System Programming HW4 - Report

b04902053 資工二 鄭淵仁

(b) thread 開在哪裡，分工在哪裡？

我總共把 thread 開在兩個地方：

(1) 在**建 decision tree** 的時候，對每一棵 tree 我都開一個 thread 去建。然後也順便開一個 thread 去讀取 testing_data。

⇨ 所以程式可以同時建很多棵 decision tree，還順便同時讀取 testing_data。

(2) 在 **traverse decision tree** 的時候，也是對每一棵 tree 都開一個 thread 去 traverse。

⇨ 所以程式可以同時 traverse 很多棵 decision tree。

(c) 畫出 thread 數量與時間的比較，以紅色標出時間最快的位置，並說明此圖表。

表一 decision tree 固定為 **30** 棵

Thread 的數量	5	10	30	50	70	100	200
Real time	14.653	8.9808	6.1486	4.658	5.3614	4.2528	4.2544

表二 decision tree 固定為 **50** 棵

Thread 的數量	5	10	30	50	70	100	200
Real time	14.7798	12.1232	8.2236	6.1352	5.5706	6.8894	6.3934

表三 decision tree 固定為 **70** 棵

Thread 的數量	5	10	30	50	70	100	200
Real time	24.392	18.224	10.2854	9.3068	7.9792	7.4498	8.067

以上所有數據都是做三次之後平均的結果。

三個表格中，紅框是時間最快的位置，綠框是 thread 跟 tree 的數量一樣多的位置。

觀察這三個表格，可以發現以下三點：

(1) 當 thread 比 tree 少的時候，**thread 越多，real time 越短**。

⇨ 我認為這是因為有越多的 thread，就可以同時建更多的 tree 和同時 traverse 更多的 tree，所以 real time 會縮短。

(2) 當 thread 增加到**大於** tree 的數量時，**real time 幾乎沒有變化**。

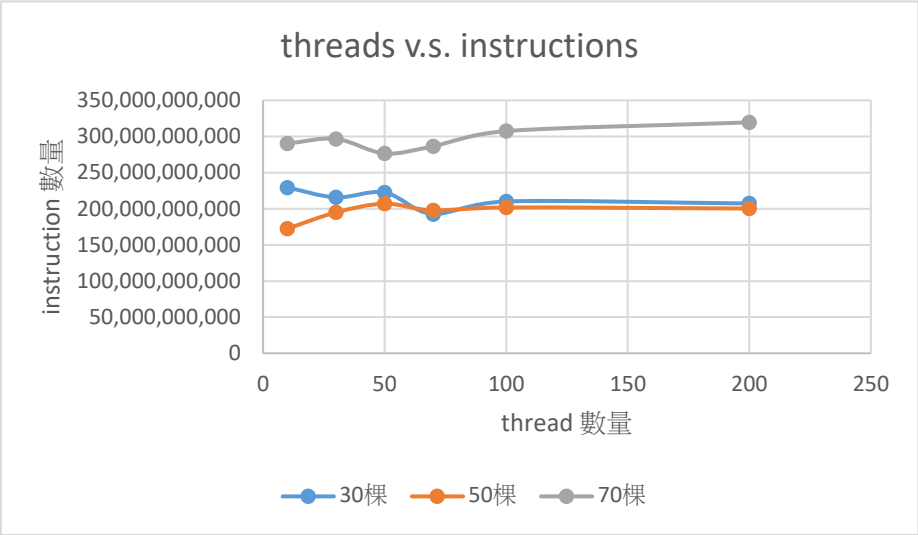
⇨ 我認為這是因為需要建的 tree 和需要 traverse 的 tree 只有固定的數量，所以如果超過了那個數量，多出來的 thread 就不會被用到，所以 real time 就不會縮短。

⇨ 這個部分依照不同的建樹方式，可能會有不同的結果。而以我而言，我只有在建 tree 和 traverse tree 的時候開 thread 去平行化運算，所以結果會是 real time 不變。

(3) 在表格中時間最快的位置（紅框），與左右的時間差很小，可以算成都一樣。

⇨ 在第(2)點中已經知道，當 thread 數量大於 tree 數量的時候，real time 的大小就差不多了，所以時間最快的位置其實只要在綠框以後，哪一個都有可能。

(d) 畫出 thread 的數量與 instructions 數量的比較，並說明此圖表。



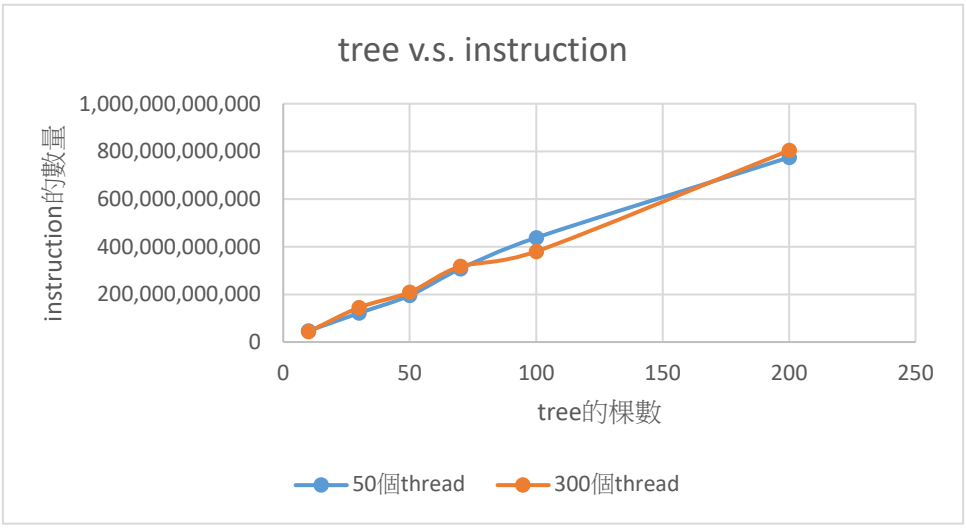
上圖是當 decision tree 數量固定為 30 棵（藍線）、50 棵（橘線）和 70 棵（灰線）時，thread 與 instruction 數量的比較圖。所有的數據都是做三次之後取平均的結果。

觀察圖中的三條線，可以發現當 decision tree 棵數固定的時候，thread 的數量**不會影響** instruction 的數量。

⇨ 我認為這是因為：雖然開了很多個 thread，但是要建的 tree 的數量還是一樣，而建 thread 本身又不會增加太多的 instruction。

⇨ 所以 instruction 的總量並不會改變，只是分配到不同的 thread 裡面而已。

(e) 畫出樹的數量與 instructions 數量的比較，並說明此圖表。



上圖是當 thread 個數固定為 50（藍線）和 300（橘線）時，tree 與 instruction 數量的比

較圖。所有的數據都是做三次之後取平均的結果。

觀察圖中的兩條線，可以發現當 thread 數量固定的時候，**tree 的棵數越多，instruction 數量也會越多**。

- ⇨ 我認為這是因為有越多棵 decision tree，就要有越多的 instruction，才能建更多的 tree 和 traverse 更多的 tree。

(f) 其他發現

我總共有三點發現：

(1) 如果依照投影片上計算切點的公式來計算最佳切點，則很容易會計算出最佳切點是切到邊緣的位置，那麼就很容易把樹建得很深，跑起來會很久。
   - ⇨ 所以我用我在網路上找到的不一樣的計算切點的公式來計算切點，公式的做法是：在總和左右節點的 gini impurity 的時候，不是直接加起來除以 2，而是**根據左右節點的資料數量做加權平均**。

(2) 投影片上有提到要取跟 training_dataset 等量的資料去做 decision tree，但是我發現取那麼多的資料來做 decision tree 很花時間。
   - ⇨ 所以我取**少一點的資料**，但是**做更多棵 decision tree** 來投票。結果發現正確率更高，而且可以用更多的 thread 做平行化運算，所以花的時間更是少非常多。

(3) 我發現 testing_data 裡面的 0 比 1 多很多（助教也有寄信通知），而我的程式輸出時，有錯誤的話，幾乎全部都是正確答案是 1，而我的程式卻輸出 0 的情況，
   - ⇨ 所以我在投票的時候，**讓 1 的票數變成 2 倍**，再去比較 0 和 1 的數量。結果正確率提高很多。
   - ⇨ 我也另外寫了一支程式去計算我的答案錯誤時，輸出結果是 0 或是 1 的數量，發現讓 1 的票數變 2 倍時，1 和 0 的比例大約是 1/10，與 testing_data 的比例接近，所以變成 2 倍是好的解法。