# HW3 REPORT

謝翔丞, 109550025

***Abstract*—本篇旨在分析掛載 8 KB cache 後之 Coremark 執行效能，透過過程中 analysis 狀態 cycle 數量測執行 latency 以及 cache 之 hit/miss 情形同時加以不同 way 數量進行試驗，發現 8 KB 條件下 miss rate 極低，且不同 way 數並無帶來明顯成效; 同時嘗試使用不同演算法以其優化執行效能，此篇使用 LRU 算法，但效果非但不如預期，反而降低 hit rate 與執行效能。 (*Abstract*)**

***Keywords—Cache; FIFO; LRU; Hit/Miss rate; Latency (key words)***

## I. INTRODUCTION

此次分為兩部分，前段建立在原有之 aquila SoC，加掛 8KB cache 取代 TCM 分析其執行效能，同時調整各種 way 數量進行測試; 後段著重更改 cache replacement policy，嘗試以不同演算法優化 Coremark 表現。

## II. 8KB CACHE ANALYSIS

### *A. 實驗方向與猜想*

下列將分別針對 2way, 4way 以及 8way，進行 Cache hit/miss、Read/Write hit/miss 和 Latency Counting，由於本次 Cache 定在 8 KB，可以預期的是即使不同 way 數量間數據有所差異，差距仍然會比 2 KB 來的較為不明顯。

### *B. 結果分析*

*1) Cache Hit / Miss*

首先，我先針對 Hit / Miss rate 部分著手，由表格 1 可見，hit rate 相當高。同時，在 1100 次 Iteration 之下，三者的數據呈現幾乎相同，唯 2 way 的 miss 數量略高一些，但是以整體 cycle 數量來看仍在可忽略的範圍內。

| Cycle Count | 2way | 4way | 8way |
|---|---|---|---|
| **Cache Hit** | 87675177 | 87675241 | 87675242 |
| **Cache Miss** | 5673 | 5609 | 5608 |

表格 1.Cache Hit/Miss Count with Different Way Nums

*2) Read / Write Hit / Miss*

接著我針對 Read / Write 部分進行 Hit / Miss rate 的實驗。我們知道要成就一個好的 Cache 設計，關鍵在於 Hit Rate 是否夠高。由表格 2 呈現，綜合比較 Read 以及 Miss rate， 4 way 在數據上呈現較為優異的表現，為三者之冠。

| Cycle Count | 2way | 4way | 8way |
|---|---|---|---|
| **Read Hit** | **87637113** | **87654622** | **87637687** |
| **Read Miss** | **3936** | **3165** | **3356** |
| **Write Hit** | **19563540** | **19563561** | **19563561** |
| **Write Miss** | **4069** | **4048** | **4048** |

表格 2.Read/Write Hit/Miss to Different Way Nums

*3) Average Cache Latency*

觀察 D-Cache Controller 的 FSM，我們不難看出整個執行過程的 Critical Part 在 Analysis 及其之後的階段; 如果成功 Cache Hit，那麼就會回到 IDLE，成就一次完美的結果；但若發生 Cache Miss，則到 Rd_from_Mem_finish 之間又能拆分為兩大階段; 其一為 Dirty 的情形，也就是需要將資料 write 回 memory，同時必定伴隨著 cycle 數的增加，因為需要額外花費時間將資料寫回 memory；而另一則是不需要經過 Wb_to_mem 和 Wb_to_mem_finish 這兩個 state，只需經過 read 的階段即可。針對這兩部分，我透過測量 p_strobe_i 與 p_ready_o 之間的 cycle 來計算 latency，同時我也測量前述 write back to memory 與否的情形下，所花費的 cycle 數差異。

*a) Latency (between p_strobe_i & p_ready_o)*

| Cycle Count | 2way | 4way | 8way |
|---|---|---|---|
| **Latency** | 87680849 | 87680849 | 87680849 |

表格 3.Latency to Different Way Nums

*b) With / Without Write back to memory*

表格 4 是我測量從進入 Analysis state 到 Rd_from_mem_Finish，期間經過 Data 寫回 Memory 與否所花費的時間差異，可以發現，2 Way Cache 在碰到 miss 後將資料寫回記憶體這項動作上相較其他 Way

Number 花費最多的時間，這邊也恰好呼應到表格 1 所呈現，2 Way Cache 的 miss rate 位居三種 way number 之冠。

| Cycle Count | 2way | 4way | 8way |
|---|---|---|---|
| Rd_Wb_Latency | 87948171 | 87944786 | 87944850 |
| Rd_Without_Wb_Latency | 87853817 | 87851686 | 87851721 |
| Difference | 94354 | 93100 | 93129 |

表格 4.Write to mem / not to Different Way Nums

## III. Other Cache Replacement Policy : LRU

關於 Cache Replacement Policy 有許多演算法可以實作，這裡我使用 LRU，Least Recently Use; 這是一個相對不難實作的算法只要額外支出些許空間紀錄每個儲存位址的順序，每次需要重新寫入 cache 時就選擇最早被寫入的位址，因此實作時只要將被取代的位址挪到最後，並將其以後的資料都往前移， 就可以完成最基本的 LRU 實作。見 Figure 1.

```
case ( { way_hit[0], way_hit[1], way_hit[2], way_hit[3] } )
        4'b1000: begin
            LRU_cnt[line_index][0] <= LRU_cnt[line_index][1];
            LRU_cnt[line_index][1] <= LRU_cnt[line_index][2];
            LRU_cnt[line_index][2] <= LRU_cnt[line_index][3];
            LRU_cnt[line_index][3] <= LRU_cnt[line_index][0];
        end
```

Figure 1. LRU Example

### A. *結果分析*

*1) Iteration / Sec*

表格 5 是我針對 FIFO 以及取代之的 LRU 進行 Iteration/Sec 的比較，發現我實作的 LRU 在效能上遠遠不及 FIFO，原先 FIFO 在不同 way number 下效能呈現是差不多的狀態，然而在更換成 LRU 之後，不但表現最好的 2way 相較於 FIFO 都更為糟糕，甚至隨著 way number 上升有分數逐漸下滑的趨勢。

| Iteration/Sec | 2way | 4way | 8way |
|---|---|---|---|
| FIFO | 70.7981 | 70.7982 | 70.7981 |
| LRU | 70.7082 | 67.7349 | 66.7570 |

表格 5. Iteration/Sec to FIFO / LRU

*2) Cache Hit / Miss*

由 Figure 2 以及 Figure 3 可以發現，FIFO 在不同 way number 下，Miss cycle 以及 Miss Rate 都與 Iteration 相似呈現持平的表現，反觀我實作的 LRU 在 Miss Rate( Cycle )上比之 FIFO 相當高，是數十甚至百倍的差距並且急遽上升，關於數據如此呈現的原因後續會進行討論。

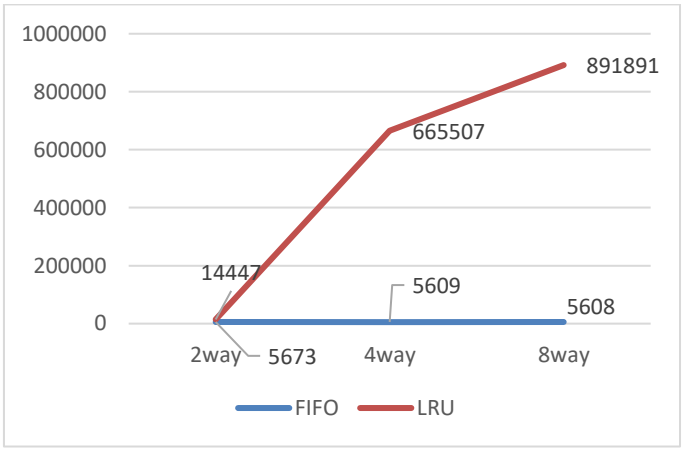


Figure 2. Miss Cycle between FIFO / LRU

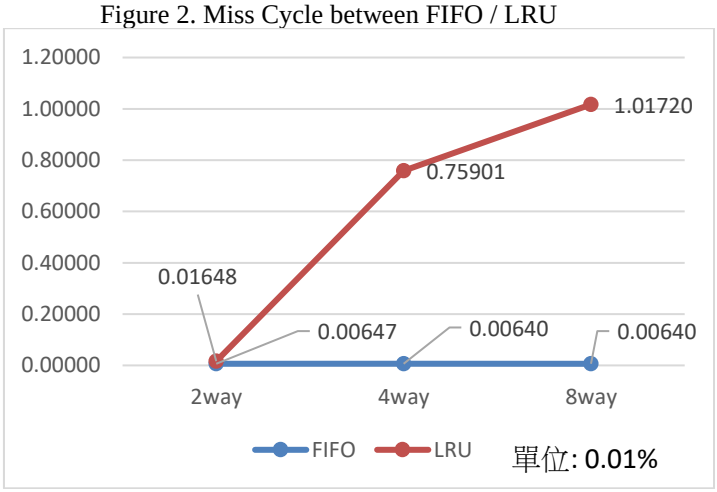


Figure 3.Miss Rate between FIFO / LRU

*3) Average Cache Latency ( between p_strobe_i & p_ready_o )*

| Cycle Count | 2way | 4way | 8way |
|---|---|---|---|
| Latency | 88368676 | 87944941 | 131711739 |

表格 6Latency to Different Way Nums (LRU)

### B. *討論*

關於為何 LRU 在這裡會有較差的表現，我個人有以下想法:

我此次實作 LRU 的方式因為省略了 hash 的過程並且只使用最簡單的將最新被替換的位址移動到序列最後，所以整體時間來說應是近似於 FIFO 的實作。

既然排除掉時間因素，那麼最大的可能就是 Coremark 執行的過程恰好對 LRU 的排程算法不利，導致不斷的形成 Cache Miss 進而致使要花費更多 cycle 做 memory 讀寫，因此造成 Iteration/Sec 的大幅下降。