SP HW4 Report

呂佳軒

January 16, 2018

1 試說明你將thread開在哪裡，是分工在哪裡

在第117行main thread完成初始化之後，會利用一個迴圈建立剩下的thread_num-1個thread。

```
// Initialize thread
  1     for (int i = 0;i < num_thread - 1;++i){
  2         pthread_t tid;
  3         int err = pthread_create(&tid, NULL, thread_entry, NULL);
  4         if (err){
  5             fprintf(stderr, "create thread err\n");
  6             exit(1);
  7         }
  8     }
```

thread被建立後，會進入一個無窮迴圈(258行)等待main thread把job串上job_list。

```
// Main loop
  1     while (1){
  2         Job* job_now;
  3         pthread_mutex_lock(&lock_of_job);
  4         while (job_list == NULL) pthread_cond_wait(&cond_of_job, &lock_of_job);
  5         job_now = job_list;
  6         job_list = job_list->next;
  7         pthread_mutex_unlock(&lock_of_job);
  8     }
```

再來main thread會把job放到job_list中，總共有三種，construct tree(128行)、make decision(178行)、跟thread terminal(209行)。

```
// Given job of construct tree
  1     srand(time(NULL));
  2     for (int i = 0;i < num_tree;++i){
  3         pthread_mutex_lock(&lock_of_malloc);
  4         Job* new_job = (Job*)malloc(sizeof(Job));
  5         int* index = (int*)malloc(sizeof(int) * num_data);
  6         pthread_mutex_unlock(&lock_of_malloc);
```

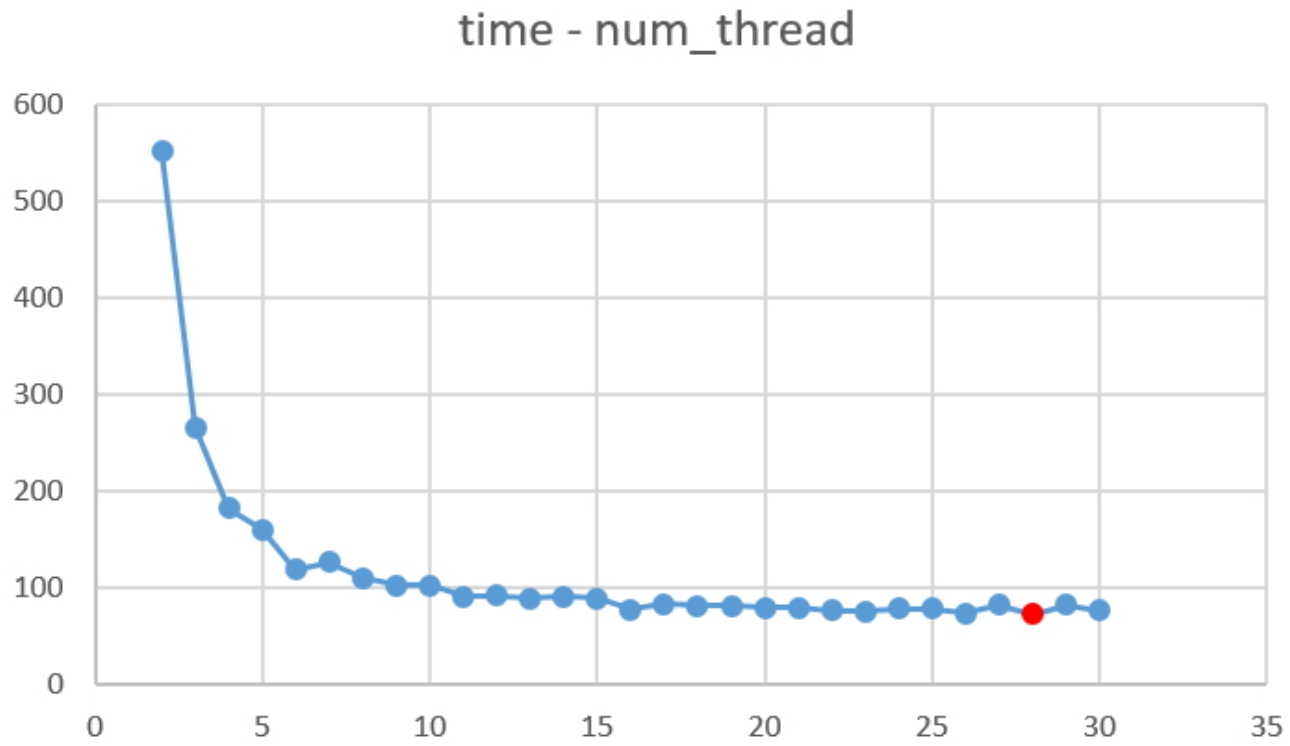
```
 7
 8         for (int i = 0;i < num_data;++i) index[i] = rand() % num_data;
 9         new_job->type = 1;
10         new_job->index = index;
11         new_job->root = root + i;
12         pthread_mutex_lock(&lock_of_job);
13         new_job->next = job_list;
14         job_list = new_job;
15         pthread_mutex_unlock(&lock_of_job);
16         pthread_cond_broadcast(&cond_of_job);
17     }
// Given job of decision data
 1     int vote_result[num_test_data];
 2     int vote_num[num_test_data];
 3     for (int i = 0;i < num_test_data;++i){
 4         vote_result[i] = 0;
 5         vote_num[i] = 0;
 6         for (int j = 0;j < num_tree;++j){
 7             pthread_mutex_lock(&lock_of_malloc);
 8             Job* new_job = (Job*)malloc(sizeof(Job));
 9             pthread_mutex_unlock(&lock_of_malloc);
10
11             new_job->type = 2;
12             new_job->index = vote_num + i;
13             new_job->result = vote_result + i;
14             new_job->test_data = test_dataset + i;
15             new_job->root = root[j];
16             pthread_mutex_lock(&lock_of_job);
17             new_job->next = job_list;
18             job_list = new_job;
19             pthread_mutex_unlock(&lock_of_job);
20             pthread_cond_broadcast(&cond_of_job);
21         }
22     }
// Terminal all thread
 1     for (int i = 0;i < num_thread;++i){
 2         pthread_mutex_lock(&lock_of_malloc);
 3         Job* new_job = (Job*)malloc(sizeof(Job));
 4         pthread_mutex_unlock(&lock_of_malloc);
 5
 6         new_job->type = 0;
 7         pthread_mutex_lock(&lock_of_job);
 8         new_job->next = job_list;
 9         job_list = new_job;
10         pthread_mutex_unlock(&lock_of_job);
11         pthread_cond_broadcast(&cond_of_job);
12     }
```

thread 接到job後，會依據不同的類型執行不同的命令。(268行)

2 試畫出或以表格做出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

num_tree定為20，在工作站2上測試的結果。

num_thread	time	num_thread	time	num_thread	time
		11	1m30.156s	21	1m19.044s
2	9m11.772s	12	1m30.814s	22	1m16.123s
3	4m25.982s	13	1m28.126s	23	1m15.301s
4	3m01.775s	14	1m30.701s	24	1m17.365s
5	2m39.867s	15	1m28.696s	25	1m17.755s
6	1m57.895s	16	1m16.694s	26	1m12.734s
7	2m05.809s	17	1m22.646s	27	1m21.660s
8	1m49.919s	18	1m20.689s	28	1m11.755s
9	1m41.921s	19	1m21.111s	29	1m21.771s
10	1m41.892s	20	1m19.009s	30	1m15.604s

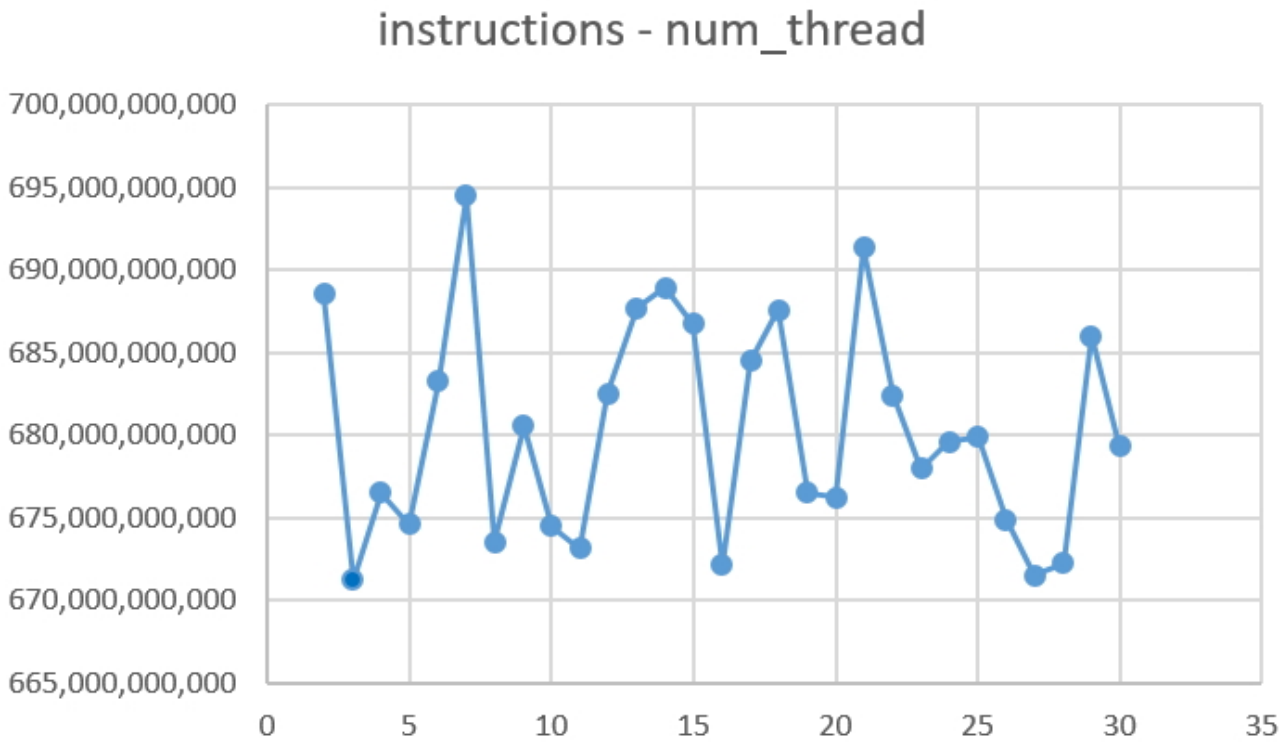


最小值發生在thread_num為28時。每個thread要建的樹的數量大致為$20/num_thread$，所以時間和num_thread成反比，但當$num_thread > 20$時，每個thread並沒有辦法建分數棵樹，而多出來的就閒置在那哩，導致$thread_num > 20$以後時間基本上差不多，只有一些隨機產生的起伏。

3 試畫出或以表格做出thread數量與instruction的數量的比較，並說明此圖表

num_tree定為20，在工作站2上測試的結果。

num_thread	instructions	num_thread	instructions	num_thread	instructions
		11	673,110,630,233	21	691,363,116,684
2	688,555,341,213	12	682,492,185,798	22	682,398,252,828
3	671,142,211,911	13	687,701,225,826	23	677,948,216,440
4	676,492,369,267	14	688,943,500,812	24	679,603,983,904
5	674,610,037,106	15	686,754,824,918	25	679,886,080,457
6	683,235,674,526	16	672,098,743,135	26	674,900,131,662
7	694,482,578,782	17	684,500,465,174	27	671,511,403,496
8	673,501,294,414	18	687,574,989,502	28	672,227,000,131
9	680,548,478,663	19	676,482,485,662	29	685,974,505,232
10	674,476,281,980	20	676,245,601,717	30	679,324,352,286

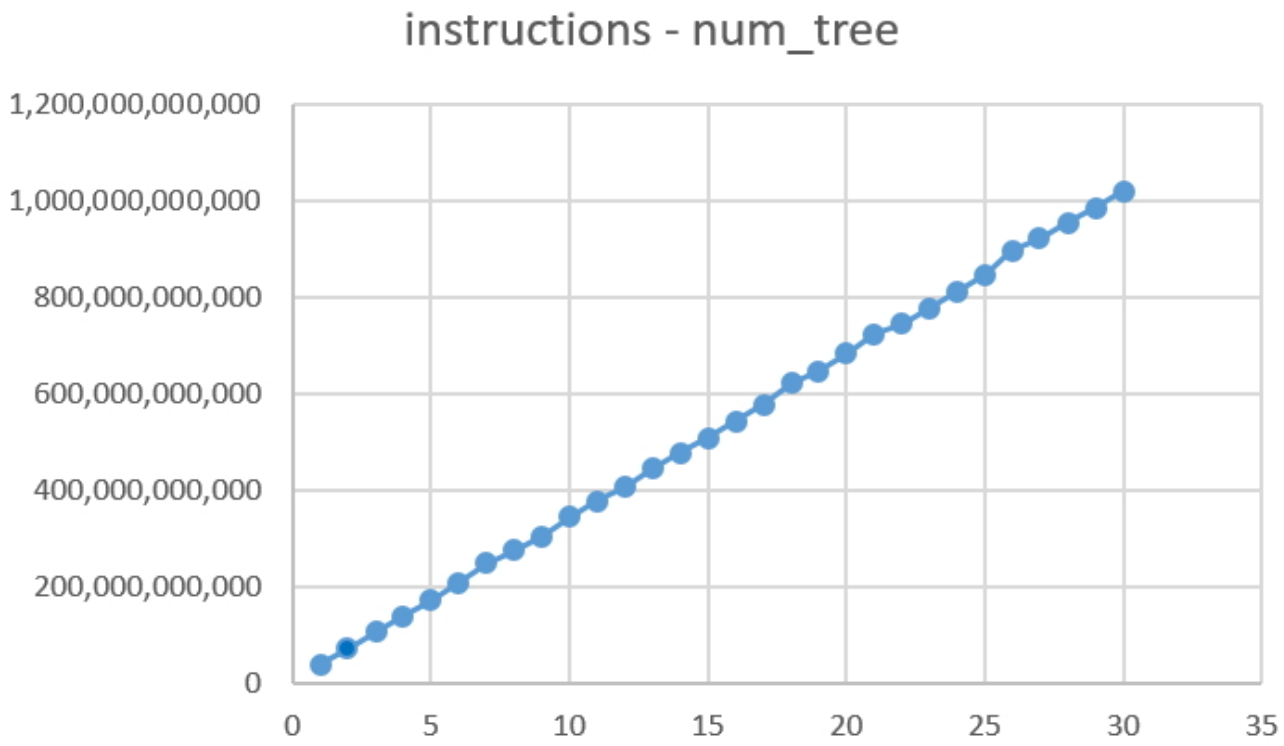


instructions的數量大致為680,000,000,000。每次建立模型所要建立的樹皆為20棵，每棵樹雖然因為隨機選取資料運算量會不盡相同，但因為選取的資料量是一樣的所以大致上不會差異過大。因此總共的instructions數量基本為定值，只有一些小幅的起伏。

4 試畫出或以表格做出樹的數量與instruction的數量的比較，以紅色標出時間最快的位置，並說明此圖表

num_thread定為20，在工作站2上測試的結果。

num_tree	instructions	num_tree	instructions	num_tree	instructions
1	37,044,388,474	11	376,751,714,004	21	724,376,950,000
2	67,863,362,805	12	406,166,530,962	22	744,542,441,612
3	105,317,222,021	13	443,879,354,339	23	777,085,082,329
4	138,179,459,001	14	476,527,377,015	24	811,696,267,168
5	171,339,931,936	15	508,033,614,269	25	847,612,535,879
6	206,656,682,721	16	542,715,206,288	26	897,386,800,932
7	248,095,634,170	17	577,382,022,438	27	920,134,060,047
8	275,119,894,515	18	622,268,480,218	28	955,270,852,685
9	304,589,582,242	19	647,646,549,586	29	985,884,547,910
10	343,430,998,314	20	684,267,298,940	30	1,020,848,985,368



每棵樹雖然因為隨機選取資料運算量會不盡相同，但因為選取的資料量是一樣的所以大致上不會差異過大。因此總共的instructions數量和樹的樹量呈線性關係，只有一些小幅的起伏。另外可以從斜率推知，建一棵樹所需要的成本約為34,000,000,000個instructions。

5 說說你的其他發現

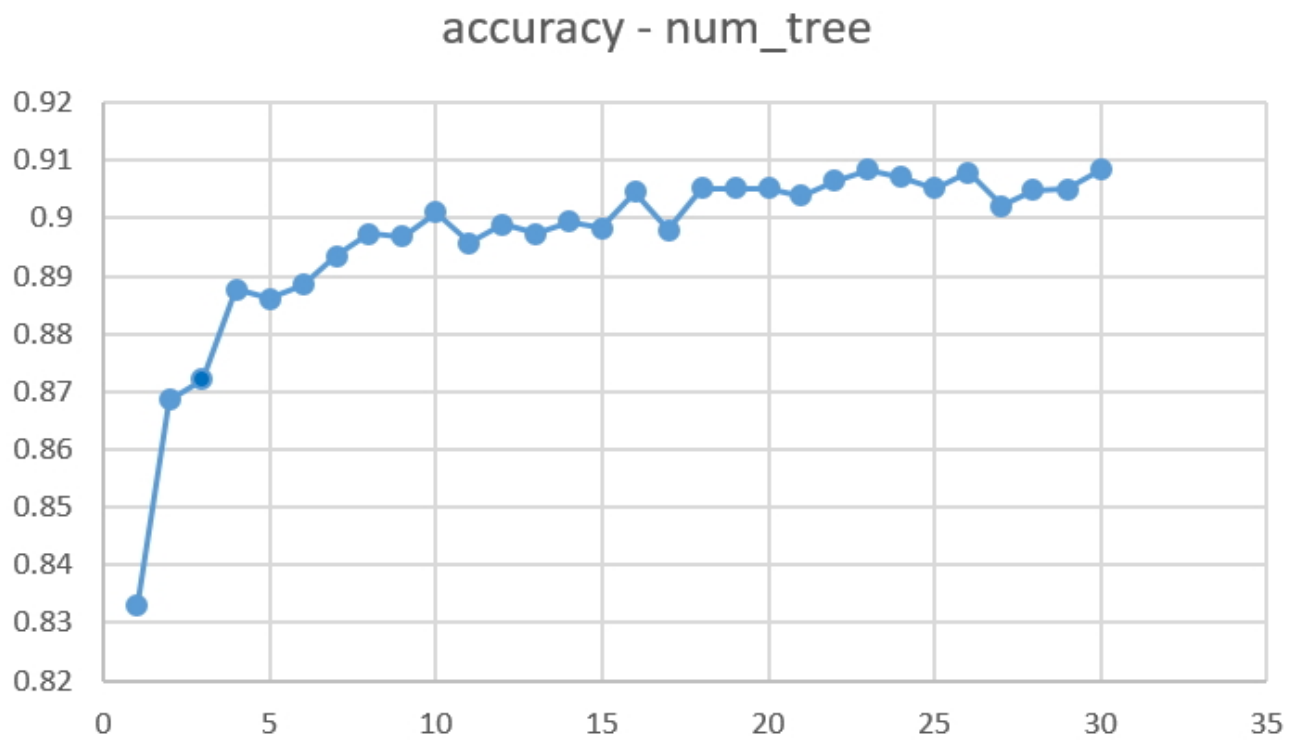
5.1 num_thread不影響正確率

thread的數量只是改變同時進行的運算數目，不影響總共執行的運算量，不影響正確率符合預期。

5.2 num_tree對正確率的影響

num_thread定為20，在工作站2上測試的結果。

num_tree	accuracy	num_tree	accuracy	num_tree	accuracy
1	0.833093	11	0.895753	21	0.903911
2	0.868562	12	0.898912	22	0.906510
3	0.872161	13	0.897313	23	0.908349
4	0.887676	14	0.899472	24	0.907070
5	0.886116	15	0.898353	25	0.905270
6	0.888556	16	0.904511	26	0.907869
7	0.893434	17	0.897993	27	0.902151
8	0.897273	18	0.905190	28	0.904950
9	0.896913	19	0.905270	29	0.904990
10	0.901112	20	0.905270	30	0.908429



當只有兩棵樹的時候準確率就已經超過要求的85%，剛開始樹的數量增加準確率上升，但很快

就接近飽和，在90%附近幾乎不成長，因此最佳的num_tree落在10到20之間，在往上增加效益有限。