

## Lab 6 Report

資工三 409410100 徐佳琪

### 1. FC-DNN 計算的效能瓶頸

MNIST FC-DNN (Fully Connected Deep Neural Network) 是一個處理手寫數字識別數據集的深度神經網路，其神經網路結構為全連接層，並常以深度學習框架實作。經由訓練，學習從 MNIST 數據集中識別和分類手寫數字。

FC-DNN 計算的效能瓶頸有並行性 (concurrency) 的問題、硬體浮點運算以及為了處理指令執行而有的 NOP (no operation)，要提升並行性就要實作多執行緒或向量化指令，這似乎是教授上課所提到的一種優化方式，但自認自己能力無法成功實作此方法，故沒有往此方向優化，雖然最後有小小優化這方面。

在 FC-DNN 中涉及大量的浮點運算，大量的神經網路節點 (neuron) 與權重相乘並相加影響效能，助教在實驗課中有提到硬體浮點加速的方式，新增自定義指令集 (MAC 指令  $d = a * b + c$ )，MAC 指令 (Multiply-Add and Accumulate) 簡化指令，降低在神經網路的運算次數，之前有嘗試實作但並未成功。

最後是為了處理指令執行而有的 NOP，也就是為了讓管線 (pipeline) 能夠執行而需要增加的。no operation 讓執行指令執行至 Rd 有正確結果能傳給之後有依賴此指令的 register，卻也讓執行時間拉長，降低效能。因此，放棄實作加速浮點運算後，以及考慮到萬一完成新增 MAC 指令但無法跑 FPGA 板的不確定性，採取較保守的方法，往儘可能降低 NOP 的數量之方向優化，以下將說明優化方式及結果。

### 2. 優化方式描述

在硬體優化方面，儘可能降低 NOP 的數量，可以新增 pipeline interlock 機制 (以 Stall 取代 NOP) 或是以 forwarding 機制處理 data hazard 的方式，實作方式以 forwarding 降低 NOP 數目，圖一為以 forwarding 實作在 hardware 資料夾 ex\_pipe.v 的架構全覽圖，主要是由四條 forwarding 的接線組成，其中 lwc1 與 mul.s 間還要有一個 NOP 才能確保 forwarding 到。應該是該以新的.v 檔撰寫 forwarding unit 較為合適，但擔心自己會寫錯，所以用較熟悉的寫法，直接在 alu\_ctrl 判斷後接線，希望能在課餘時間再嘗試以新的.v 檔寫。

Instruction \ stage											
addi \$5, \$5, 4	F	D	(X)	(M)	W						
lwc1 \$f1, 0(\$5)		F	D <sup>1</sup>	(X)	M	W					
addi \$5, \$5, 4			F	D	(X)	M	W				
lwc1 \$f2, 0(\$5)				F	D	(X)	M	W			
NOP					nop	nop	nop	nop	nop		
mul.s \$f3, \$f2, \$f1						F	D	(X)	M	W	
add.s \$f4, \$f4, \$f3							F	D	(X)	M	W

圖一、forwarding 實作架構全覽圖

Instruction \ stage											
addi \$5, \$5, 4	F	D	X	1 M	W						
lwc1 \$f1, 0(\$5)		F	D	X	M	W					
addi \$5, \$5, 4			F	D	X	1 M	W				
lwc1 \$f2, 0(\$5)				F	D	X	M	W			

圖二、第一條 forwarding 接線

一條一條實作接線，確定寫對再判斷下一條，以方便除錯，testbench 測對，再替換 software 組合語言。首先，觀察在 im\_data 資料夾的 im.txt 的 machine code，在 beq 指令後的兩次加法及儲存值之間存在資料依賴 (Data dependency)，為避免 data hazard 而增加 3 個 NOP。若要移除 NOP，以 if (X/M.Regwrite & (X/M.RegisterRd = D/X.RegisterRs\_fp)) 判斷式找出資料依賴發生處，X/M.Regwrite 為在 XM 階段的 reg\_write，X/M.RegisterRd 為在 XM 階段的 Rd 編號數，D/X.RegisterRs\_fp 是 DX 階段的浮點數 Rs 編號數，確認寫入暫存器編號為即將用到的暫存器編號，再直接將 XM 階段的運算值 (即 alu\_out\_xm) 取代原本需要等 3 個 NOP 才有正確值的浮點數 Rs (即 alu\_src1\_fp)。上述為圖二所示的 forwarding，需注意的是若 D/X.RegisterRs\_fp 在 DX 階段取值後，要再延遲一個 clock 讓 XM 階段可以順利取值判斷，此條 forwarding 可減少 4 個 NOP，因為還存在兩次加法間的資料依賴。

Instruction \ stage											
addi \$5, \$5, 4	F	D	X	M	W						
lwc1 \$f1, 0(\$5)		F	D	X	M	W					
addi \$5, \$5, 4			F	D	X	M	W				

圖三、第二條 forwarding 接線

為了避免兩次加法間的 data hazard 而伴隨的 2 個 NOP，以判斷式 if (M/W.Regwrite & (M/W.RegisterRd = D/X.RegisterRs)) 找出資料依賴發生處，M/W.Regwrite 為在 MW 階段的 reg\_write，M/W.RegisterRd 為在 MW 階段的 Rd 編號數，D/X.RegisterRs 是 DX 階段的 Rs 編號數，做確認後直接將 MW 階段的運算值 (即 alu\_out\_mw) 取代原本需要多 2 個 NOP 才有正確值的 Rs (即 alu\_src1)，上述為圖三所示的 forwarding。

Instruction \ stage											
lwc1 \$f2, 0(\$5)				F	D	X	M	W			
NOP					nop	nop	nop	nop	nop		
mul.s \$f3, \$f2, \$f1						F	D	X	M	W	

圖四、第三條 forwarding 接線

接著，為避免兩次加法後的 lwc1 與 mul.s 間的 data hazard，以判斷式

if ( M/W.Regwrite & (M/W.RegisterRd\_fp = D/X. RegisterRs\_fp)) 找出資料依賴發生處，M/W.Regwrite 及 D/X.RegisterRs\_fp 同前所述，M/W.RegisterRd\_fp 為在 MW 階段的浮點數 Rd 編號數，因為 lwc1 將值存入記憶體，做確認後直接將 MW 階段記憶體值 (即 mem\_data\_to\_reg\_fp) 取代原本要輸入浮點乘法 (fp\_mul) 的浮點數 Rs (即 alu\_src1\_fp)，且需多加 1 個 NOP 才能執行，上述為圖四所示的 forwarding，圖三及圖四的邏輯相似。

Instruction \ stage											
mul.s \$f3, \$f2, \$f1						F	D	X	4 M	W	
add.s \$f4, \$f4, \$f3							F	D	X	M	W

圖五、第四條 forwarding 接線

最後，為避免 mul.s 與 add.s 間的数据 hazard，以判斷式 if ( X/M.Regwrite & (X/M.RegisterRd\_fp = D/X. RegisterRt\_fp)) 找出資料依賴發生處，X/M.Regwrite 同前所述，M/W.RegisterRd\_fp 為在 MW 階段的浮點數 Rd 編號數，D/X. RegisterRt\_fp 是 DX 階段的浮點數 Rt 編號數，做確認後直接將 MW 階段運算值 (即 alu\_out\_fp\_xm) 取代原本要輸入浮點加法 (fp\_add) 的浮點數 Rt (即 alu\_src2\_fp)，便可以減少 3 個 NOP，上述為圖五所示的 forwarding，圖二及圖五的邏輯相似。

再者，在硬體的優化中，還可以更換指令順序 (Instruction rescheduling) 減少一個 addi 指令數，將 beq 後的兩次加法合併，提升效率，也算有優化到並行性。軟體方面的優化，只有降低 mips\_dnn.c 的計算加權總和的迴圈數。

更換前指令	更換後指令
addi \$5, \$5, 4	addi \$5, \$5, 8
lwc1 \$f1, 0(\$5)	lwc1 \$f1, 4(\$5)
addi \$5, \$5, 4	lwc1 \$f1, 8(\$5)
lwc1 \$f2, 0(\$5)	

圖六、更換順序前後的指令

### 3. 優化結果

```
Accuracy:      0.980200 (9802 / 10000)
Performance counter stats for './mips_dnn':
   204365990025      cycles
   16063238966      instructions
    89128277      cache-misses
   5062397317      cache-references
306.765632130 seconds time elapsed
```

▲ 優化前，執行時間約300秒

```
Accuracy:      0.980200 (9802 / 10000)
Performance counter stats for './mips_dnn_new':
   151231904582      cycles
   13513509841      instructions
    79248613      cache-misses
   4584610584      cache-references
227.141534841 seconds time elapsed
```

▲ 優化後，執行時間約225秒

摘自 CO\_LAB6a.pdf 的第 19 頁，做為優化前數據，及降低 mips\_dnn.c 的計算加權總和的迴圈數的優化數據。圖七至圖九皆是已有在軟體方面優化的條件下。

```

references ./mips_dnn/mnt# perf stat -e cycles,instructions,cache-misses,cache-r
Zynq_BASE mapping successful :
0x40000000 to 0x8c040000, size = 65535
Accuracy:      0.980200 (9802 / 10000)

Performance counter stats for './mips_dnn':

    137311845441      cycles
    12977735068      instructions          #    0.09  insn per cycle
    76793561      cache-misses          #    1.733 % of all cache refs
    4430371657      cache-references

    206.216410069 seconds time elapsed

    205.400000000 seconds user
    0.560000000 seconds sys

```

圖七、只做完第一條 forwarding 接線的 perf 結果

```

ns,cache-misses,cache-references ./mips_dnn stat --repeat 1 -e cycles,instruction
Zynq_BASE mapping successful :
0x40000000 to 0x8c040000, size = 65535
Accuracy:      0.980200 (9802 / 10000)

Performance counter stats for './mips_dnn':

    113981648457      cycles
    12292155072      instructions          #    0.11  insn per cycle
    76794055      cache-misses          #    1.828 % of all cache refs
    4199930766      cache-references

    172.305828693 seconds time elapsed

    170.380000000 seconds user

```

圖八、做完全部四條 forwarding 接線的 perf 結果

```

ns,cache-misses,cache-references ./mips_dnnperf stat --repeat 1 -e cycles,instruction
Zynq_BASE mapping successful :
0x40000000 to 0x8c040000, size = 65535
Accuracy:      0.979600 (9796 / 10000)

Performance counter stats for './mips_dnn':

    110609611483      cycles
    12187781997      instructions          #    0.11  insn per cycle
    76570735      cache-misses          #    1.848 % of all cache refs
    4142569265      cache-references

    166.437847846 seconds time elapsed

    165.250000000 seconds user
    0.660000000 seconds sys

```

圖九、之後更動指令順序的 perf 結果

#### 4. 結論與心得

從圖七時的執行 36 個指令到圖八時的執行 29 個指令，進行完 forwarding 優化機制後，從 206 秒到 172 秒，優化前後的 perf 數據顯示執行時間縮短 34 秒。由此得知 forwarding 機制減少指令數量，提高 pipeline 的效率，解決資源競爭，也降低指令執行的停頓時間，進而提高程式執行效能。

再從圖八做完 4 條 forwarding 接線到圖九的更換指令的 perf 數據得知，雖然只少了一個指令數，但也從原本的 172 秒降至 166 秒，少了約 6 秒。這個是之前在組合語言的課程有學習過的方法，多虧同學的提點，才發現還有這個地方可以優化，找回關於組合語言的知識，也增加程式的並行性。而更動 mips\_dnn.c 的計算加權總和的迴圈數，大概能降低 75 秒。

perf 數據的前後對比顯示出 forwarding 與其他方式優化的實際效果。其實從軟體上降低運行迴圈數，讓整體效能提升是最明顯的，這應該是與學期初學到的 performance 公式有關，可能是硬體優化了 clock cycle time，但 clock 時長很短，所以減少時長並沒有像直接減少迴圈數來的多。

如果要繼續優化的話，可能是要用 branch prediction 解決 branch hazard，或者實作新增 MAC 指令。

實作這個 lab 等於是複習整學期除了 cache 外的知識，原本只是大概了解 datapath 跟不同執行階段還有 pipeline 相關知識，實作 lab6 之後，有了更深的體悟，以及更加注意不同指令讀寫記憶體的位置。本來覺得要在一個大架構下，讀懂並實作優化難度很高，但實作過程中，除了一開始對於接不同.v 檔的線疑惑外，以及小心時脈上的問題，後續都沒有想像中的那麼困難，雖然還是遇到些問題，但所幸在許多人的協助下最後有解決。