HW#5 Report

賴御安, 110550168

***Abstract*—**本文針對如何透過硬體來使特定的運算加速。在本篇中是利用**MLP neural network**中**inner product**的部分進行實驗，比較對象為以**C code**撰寫的**inner product**和透過**Floating Point IP**連接**Aquila**來進行**inner product**。最後會對兩者的計算時間進行比較。**(*Abstract*)**

Keywords*— DSA; neural network; IP Catalog; *Aquila; MLP; floating point IP; Verilog; (key words)

I. Introduction

本次作業是針對MLP neural network中inner product的部分進行加速，由於做inner product會有許多的乘法運算，因此決定運算時間的主要便會是運算乘法的速度。在觀察透過C所寫的程式碼編譯出的objdump，可以發現運算inner product的方式是透過跳到 __mulsf3 和 __addsf3 兩處進行運算。細看這兩個函數，他們分別有205和284條指令，即使不計算stall cycle的數量也需要花費相當多的cycle數去進行計算。因此，若能透過某些方式進行加速，便能省下不少cycle數，同時再加上MLP中會有許多次的inner product計算，便能將MLP的運算速度提高。這部份便是這次作業的重點。

II. C code的觀察與修改

這次我們需要對neural network中計算兩個vector的inner product的for迴圈進行修改，將原先的inner_product計算方式傳到floating point IP進行計算。

A. 初步想法

在教授針對這次作業進行講解時有提到，我們可以將變數透過memcpy進行搬移，等data feeder接到值後再放進IP進行計算。事後與其他同學討論時，得出可以先給定變數的位置再進行搬移，這樣在硬體端就可以對特定的位置進行存取，讀出想要的值並放到對應的位置。因此我便著手進行修改。

我將p_neuron、p_weight透過memcpy搬移到指定的地址。這邊我透過觀察soc_top.v中Core2AXI_0的實作以及select和out的部分，發現註解上有DSA device的設定，我便將位置訂在0xC400_0000和0xC410_0000。另外我也想到我需要將inner_product當前的值從下來作為data C加法的部分，因此便將inner_product搬移到0xC420_0000。最後便將計算後的結果存在0xC430_0000，搬回inner_product存放的位置。

B. 遇到的問題

在使用memcpy時，我發現在編譯時會跳出memcpy會覆蓋volatile的警告。起初我並未重視這個警告，便開始對make出來的elf檔測試。這時我發現dev_addr會跑出0xC400_0000 - 0xC4FF_FFFF的範圍並且不會計算出結果。這時我在硬體端進行許多修改，但都未見成效。最後我詢問其他同學是否有相同問題時，才得以會得解答。我會在下一部份詳細敘述最後的處理方式。

C. 解決辦法

在向同學請教後我得知我並不是只能依靠memcpy來做資料的轉移，而可以直接利用*data1 = p_neuron (data1是指定的位置，float type的volatile pointer)的方式將p_neuron以及其他計算inner_product會用到的值放到我指定的位置。將memcpy替換成上述方式後，便是我最終對這個問題的解決方式。

D. 分析*objdump*

觀察修改過後的c code所產生的objdump可以發現原先會跳去執行的 __mulsf3 和 __addsf3變成從memory load & store，也就是將當前的p_neuron、p_weight、inner_product存到memory，等到計算出result並存入指定的位置後，inner_product再從那個位置讀出值。這樣的指令數量就驟降至4個，即使加上floating point IP計算結果的cycle數也會比透過 __mulsf3 和 __addsf3去計算還少。

III. 硬體的觀察與修改

這次在硬體端的修改則是根據教授上課給的提示，參考CLINT的實作方式。在實作時，我也參考了其他的模組，並最後實作出data_feeder以及其與floating point IP 的溝通，完成最後一步。底下將分為對soc_top.v的修改以及data feeder的實作。

E. soc_top.v

在這個檔案中最重要的是Aquila和device之間溝通的方式。前面有提到我參考CLINT以及其他模組，可以發現在Aquila的I/O port中有一組I/O device ports，分別是以dev作為前綴。這邊的信號會拉到其他device並在內部進行其他運算。仔細trace code後發現其中Device address decoder的部分會決定我最後送回Aquila的值是多少。因此我便在仿造Uart device在這部分的定義，並配合原先寫在註解DSA device的位置進行設定，透過判斷dev_addr的範圍是否在0xC400_0000 - 0xC4FF_FFFF間來分辨DSA device。

F. data feeder

透過仿造CLINT的方式，我決定了我的data feeder的I/O ports，在拉線上不一樣的就是en_i是看是否是DSA device的address範圍，而輸出的訊號則是兩個新定義的wire，會回到Device address decoder來判斷送回Aquila的值。

接著便開始敘述對於data feeder內部的實作。首先，資料進來後，會先根據en_i和we_i的值來判斷，當都為1時才需要將輸入的值存進buffer中。接著則是依照進來的address決定要放到data A、B或C的buffer中，這邊的A、B、C則是對應floating point IP中的A、B、C的data。當data C讀入後便啟動floating point IP。輸出的結果則會不斷給到data_o中，透過判斷result_valid，也就是floating point IP中的result valid來決定

是否是需要output結果。底下的Fig.1便是Aquila跟data feeder和floating point IP的流程圖。

G. Floating point IP

對於添加這個IP的方式便是依照講義教學，透過IP catalog加入floating point，選擇Fused-Multiply-Add和Both operation。另外我到options選擇non-blocking，這樣我便能在資料傳進來後就啟動floating point IP的計算。對於Latency我則沒有特別設定，使用的是Max Latency，也就是16 cycles。在底下的流程圖中，我並未畫上VALID的PORT，其中，對於data A、data B、data C的valid設定則是等到data C進到buffer時會同時將三個valid啟動。可以這麼做的原因則是由於資料進入而存進buffer會依照順序，因此當data C進入後便能啟動計算。

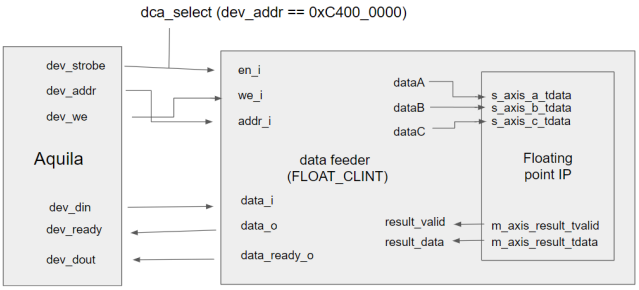

Fig. 1. 流程圖。dataA, dataB, dataC, result_data是register in data feeder。箭頭方向是資料傳輸方向。

H. 特別處理

在分析ILA抓下的電路時，我發現在第一筆的result計算完成時的address並非是他應該存放的位置，也就是0xC430_0000，因此我在data feeder中有特別寫一小部分的電路去處理。實作方式為當result_valid變成1時，我會將一個自訂的flag設為1，且在接下來的cycle中data feeder只會去檢查address是否為0xC430_0000，當符合時才會將data_ready_o設為1，確保在正確的request到來前不會將計算好的result輸出。

IV. 數據分析與討論

首先我先在我的本地端試跑未修改前的程式碼，再跑修改後的程式碼，確定兩者的比較是在同一個平台上。

```
(1) Reading the test images, labels, and neural weights.
It took 5617 msec to read files from the SD card.

(2) Perform the hand-written digits recognition test.
Here, we use a 3-layer 784-48-10 MLP neural network model.
Begin computing ... tested 100 images. The accuracy is 85.00%

It took 22137 msec to perform the test.
```

Fig. 2. 未修改程式碼所需要花費的時間

```
(1) Reading the test images, labels, and neural weights.
It took 5826 msec to read files from the SD card.

(2) Perform the hand-written digits recognition test.
Here, we use a 3-layer 784-48-10 MLP neural network model.
Begin computing ... tested 100 images. The accuracy is 85.00%

It took 4601 msec to perform the test.
```

Fig. 3. 修改程式碼後所需要花費的時間

	feed time	calculating time
cycle	15244813	60979200
time (s)	0.3658755091	1.463500788

Fig. 4. 在16cycle的latency下的feed time和calculating time，其中feed time cycle是由input feeding cycle 11433600 加上output feeding cycle 3811213而來

上方的圖便是我在latency是16個cycle的情況下算出的數值，將input feeding cycle平分給3個data的input後，得到的是3811200，也就是運算次數。將運算次數乘上latency後得到的cycle數確實符合最後獲得的cycle數。

接著我嘗試調整latency，看看是否能找到在Aquila的平台下使用floating point ip的最小latency。我根據Product Guide中的說明將latency調整為類似他舉例的情況並逐步減少，最後得到latency的最小值為2。在Product Guide中有關latency的敘述中，我也發現理論上的最小值就是2。實際驗證過後也是正確的。底下便是我所得出最好的優化時間以及feed time & calculating time。

```
(1) Reading the test images, labels, and neural weights.
It took 5826 msec to read files from the SD card.

(2) Perform the hand-written digits recognition test.
Here, we use a 3-layer 784-48-10 MLP neural network model.
Begin computing ... tested 100 images. The accuracy is 85.00%

It took 3340 msec to perform the test.
```

	feed time	calculating time
cycle	15244827	7622400
time (s)	0.3658758451	0.1829375985