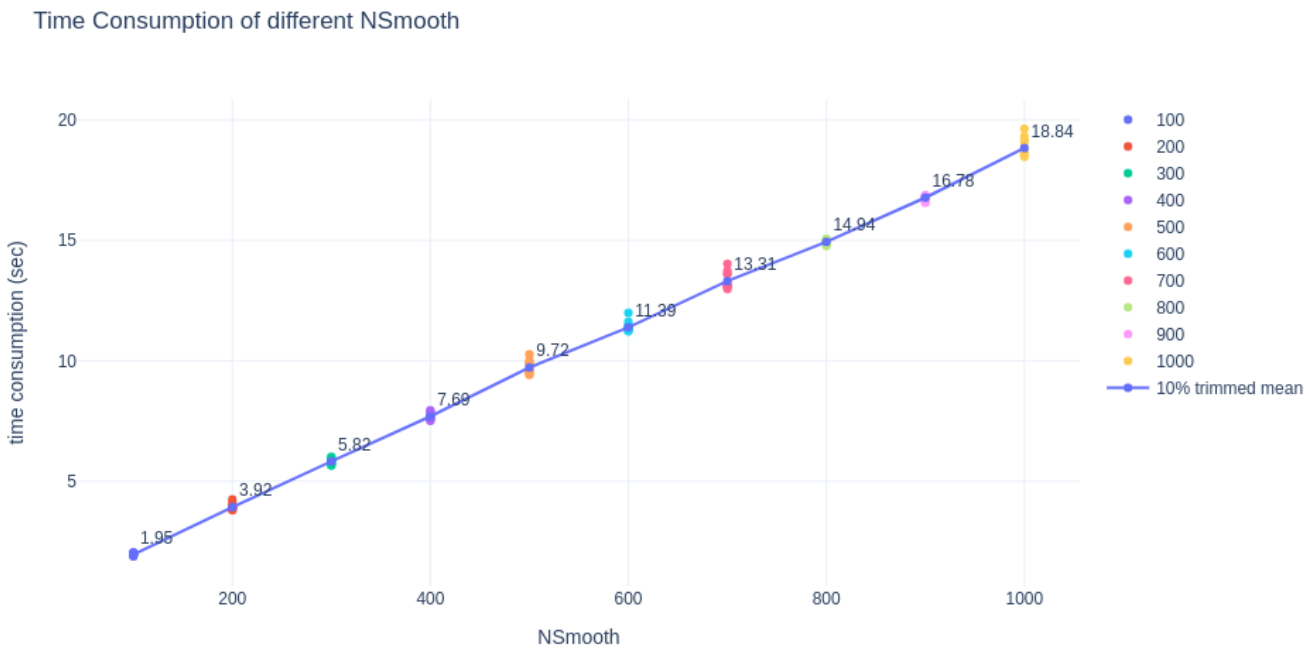
因為我的 CPU 有 12 個邏輯核心，所以 CPU 使用率的上限是 1200%。從途中可以看出當 n_thread 等於 1 時，共有 main thread 和一個 child thread，使用率為 200%，依此類推。n_thread 越大執行時間越短，但瞬時的 CPU 佔用率也高。比較特別的是 n_thread 等於 5 和 6 時，佔用率都維持在 550% 左右，重複實驗若干次還是一樣的結果，找不出原因。

接下來測試 NSmooth 的值對時間的影響，以下測試都是固定使用 9 個 child thread：

```
# test_script.sh
g++ h4_problem1.cpp -o a -lpthread

for (( t = 100; t <= 1000; t = t + 100 ))
do
    echo "testing for $t NSmooth"
    for (( i = 0; i < 10; i++))
    do
        ./a 9 "$t" | grep 'time'
    done
done

rm a
```



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?

以前有幾次作業都使用過 pthread 所以程式部份沒有什麼問題。

量測時間時，發現使用 `time(NULL)` 只能量到以秒為單位的時間、使用 `clock()` 會量到 cpu time，後來找到 `clock_gettime(CLOCK_REALTIME, &begin)` 可以量測較為精確的 wall-clock time。但是繳交作業的 server 不支援 `clock_gettime()`，所以註解掉了。