Multi-core Programming Final Project

9617145 許晏峻
9616167 蔡孟儒

1. 實驗目的

驗證和運用課堂上所學習的 multicore 平行化方法。

2. Part1: bfs

2.1. 分析觀察過程

經由 gprof 分析後，我們發現最耗時的部分為 graph_from_edge_list 和 bfs 這兩個 function，因此致力於在這裡做平行處理；而在 graph_from_edge_list 中，我們找到迴圈可以平行的部分：

```
for (e = 0; e < G->ne; e++) {
    i = G->firstnbr[tail[e]]++;
    G->nbr[i] = head[e];
}
```

2.2. 加速方法

在上述的迴圈使用 OpenMP 來平行處理，只要在迴圈前加上：

```
#pragma omp parallel for
```

，並且將 i 宣告成 for 迴圈內的區域變數即可。

2.3. 效能表現

原始程式速度：20.9992 秒。
加速後速度：10.6276 秒。
加速約 49.39 %。

3. Part2: ray_tracing

3.1. 分析觀察過程

一開始經由 gprof 去觀察花最多時間的 function 為何，最後發現為 ray_sphere 這個 function，而後我們就開始向上 trace 到底是哪層用 for 迴圈 call 最多次，經過 trace 後發現 render function 是 for 迴圈跑最多次的，故我們就朝此處下手。

3.2. 加速方法

我們加速方式是將外層的 for 迴圈，利用 pthread 的方式將他平行化；而唯

一會有礙於平行化的就是 fb 這個 data structure，因他的 fb 是每次做完才指向下一個。而我們利用的方式是動態宣告一個 array，讓 array 的每一格指向 fb 的每個 node，如此一來便可以藉由給每個 thread 適當的參數，來將外層的 for 迴圈做平行化。

3.3. 效能表現

原始程式速度：180.171 秒。

加速後速度：41.625 秒。

加速約 76.9 %。

4. Part3: mpeg4_encoder

4.1. 分析觀察過程

也是先使用 grpof 做程式架構分析，並且在輸出結果中發現最花時間的 function 為 transfer_8to16sub_c、quant_inter_c、fdct_int32、sad16_c；這 4 個 function 內部雖然都是迴圈計算的程式，但迴圈次數都很小，我們嘗試過將迴圈展開，對於效能並沒有改變，有時候反而還拖慢執行速度；而耗時的主要原因都是被呼叫非常多次累計的，所以對於對這些 function 內部做平行化效果並不大，應該對於呼叫這些 function 的地方做平行處理。

所以往外一層看到的是 FrameCodeP 這個 function，這裡的看似可以加速的地方為一個兩層的 for 迴圈，迴圈內程式碼有約 1000 行，並且大量使用到同一個 structure 做各種運算，因此太多的 critical section 讓我們無法對這裡做平行化的處理。

因 FrameCodeP 的兩層 for 迴圈不大可能平行化，所以我們去看了它裡面所包的一千多行 code，發現到它是利用 sequencial 的方式去做六個 block 的處理，因此我們將六個 block 的處理寫成一個 for 迴圈，再利用 openMP 的方式去平行化處理，但後來發現這樣做並不會比較快，因為畢竟這樣做 for 迴圈只跑六次，且每次跑的時間也是相當短促，做平行化反而會造成更多時間在處理每個 thread 的開始與結束。所以最後再分析往外一層呼叫 FrameCodeP 的地方：從 encoder_encode、xvid_encore、enc_main 直到 main 才有迴圈可以觀察。

main function 的 encoding loop 的部分，也因為不斷使用同一個 structure 造成難以平行化，也許需要重新修改程式架構才有機會加速成功，因此到這裡皆無功而返。

4.2. 加速方法

無。(但事後發現 gcc 使用‘-O3’參數效能會變好，約快 10 秒)

4.3. 效能表現

原始程式速度：42.554 秒。

加速後速度：42.554 秒。

加速 0 %。

5. 結論心得

這個 final project 的計分方式和以往不同，比較的是和其他組別相對的效能提升，而不是絕對的一個標準值，這讓這個 final project 變得更有趣，雖然最後我們的名次實在不高。

在嘗試分析程式的過程中，我們漸漸地養成不再沒有想法地就直接從頭開始看程式碼的習慣，而是改成運用某些特定的程式分析工具來協助我們，找到較為重要的部分後，再針對這個部分詳細地看要如何去加速程式，判斷是否該網上一層作平行化，或是該修改程式的架構…等。

而在加速程式的過程中，也等於是複習了很多上課時的概念，當然最重要的就是多執行緒的概念，我們更發現 OpenMP 並沒有我們想像中的比 pthread 還不如，而 CUDA 最強這樣子的想法，大部分的時間我們都是使用 OpenMP 做測試的，既快速又容易撰寫；但還有許多小地方例如編譯器的最佳化參數，這在這個 final project 占了不小的地位，由於第一次 demo 我們的程式錯誤讓我們以為'-O3' 真的沒有比'-O2 快，讓我們 part3 沒有加速成功，實在是個遺憾；而我們最忽略的地方應該是 inline function 的部分，在 part2 中可看出效能會有所差異，還有 part3 也許可以讓這個方法加速，但當時並沒有想到這個地方…等。

總而言之，完成這次 final project 的過程中，無形地幫助我們又複習一次這學期課程中的許多重點，然後讓我們實作這些東西的過程才是最可貴的。