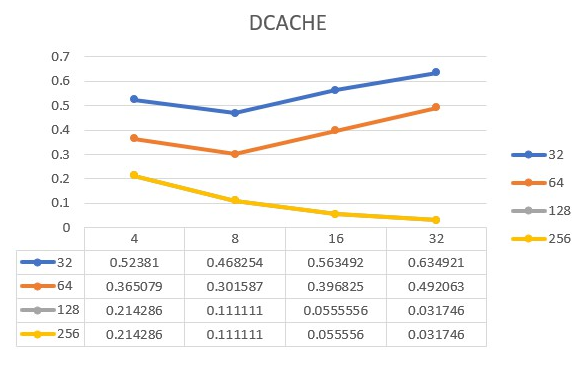
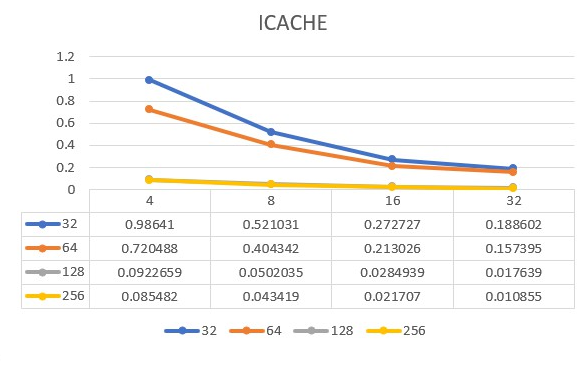
HW4 Report

姓名學號: 林哲宇 0616018, 張哲銓 0616032



當 block size 增大，會因為 spatial locality 的特性降低 miss rate，因此可以解釋上圖折線下降；然而，如果在同個 cache size，一直提高 block size，則會導致 pollution，提高 miss penalty，因此可以解釋上圖折線圖上升。



當 block size 增大，會因為 spatial locality 的特性降低 miss rate，因此可以解釋上圖折線下降；然而，如果在同個 cache size，一直提高 block size，則會導致 pollution，但是圖中並沒有發現上升的情況，因此可以合理推測block size的範圍還沒大到足以產生pollution。

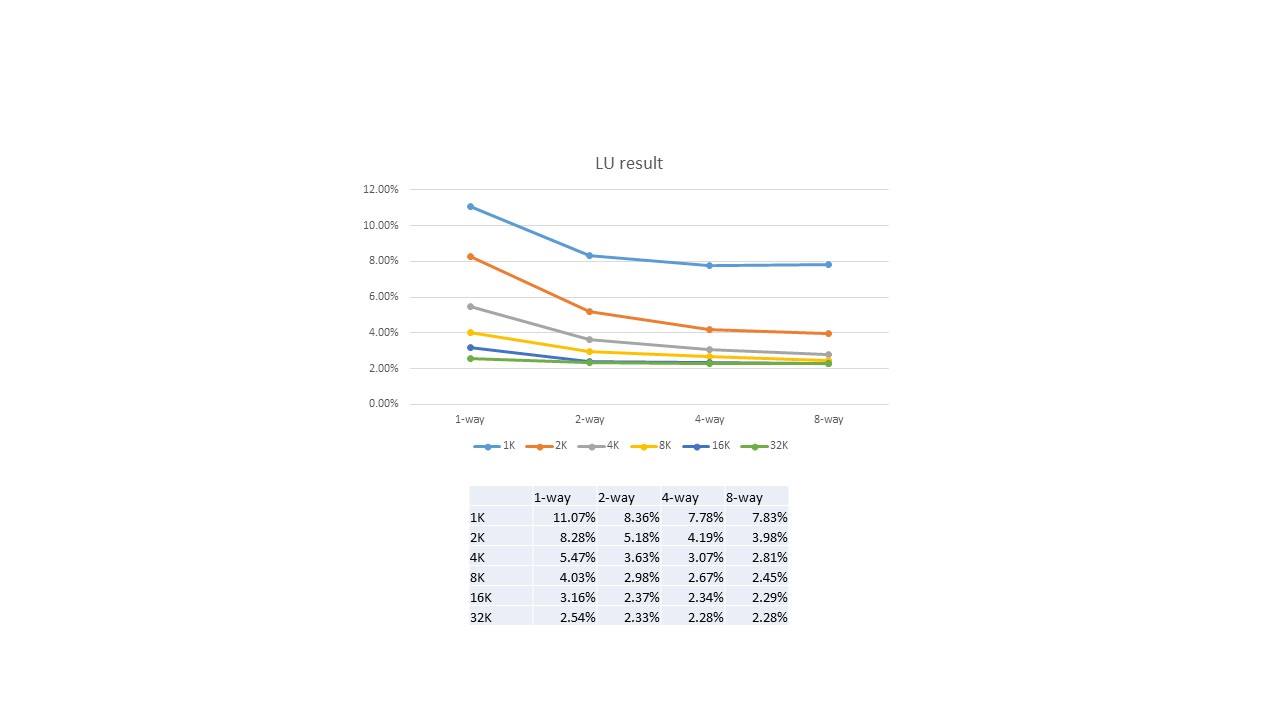
**BASIC結論**

程式的部分我們把一些 module 和 entry 名稱能對應到 testbench。Cpp檔我們把就計算總次數和 miss 的次數最後相除結果就是 miss rate。

關於 ICACHE 和 DCACHE的 cache size 和 block size，我們觀察 Verilog 執行後生成的 ICACHE.txt 和 DCACHE.txt 之後，發現只有後兩個 bytes 有被用到，所以我們把 cache size 調成 32, 64, 128, 256，並且把 block size 調成 4, 8, 16, 32，單位為 B。

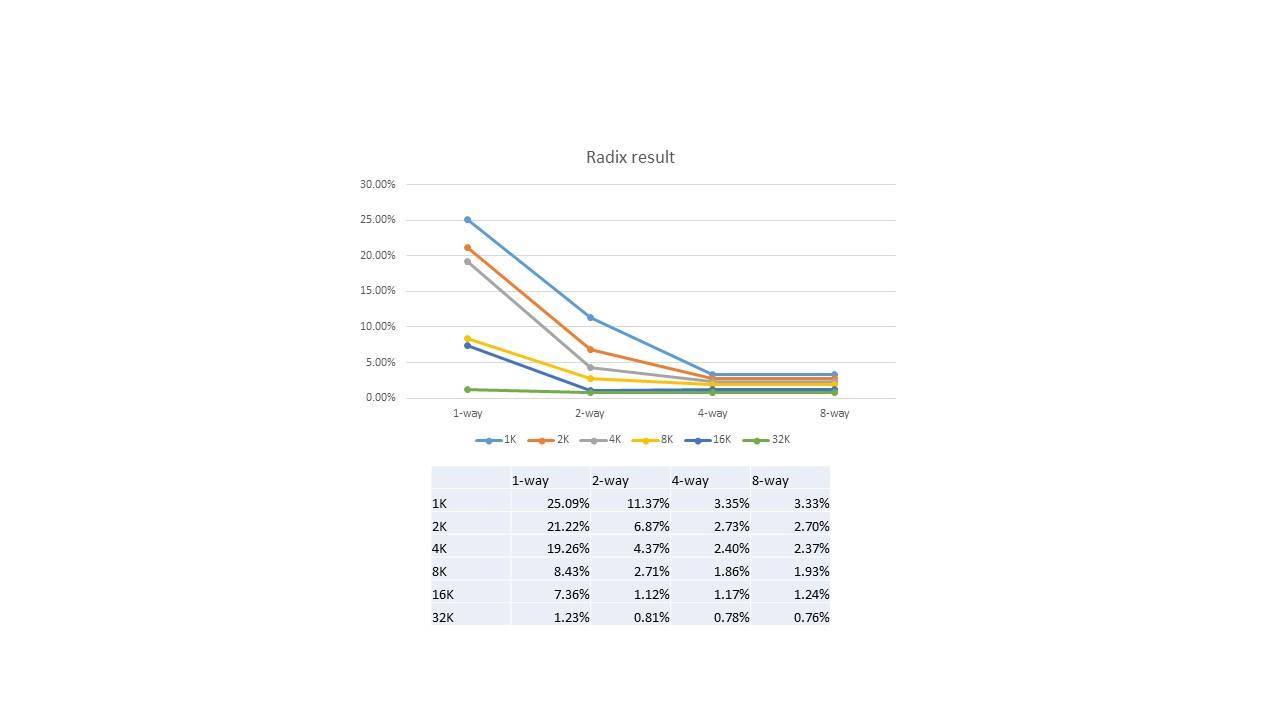
至於為什麼ICACHE是持續下降而DCACHE卻有上升也有下降，我們認為是因為ICACHE.txt的spaital locality的範圍較大，因此在block size持續增大的過程，仍然不會導致pollution，使miss penalty提高。

LU:



從上圖可以看到在cache size和 n越大，miss rate就越小。至於為什麼LU和RADIX跑出來的值有那樣的不同，會在下面做說明。

RADIX:



從上圖可以看到在cache size和 n越大，miss rate就越小。至於為什麼LU和RADIX跑出來的值有那樣的不同，會在下面做說明。

**ADVANCED 結論**

Advanced 的部分加了 n-way set-associative，所以在程式中，需要 struct 中的 tag 變成陣列，儲存每個 record。一開始先把每個 tag 初始化成 -1，每次判斷是否為 hit，就掃過去 tag 陣列，如果是 hit，就把對應的 tag 移到陣列的第零項，之前的就依照原本順序往後順移一個；如果是 miss，就把最後一項改成新的 tag，並把它移到第零項，之前的就依照原本順序往後順移一個。這樣就可以順便解決每一筆 record 最近被用過的順序了。

分析的部分，由上圖可以很明顯發現每條線隨著 n 越大，miss rate 就越小，這是因為 block size 都固定，但是隨著 n 越大，能存的 tag 數越多，miss rate 自然就下降。Cache size 上升也會讓 miss rate 下降是因為 line 上升，能存的 index 也越多。

關於RADIX和LU的比較，我們討論為什麼在1-way的情況下，cache size為1k和32k時一個RADIX的miss rate比較大，另一個比較小。可能的原因是當cache size變大，block size不變的情況下，line會變多，所以如果RADIX的前幾bits沒什麼變動，tag就不會換，也就不會有miss發生。

我們也討論了為什麼在同個cache size的情況下，可能n=1時LU比較小，n=8時RADIX卻比較大。我們認為是因為RADIX的spatial locality的範圍較小，所以重複取道同個tag的次數也比較多，因此hit數也相對高。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1-way | 2-way | 4-way | 8-way |
| 1K | 8560 | 8576 | 8592 | 8608 |
| 2K | 17088 | 17120 | 17152 | 17184 |
| 4K | 34112 | 34176 | 34240 | 34304 |
| 8K | 68096 | 68224 | 68352 | 68480 |
| 16K | 135936 | 136192 | 136448 | 136704 |
| 32K | 271360 | 271872 | 272384 | 272896 |

total bits =

line \* (block size \*8 + 32 – log2(line) + log2(n) – log2(block size) +1)