Lab5\_Report

0710028陳敬諺

1. Code and Calculation Explanation

Total memory stall cycle:

Fig 1.(a): Hit時，address傳到cache，address傳入cache內，再回傳，所經過的memory stall cycle (delay)為1+2+1=4。而miss時，memory傳到cache要經過一次傳送address，一次傳送data，一次access memory content，一次access single cache content，共1+100+1+2=104個delay，但因為cache抓8 words會依序傳送所以一次miss在memory與cache間要等8\*104個delay，再加上一次processor與cache間的4個delay，所以列出算式為(1+2+1)\*(access-miss)+miss\*(1+8\*(1+100+1+2)+2+1)

Fig 1.(b): 和(a)的算法類似，但由於memory到cache一次能傳送8 words所以一次miss的delay縮減為4+104，算式列為

(1+2+1)\*(access-miss)+miss\*(1+(1+100+2+1)+2+1)

Fig 1.(c): 若在L1中hit，所經delay改為1+1+1=3，而L1中miss在L2時hit，L1與L2間的delay為1+10+1+1，但要依序傳送4 words，所以乘以4，之後再加上L1與processor間的delay有3，要是L2中也為miss，則為講義上的算法，delay為1+32\*(1+100+1+10)+4\*(1+10+1+1)+1+1

所以算式記為

miss\_2 \* (1+32\*(1+100+1+10)+4\*(1+10+1+1)+1+1)+

(access\_1\_2 - miss\_2) \* (1+4\*(1+10+1+1)+1+1)+

(access - access\_1\_2) \* (1+1+1)

Execution:

從assembly code中可以看出在k層的loop之中包含了22行指令，j層則除了原來k層的指令再加上5行，且k層迴圈結束時，會先跑過k層的開頭兩行，j層所經過的指令數會是5+k\*22+2，同理i層要經過的指令數5+j\*(5+k\*22+2)+2，最後是外部開頭的兩行指令與i層終止時的兩行指令與最後的運算終止，所以會是2+i\*(5+j\*(5+k\*22+2)+2)+2+1。

Code:

Fig 1.(a)(b)兩者的code在模擬cache的行為上是相同的，只是最後結算total memory stall cycle所用的算式不同，因為memory與cache間傳輸的負載不同。

Fig 1.(c)在L1 cache中出現miss之後必須先access L2 cache，若又是miss，才到memory，在code的實現上，是在兩個cache的架構是一樣的，即讓在L1中miss發生的address立即丟入L2運作。

1. Discussion

以下是4筆測資所算出來的total memory stall cycles，順序依序為

fig 1.(a) fig 1.(b) fig 1.(c):

c1:

6016 1648 12144

c2:

19072 5968 32736

c3:

7626560 1068008 791552

c4:

251426432 35098320 998152040

由上面的資料可大概看出，在上面測資測試下，(b)相對(a)都是表現較好的，因為一次傳送多筆資料使得miss penalty在相同的情況下被減少許多，另外，當在資料量少的時候，(c)的效率明顯較前兩者差許多，因為(c)在讀取memory data會花大量時間取得多筆資料，但在c3的測資中可明顯發現(c)優於另外兩者，可在c4的效率卻又小於另外兩者，推測(c)的架構，在資料大量且集中的時候使用效益會是最好的。

1. Experience

延續lab4，lab5在實現上並不困難，多半是了解fig 1.(c)的架構後，某幾段的code複製貼上，修改參數就好，但我一直到做完才發現一個很嚴重的問題，我lab4的n-ways associative的架構是錯的，我tag的bit忘記隨set做更動，因為當時做出來的結果合幾個朋友的結果是相同的，所以就沒多想，現在才知道自己錯的多離譜，看來上次對於cache的架構那部分並未熟讀過。