RISC_V Final Report

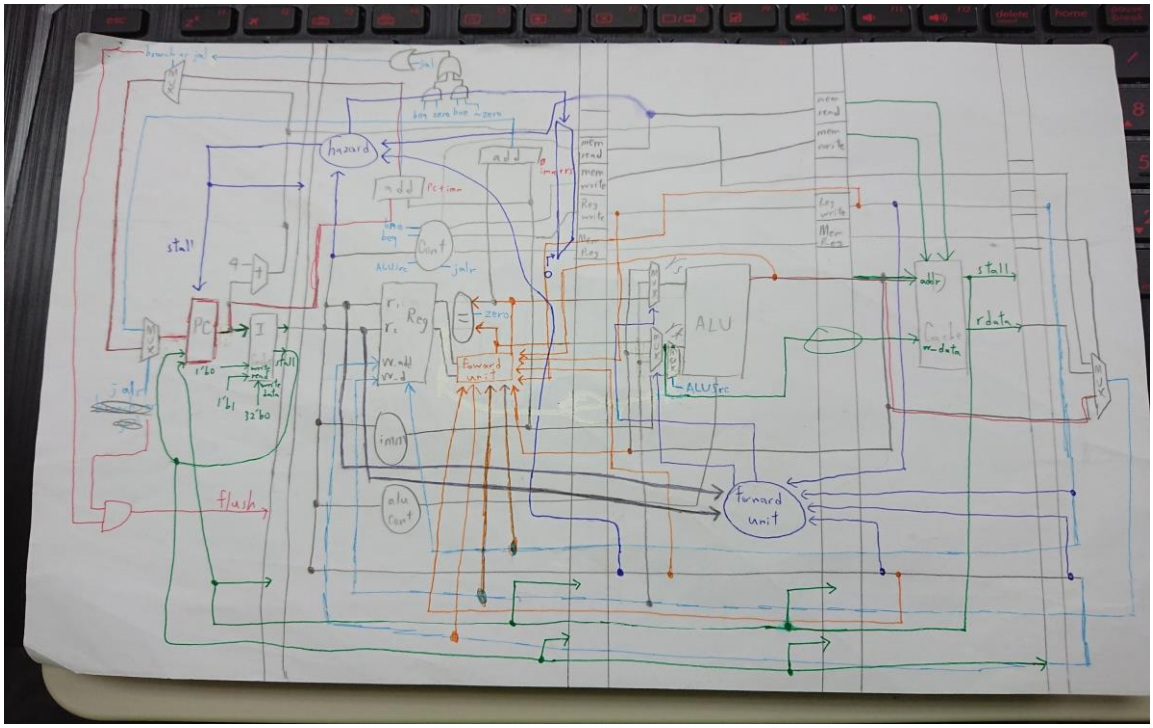
R2 B06901176 趙彥安

B06901177 劉凡

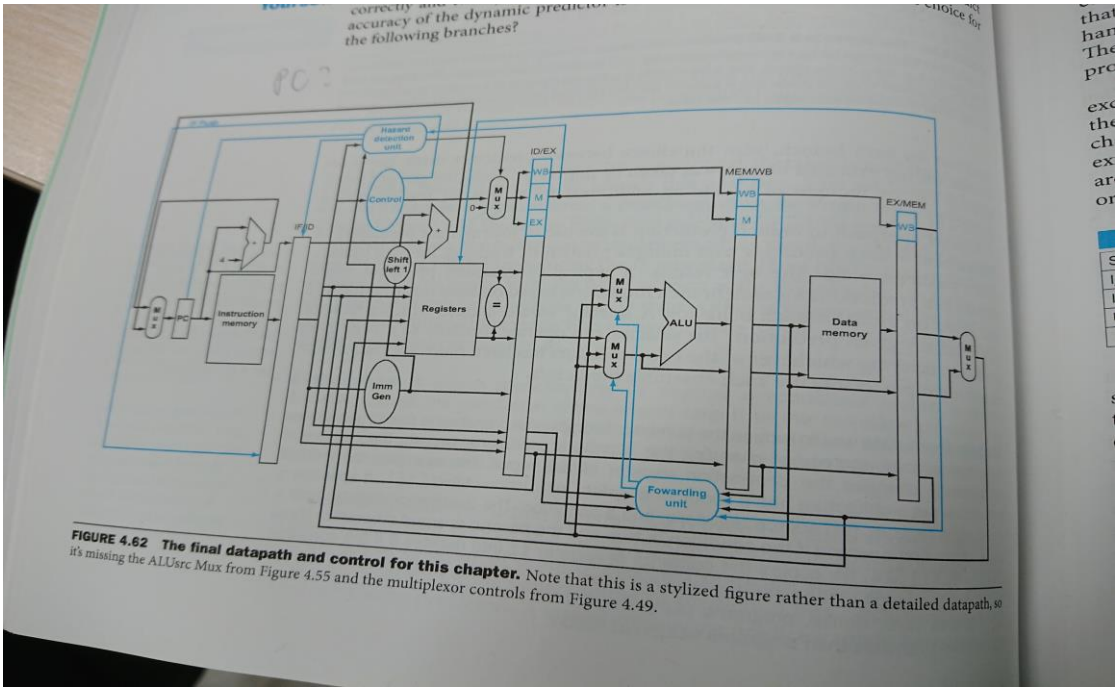
B06901178 蔡易霖

❖Baseline

以下為自己畫的 pipeline 架構圖:



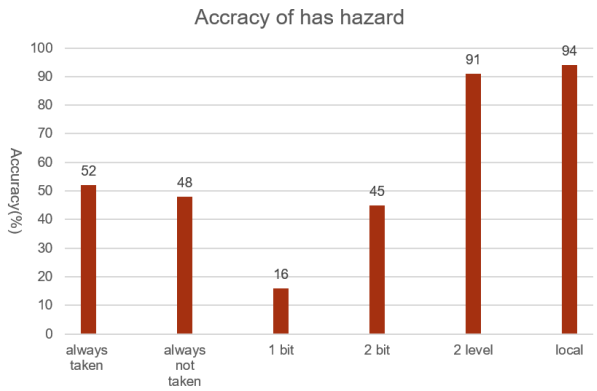
和課本做比較:

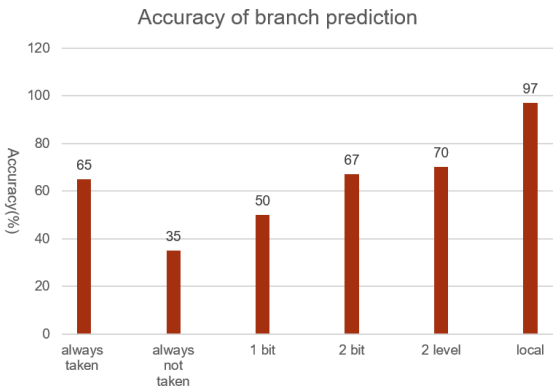


- ❑ 可以發現我們將 ALU_control 移到 ID_stage，因為它是以instruction 控制，我不希望還要再將 instruction 牽線到 EX_stage，而且它也不會對 ID_stage 有影響
- ❑ 為了測試 beq 和 bne，需要在 ID_stage 增加 forward_unit避免錯誤
- ❑ 我們將 ID_stage 的 ALU_Src 判斷移到 EX_stage 避免完成 forward 後 imm 被洗掉

❖ Branch prediction(註:我繳交的CHIP.v是2 level adaptive predictor)

Comparison about accuracy between each predictor





- Design methodology for good score (before/after) and detailed discussion:
  - before 即為always not taken performance
  - Branch predictor的設計原理是我的prediciton unit先在IF stage output我的預測，然後在ID stage 傳給prediction unit預測是否正確，如果是dynamic prediction的話prediction unit會根據正確與否修正自己的預測(例如改變saturating counter的state)，進而使下次預測更加準確，另外，如果預測錯誤的話ID stage會flush並且傳入正確的pc值，以補救預測錯誤造成的影響。
  - 2 level apdative predictor的原理是把該程式前幾次branch記錄存起來，實作上我們存兩個bit，每個不同的歷史紀錄都有自己的saturating counter，因此我們總共有2^2=4個saturating counter，我們每次做預測實會根據當前的歷史紀錄選擇對應的saturating counter，再根據該saturating counter所處的state做出預測。

- Local predictor 的原理是把每一個instruction的branch記錄存起來，我們實作上是存前兩個bit，而每一個instruction都有4個saturating counter，做預測時我們會先根據我們在哪一個instruction，再根據該Instruction目前歷史紀錄為何，選擇相應的saturating counter並根據棋目前處於的state 做出預測。
- 從上圖可知local predictor正確率最高，2 level adaptive predictor次之，但二者都超過90%(除了branch prediction的測資，下面會說明原因)，可知只要我們預測時，掌握越多資訊(如global 和local歷史紀錄)就能做出越準確的預測，但要存取越多資訊就要犧牲越多面積，因此這是一種area和accuracy(也是execution time)的trade off。
- I_mem_BrPred測資之所以會讓global history的predictor正確率只有70%的原因是因為interleaving時BRANCH 情形是 T,NT,……T,NT,…這個測資，也就是有連續的branch指令，這樣在地一個branch結果由ID stage傳給branch prediction unit時，下一個branch instruction在IF stage就需要預測了，但是這時結果還沒存入history register，導致第二個prediciton是根據錯誤的history(也就是T,NT，但正確應該時NT,T)，另外當ID stage傳入正確的結果時，這時history register已經更新為NT,T，導致01這個saturating counter需要為不是

自己預測的結果被修正，此二點導致預測正確率下降。

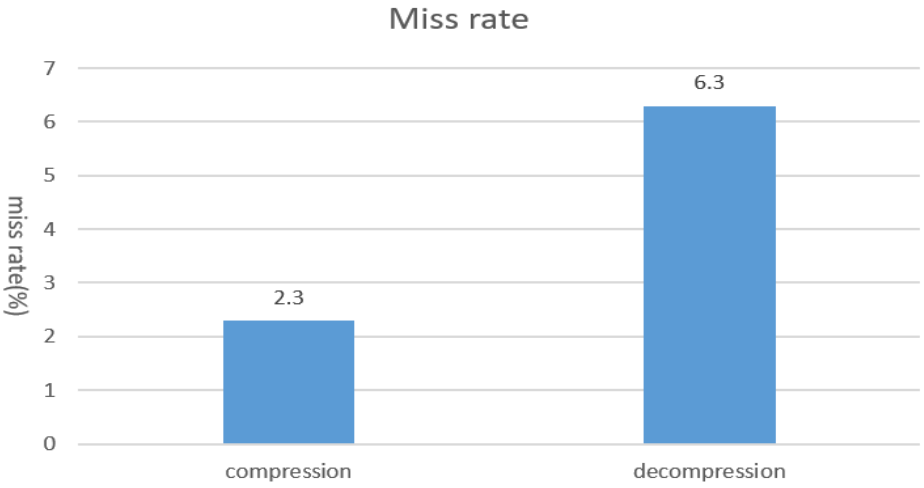
- ❖ The relationship between design BPU and parameter size for generating test program:
  - ➢ 以I_mem_BrPred為例:always not taken 的正確率就會受到a和b/2影響，always taken會受到b/2和c值影響，1 bit是當T,NT,T,NT出現時正確率會很差，

  2 bit是當T,T,NT,NT,T,T,NT,NT出現時正確率會很差，至於2 bit adaptive counter 因為在Interleave會因為上述問題正確率受b值影響，至於local則不受a,b,c影響。
- ❖ What you have learned?
  - ➢ 我覺得學到最多的是自學和實作出來的能力，2 bit adaptive predictor和local predictor都是上課沒教的，因此我要利用網路上資訊實作出一個全新的predictor，並且有bug時也要自己找資料解決，這段經驗令我受益良多。

❖Compression

- ➢ What is the advantage of the C extension? Verify it with your simulation results.
  - ■ Less miss rate: 由於壓縮後的指令都只有16bit，所以offset只有2 byte，因此從I cache fetch instruction的過程中，相較於32 bit instruction，pc比較不會超出cache size，因此有較低的miss rate，較低miss rate可以減少simulation time 以及消耗功率(access main memory會

消耗功率)，下圖是I_mem_compression這個測資中，compression 和decompression I cache miss rate 比較



- Less simulation time:
- 這點也和miss rate有很大的，由於miss rate低因此可以省下access main memory的時間，具體compression 和decompression時間如下(under clock cycle=10ns):
- compression:

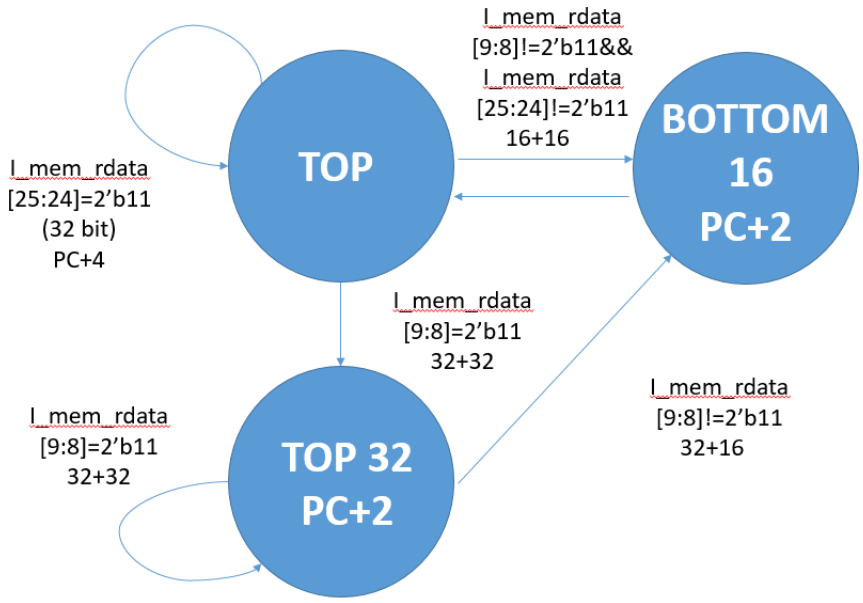
```
-------------------------------------------------

TART!!! Simulation Start .....

-------------------------------------------------

SDB Dumper for IUS, Release Verdi_N-2017.12, Linux, 11/12/2017
C) 1996 - 2017 by Synopsys, Inc.
Verdi* FSDB WARNING: The FSDB file already exists. Overwriting the FSDB file may crash the p
Verdi* : Create FSDB file 'Final.fsdb'
Verdi* : Begin traversing the scope (Final_tb), layer (0).
Verdi* : Enable +mda dumping.
Verdi* : End of traversing.
Verdi* : Begin traversing the scopes, layer (0).
Verdi* : End of traversing.
------------------------- Simulation FINISH !!-------------------------
================================================================================

\(^o^)/ CONGRATULATIONS!!  The simulation result is PASS!!!

================================================================================
imulation complete via $finish(1) at time 3735 NS + 0
/Final_tb.v:158              #(`CYCLE) $finish;
csim> exit
b06178@cad29 generate]$
```

- decompression:

```
               Initial blocks:           1       1
               Cont. assignments:       27      83
               Pseudo assignments:       6       6
               Simulation timescale:  10ps
       Writing initial simulation snapshot: worklib.Final_tb:v
Loading snapshot worklib.Final_tb:v ..................... Done
*Verdi* Loading libsscore_ius152.so
ncsim> source /usr/cad/cadence/INCISIV/cur/tools/inca/files/ncsimrc
ncsim> run
-----------------------------------------------------

START!!! Simulation Start .....

-----------------------------------------------------

FSDB Dumper for IUS, Release Verdi_N-2017.12, Linux, 11/12/2017
(C) 1996 - 2017 by Synopsys, Inc.
*Verdi* FSDB WARNING: The FSDB file already exists. Overwriting the FSDB file may crash the prog
*Verdi* : Create FSDB file 'Final.fsdb'
*Verdi* : Begin traversing the scope (Final_tb), layer (0).
*Verdi* : Enable +mda dumping.
*Verdi* : End of traversing.
*Verdi* : Begin traversing the scopes, layer (0).
*Verdi* : End of traversing.
------------------------- Simulation FINISH !!-------------------------
================================================================================

 \(^o^)/ CONGRATULATIONS!!  The simulation result is PASS!!!

================================================================================
Simulation complete via $finish(1) at time 4215 NS + 0
./Final_tb.v:158              #(`CYCLE) $finish;
ncsim> exit
[b06178@cad29 generate]$
```

➢ How do you design your chip to support this extension?

- Finite state machine:
-



➢

- TOP state:這個state是當我們剛收到一個instruction，不知道這是全32bit, 16bit和32bit前半段還是兩個16bit，我

們就要透過32 bit instruction兩個LSB=2'b11的性質判斷這是全32bit, 16bit和32bit前半段還是兩個16bit，若是全32則pc+4，且下一個state還是留在TOP，若是兩個都是16bit instruction則pc+2，且下一個state去BOTTOM 16，若是16bit和32bit前半段則decompress 16bit instruction並且儲存32 bit前半段，pc+4並進入TOP 32 state，下一次就直接拿32 bit後半段當instruciton就不需要stall一個cycle，此外這個state也是初始化時的state。

- BOTTOM 16:這個state是當我們的位置在這個instruction的後半段且確定是16 bit instruction，這時我們就decompress instruction並進TOP state。
- TOP 32:這個state是當我們的instruction的前半段是32 bit instruction的後半段，這時我們就從buffer拿上一個state儲存的32 bit前半段，兩者組合後就是這次的instruction，並且根據LSB的性質判斷這個instruction後半段是16 bit instruction還是32 bit instruction的前半段，若是前者則進入bottom 16，若是後者則將32 bit前半段存進buffer並留在TOP 32。
- Decompress:
    - decompress其實只需將每個16 bit instruction一一對應到其在32 bit 的位置，比較需要注意的是部分Instruction register需要+8，因為16 bit只會用到x8~x15 register。

- Any improvement on the performance, especially on how you reduce the number of cycles to complete the simulation.
  - 我減少simulation time的方法就是我使用buffer存32bit的前半段，因此我可以直接pc+4並將後半段和前半段組合後輸出，不需要stall一個cycle

➢L2Cache

一. 架構

L1 write through, L2 write back(prefer):

在課堂中學過，L1用write through policy，L2用write back policy是最佳的結構，因　　為L1到L2的miss penalty很小，所以可以輕易寫入，L2到memory的misspenalty很大，因此用write back較佳。

L1 write back, L2 write through(our design):

由於我們的write back L2 cache過了hasHazard和noHazard的測資，但L2Cache的過不了，花了幾天debug不成功後決定使用write through policy。

二. L2Cache 架構

我們採用總大小為128 words和256 words的L2Cache來進行比較，因為cache面積較大，若能使用較小的面積可以得到不會差太多的面積的話，那麼A*T value會更好。

seperate cache:

我們採用了64 words和128 words的direct map memory，和64 words和128 words的2 way associative memory，使用LRU。

unified cache:

我們採用了128 words和256 words的direct map memory，並且使用額外的一個bit來記住memory是從instruction memory來還是從data memory來。

三. 互相比較

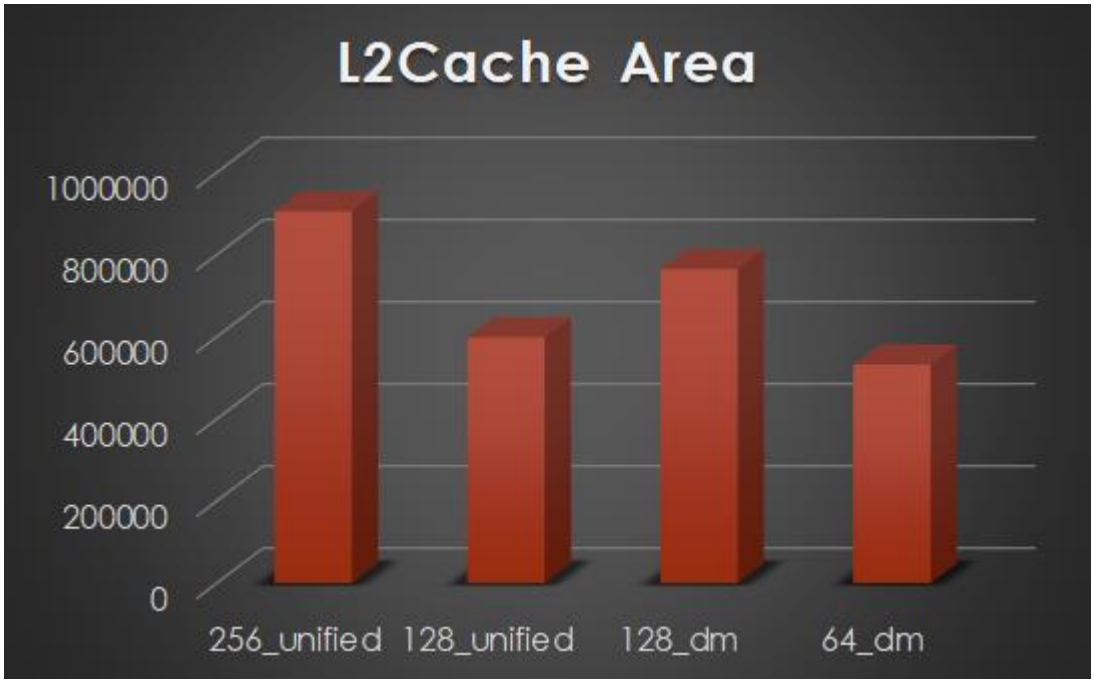
我們使用相同的cycle time下對所有L2Cache進行比較。

	Total cycle	Area(um^2)	A*T value(um^2*ns)
L2_unified_256	41951	905408	41781048*10^4
L2_unified_128	42509	598398	27981030*10^4
L2_dm_128	41964	764786	35302521*10^4
L2_dm_64	42591	532583	24951300*10^4
L2_2way_128	42083	725736	33595300*10^4
L2_2way_64	42641	512888	24057100*10^4

因為我們的2way cahe理論上total cycle要比較小，但是我們的卻比較大，因此可能在LRU的判斷做的不好，因此後來捨棄這個cache。

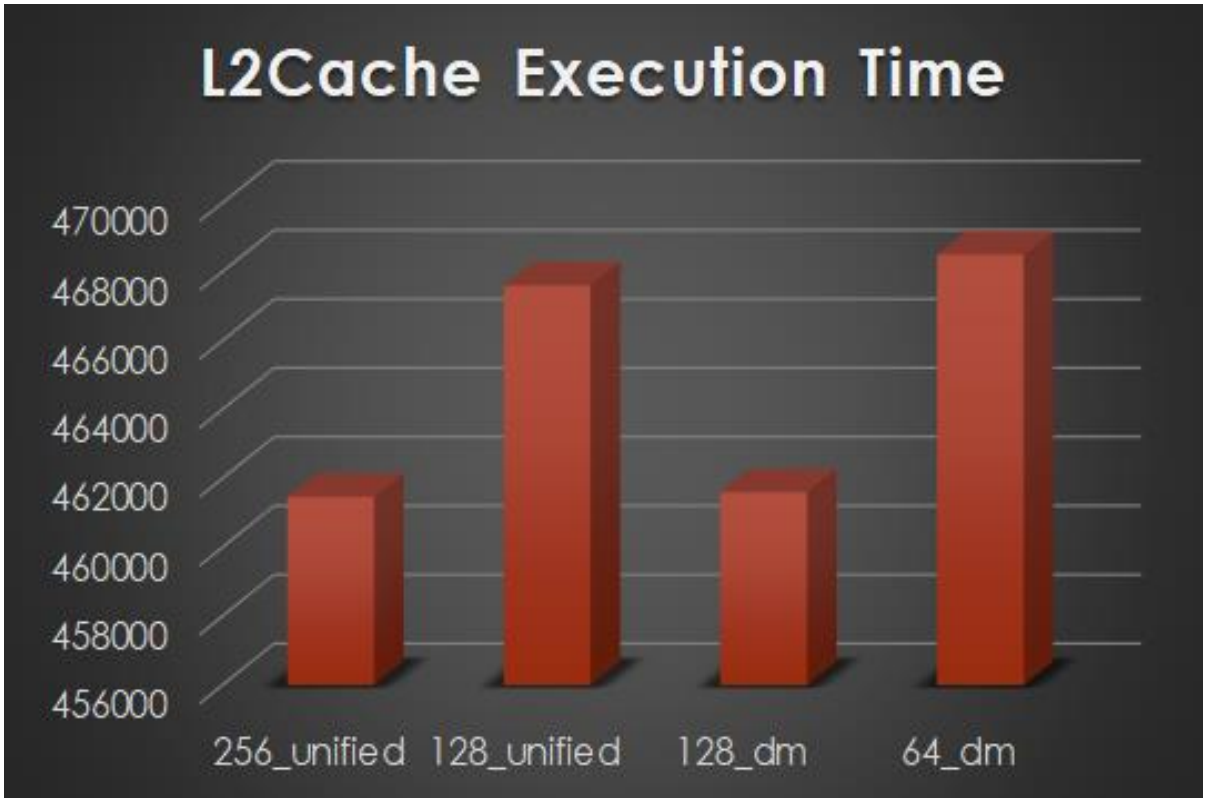
AREA

從下圖可以發現unified的面積會因為要增加判斷的bits和判斷條件的gate，所以會比較大。另外若用較少的block面積也會有明顯的下降。



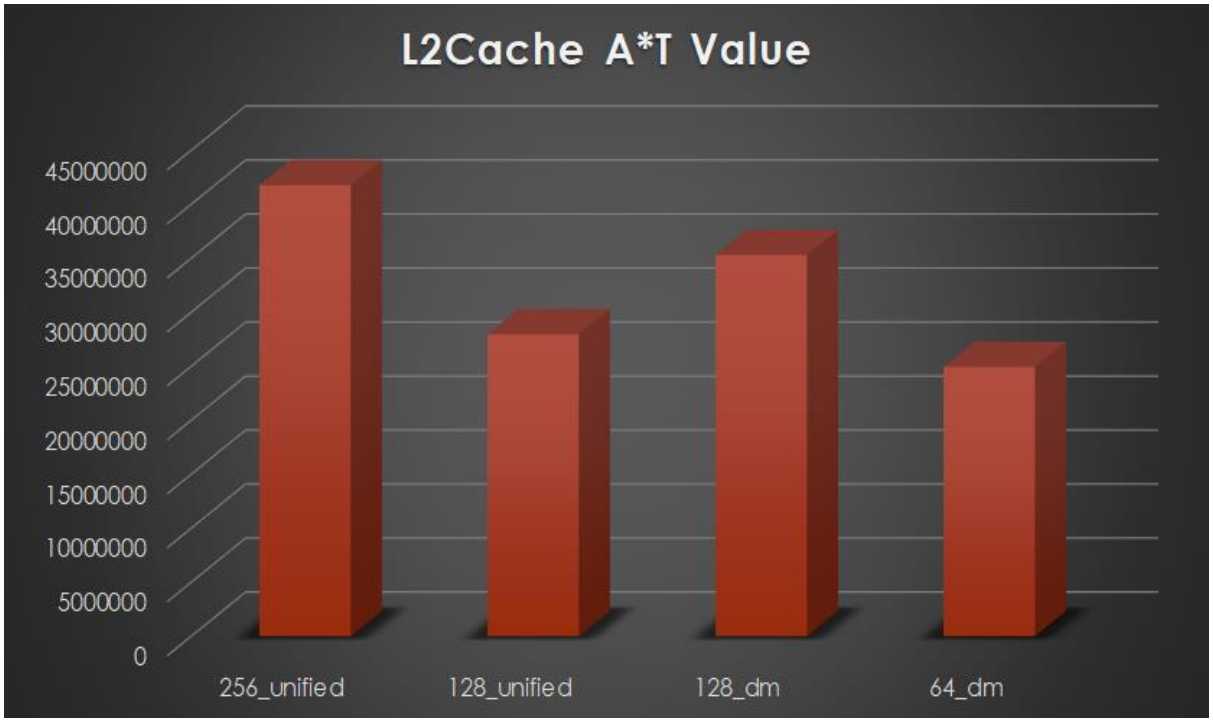
Execution time

從下圖可得知較小的cache會有較長的execution time，不同架構之間的差別也不會太大。



A*T Value

從下圖可得知面積的影響非常大，使總共256 words的cache都明顯較大。



Miss Rate

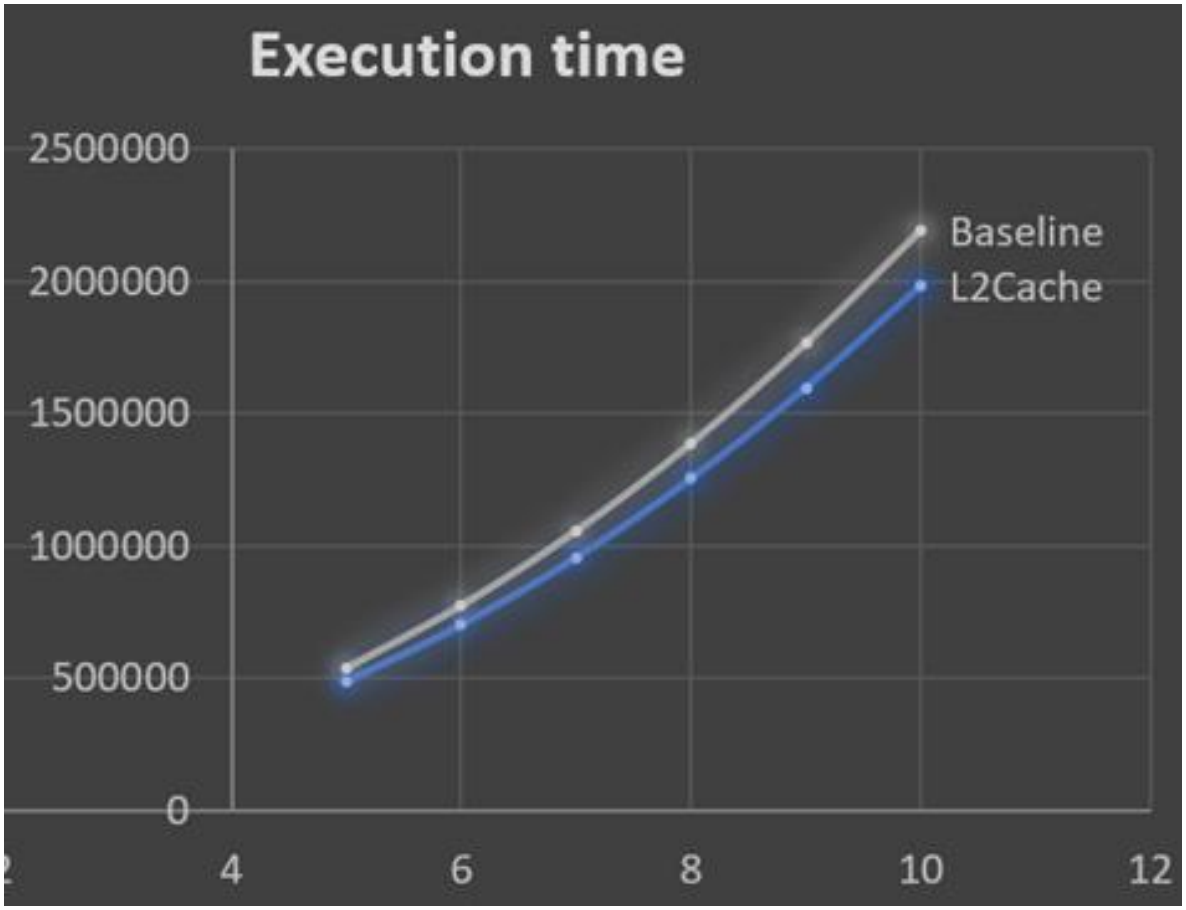
	256_unified	128_unified	128_dm	64_dm
Miss Rate	1.8%	2.1%	1.9%	2.3%

256_unified is the best !(Baseline miss rate: 6.5%)

Dfferent nb

接著我們透過不同的測資來檢驗我們的L2Cache，選用上面時間和miss rate表現最好的256_unified Cache，拿來跟我們的Baseline進行比較，下圖為不同長度的測資下，進入memory cycle 的次數，橫軸是nb，縱軸為次數，從圖中可以看到當測資越長，L2Cache進入memory cycle的次數就會和Baseline進入memory cycle的次數差距越來越大，測資變複雜使L2Cache的功能被更加顯現出來，因為require memory的次數增加，baseline就更容易因為cache較小而增加miss rate，L2Cache雖然也會增加，但是因空間較大所以幅度較不明顯。

因為進入memory次數的差距，導致Baseline和L2Cache之間漸漸出現差異，如下圖所示，橫軸為nb，縱軸為execution time(ns)，達到了我們優化execution time的效果。



結論

在不考慮面積的情況下，256 words的unified cache有最好的miss rate，我們推測是因為data miss rate比instruction miss rate還要高出許多，因此將instruction的部分分給data使用會使miss rate更均衡而且降低，因此最後採用256 words 的 unified cache版本，使用cycle=4.5ns合成，cycle=6.5ns通過gate level simulation。