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)

# 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。

# 8KB cache Analysis

## 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。

## 結果分析

### 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

### 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

### 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。

#### 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

#### 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

# 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.

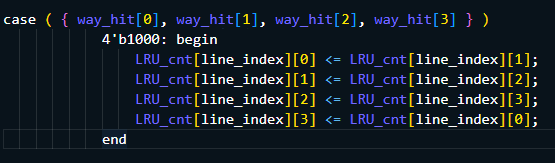


Figure . LRU Example

## 結果分析

### 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

### 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 . Miss Cycle between FIFO / LRU

Figure .Miss Rate between FIFO / LRU

### 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)

## 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。