Computer Organization Final Project Report

0710764 黃聖偉

	system.cpu.workload.profile_totaltime search 總執行時間	profile_totMemAccLat_wo_overlap Memory time	Compute time	Compute time ÷ memory time
Number of ticks Original	130680000	73522749	57157251	0.777
Number of ticks Modified	121743500	66158578	55584922	0.840

	Final tick	system.cpu.dcache.overall_miss_rate::total
Original	4257980000	0.688914
Modified	4089685500	0.683599

- system.cpu.workload.profile_totaltime 是 total function time。
- profile_totMemAccLat_wo_overlap 是 memory access time。
- compute time = total function time - memory access time。

根據模擬的結果，不論是原本的程式或是優化後的，其模擬的 Compute time 皆略小於 memory access time，compute time / memory time 約等於 0.8，所以判斷 string matching with suffix array 的 bottleneck 為 memory access。

另外 binary search 的 time complexity 是 O(log n)，基本上已經是搜尋的極限不會再更快了，能夠減少 compute time 的方法只有想辦法減少程式 constant。

而能夠大幅加快這次程式速度的方法應該是想辦法加快 bottleneck 也就是 memory access，我的作法是減少 memory access 的次數，但這個方法也有限制，畢竟演算法上每個迴圈都需要存取 memory 不同位置，也很難再減少 memory access 次數。

如何改善程式

Original

```
12  void search(char *pat, char *txt, int *suffArr, int n)
13  {
14
15      // Do simple binary search for the pat in txt using the
16      // built suffix array
17      int l = 0, r = n-1;  // Initialize left and right indexes
18      // Find the top index
19      while (l != r)
20      {
21          int mid = (l + r) / 2;
22          int res = strncmp(pat, txt+suffArr[mid], strlen(txt+suffArr[mid]));
23          if (res < 0) r = mid;
24          else l = mid + 1;
25      }
26      int top = l;
27
28      // Find the bottom index
29      l = 0, r = n-1;
30      while (l != r)
31      {
32          int mid = (l + r + 1) / 2;
33          int res = strncmp(pat, txt+suffArr[mid], strlen(pat));
34          if (res == 0 || res > 0) l = mid;
35          else r = mid - 1;
36      }
37      int down = l;
38
39      for(int i = top; i <= down; i++){
40          cout << "pattern matched at pos " << suffArr[i] << "\n";
41      }
42  }
```

Modified

```
12  void search(char *pat, char *txt, int *suffArr, int n)
13  {
14
15      // Do simple binary search for the pat in txt using the
16      // built suffix array
17      int l = 0, r = n-1;  // Initialize left and right indexes
18      // Find the top index
19      while (l != r)
20      {
21          int mid = (l + r) / 2;
22          const char * cmpStr = txt+suffArr[mid];
23          int res = strncmp(pat, cmpStr, strlen(cmpStr));
24          if (res < 0) r = mid;
25          else l = mid + 1;
26      }
27      int top = l;
28
29      // Find the bottom index
30      l = 0, r = n-1;
31      while (l != r)
32      {
33          int mid = (l + r + 1) / 2;
34          int res = strncmp(pat, txt+suffArr[mid], strlen(pat));
35          if (res == 0 || res > 0) l = mid;
36          else r = mid - 1;
37      }
38      int down = l;
39
40      for(int i = top; i <= down; i++){
41          cout << "pattern matched at pos " << suffArr[i] << "\n";
42      }
43  }
```

由於 binary search 演算法看起來已經沒甚麼進步空間，因此我們只從一些程式碼細節做修改。

從以上比較可以看出，在紅框的部分，我們將原本 txt+suffArr[mid]存成一個新的字串常數 cmpStr，做出這樣改變的主要原因是在原本程式中每一次 while 迴圈 txt+suffArr[mid]被使用了 2 次，這樣 suffArr[mid]就需要被存取 2 次，txt 也被存取了 2 次，cache miss 的機率會變高，也需要花更多時間存取記憶體與加法運算。因此我改動程式後每一次 while 迴圈 txt 與 suffArr[mid]存取次數改為一次，將存取結果運算後存入變數 cmpStr，再將 cmpStr 傳入 strncmp 運算。

結果看起來 cache miss rate 從 0.688914 降至 0.683599，Final tick 由 4257980000 降至 4089685500。程式速度有變快，cache miss rate 也有降低，有成功改善程式。