**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 | Modified |
|  |  |

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。