DNN accelerator: The hardware design and power analysis of RS, WS, and OS

DNN 加速器：RS 與 WS、OS 的硬體設計與能耗分析

指導教授: 黃朝宗　組別：B156

組員: 李宇洲、王傑立、王靖淳

摘要

深度學習運算過程會需要使用到大量的資料，包括 image、filter weight、bias 等且儲存在記憶體中，多次存取記憶體會提高 energy consumption，因此出現幾種 dataflow 提高 energy efficiency，例如 WS (weight stationary)、OS (output stationary) 和 RS (row stationary)等。其中在 Efficient Processing of Deep Neural Networks[1]這篇 paper 中表示 RS 的 energy efficiency 最好，所以我們想知道是否在一些情況下，其他 dataflow 的 energy efficiency 反而會隨著應用、model、convolution layer、image size 等的不同而比 RS 還要好？

我們希望可以模擬深度學習在實際應用中的運算情況，因此我們選擇 style transfer 作為加速器的應用。我們的研究方法為，先訂定規格，例如圖片大小、FPS、clock rate、PE (processing element)數量等。設計三種 dataflow 的硬體架構，完成 RTL code 以及測試 code 的正確性。合成 RTL 後，利用 pre-layout gate-level simulation 的 waveform 做精準的 power estimation。最後我們會探討在不同情況下，使用哪一種 dataflow 可以有較好的 performance。

內容

一、 研究動機

DNN (Deep Neural Network)在影像辨識上有相當高的正確率，然而需要進行非常大量的運算，於是 DNN 加速器的設計成為了一個值得研究的課題。DNN 中的運算大多數是規律性的 convolution，架構由加法器和乘法器所組成。

相較於 CPU、GPU，加速器的運算速度已經快上許多，因此我們想要研究的重心會放在 energy consumption 上。其中加速器有幾種不同的 dataflow，例如 WS

(weight stationary)、OS (output stationary)、RS (row stationary)等，而我們的目標會是從這些 dataflow 中，比較不同情況下應該適合使用哪一種 dataflow。

二、 背景簡介

在我們專題中所求得的 feature map 計算過程如圖 1 所示，我們會先將 image 或 ifmap (input feature map)以及 weight 做量化後做 convolution，可得 ofmap (output feature map)。不過由於是量化過的 output，因此我們還需要去量化，才能得到接近真實的 ofmap 並計算 loss。其中 DNN accelerator 是我們這次需要討論的區塊。

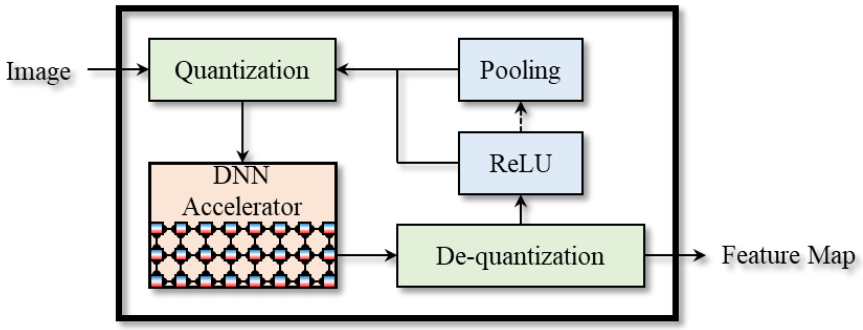


圖 1 Feature map 計算流程圖

在我們的實作裡所使用的 model 是 VGG-19。VGG-19 pretrained model 是一個從 ImageNet 中的 100 萬張圖片，共 1000 種不同類別所訓練出來的模型，也就是說 VGG-19 原本是用於圖片分類。不過 VGG-19 同時也能作為 feature extractor 的 model，所以 VGG-19 的 convolution layer 也被廣泛的應用在 style transfer 上。

三、 Dataflow

在深度學習加速器中，由於 convolution 重複利用 data 的頻率相對比較高，因此我們需要利用 dataflows 提高 data reuse，減少不必要的 energy cost。我們總共會比較三種 dataflows，分別是 WS、OS 及 RS。

1. Weight Stationary (WS)

WS 是一種重複利用 weight 的 dataflow。WS 將 weight 固定在每一個 PE 中，activation 則是會從 global buffer 傳遞給每一個 PE。每經過一個 PE，partial sum 就會更新(累加)一次。

2. Output Stationary (OS)

OS 的主要設計理念為減少 partial sum 的存取。DNN 加速器每經過一個 MAC，就必須更新一次 partial sum，因此將 partial sum 固定存在 PE 中的 register 內，就不需要至 global buffer 存取 partial sum。我們將 activation 以及 weight 傳遞至 PE array 中，partial sum 則被固定在每一個 PE 中而不會流動。

3. Row Stationary (RS)

RS 設計理念是希望能最大化 weight 和 activation 在 PE array 中的使用次數。不同於 WS 和 OS，在 filter 大小為3×3的情況下，row stationary 的 PE 中會有 3 個 registers 儲存 ifmap，3 個 registers 儲存 weights，以及 1 個 register 儲存 partial sum。由於 row stationary 的架構和 filter 的大小有關，所以在之後的討論中都以 filter 大小為3×3的狀況討論。

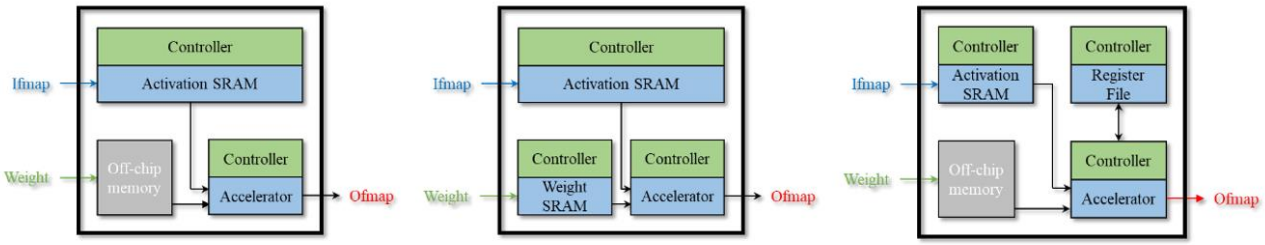
四、 系統設計

我們的 input image 選用128×128 (RGB)作為基準，在 clock rate 為 250MHz 的情況下，我們希望 FPS 至少要有 50 以上，因此我們選擇使用$128 \times 9 = 1152$個 PEs 以符合這個標準。

WS 的 weight 會固定在 PE array 中且只會被使用一次，而 ifmap 會不斷地被重複使用，因此 weight 會儲存在 off-chip memory，而 ifmap 則是會儲存在 SRAM 中。

OS 在限制 PE 數量的前提下，同一 output channel 需分批計算，一批同時計算不同且有限個 input channel，因此不同於 WS、RS，OS 的 weight 與 ifmap 都需要被重複使用，因此使用 SRAM 儲存。

RS 的方式和 WS 相似，都是讓 weight 固定在 PE 中，流動的則是 ifmap，因此在記憶體的配置上也和 WS 相似。ifmap 因為需要重複使用，所以用 SRAM 儲存，而 weights 只需要讀取一次，所以用 off-chip memory 儲存。不過 RS 需要以 row 為單位來計算 ofmap，在我們所定的 PE 數量下，RS 無法直接計算出 ofmap，需要分批計算，因此 RS 會需要額外的 register file 來暫存 ofmap partial sum。



(a) WS 系統架構　(b) OS系統架構　(c) RS系統架構

圖 2 系統架構

五、 實驗成果與結論

我們利用 python 去做 quantization 及 de-quantization，並建立出每一層 convolution layer 的 ifmap、ofmap 作為我們驗證電路正確性的根據。比對並確定完電路正確性之後，利用 Synopsys Design Compiler 合成電路產生 netlist 及驗證 gate-level simulation。

合成規格：

- TSMC 40nm 製成
- Condition: SS corner
- Operating voltage: 0.81V
- Temperature: 125℃

表1 合成結果

	WS	OS	RS
Timing	4 ns		
Area	9,488,890 μm^2	11,359,573 μm^2	9,425,788 μm^2
SRAM size (8-bit)	1,048,676	1,638,400	1,048,676
SRAM bandwidth (8-bit)	384	200	128
Total cycles	4,512,512	4,600,005	3,747,732
Throughput	55 FPS	54 FPS	66 FPS
HDL	Verilog		
Synthesis tool	Synopsys design compiler		

表2 不同 layer 下三種 dataflow 的 total energy cost

單位：mJ

	Conv2d_1	Conv2d_2	Conv2d_3	Conv2d_4	Conv2d_5
WS	0.03698	0.82383	0.47104	1.29499	0.75693
OS	0.09103	0.58874	0.29437	0.55384	0.27704
RS	0.03068	0.62643	0.34414	0.69398	0.36106

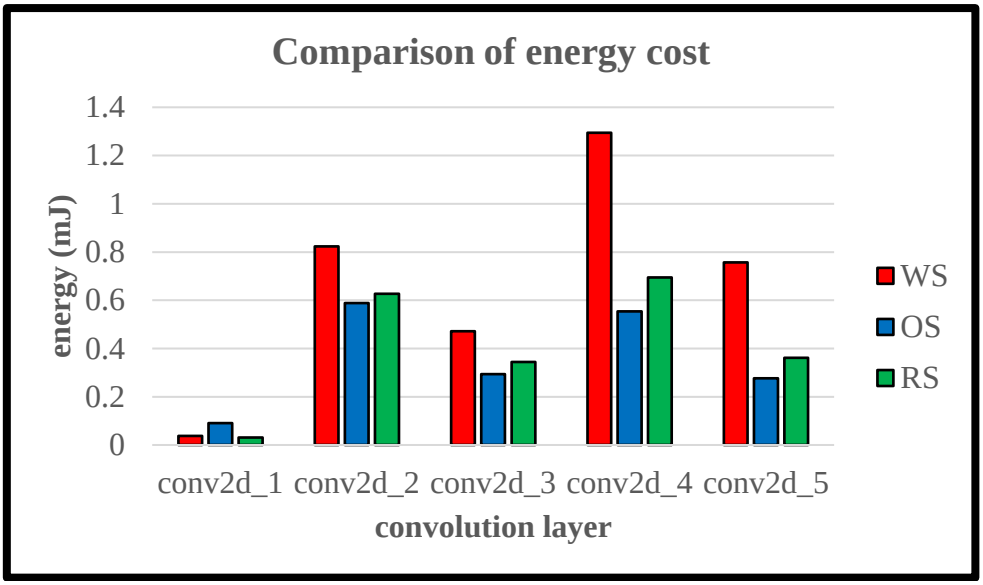


圖3 不同 layer 下三種 dataflow 的 total energy cost 直條圖

表3 不同 layer 下三種 dataflow 的 memory energy cost

單位：mJ

	Conv2d_1	Conv2d_2	Conv2d_3	Conv2d_4	Conv2d_5
WS	0.02190	0.70291	0.39223	1.13574	0.64017
OS	0.08165	0.53370	0.26685	0.50203	0.25102
RS	0.02620	0.55653	0.28760	0.57589	0.30220

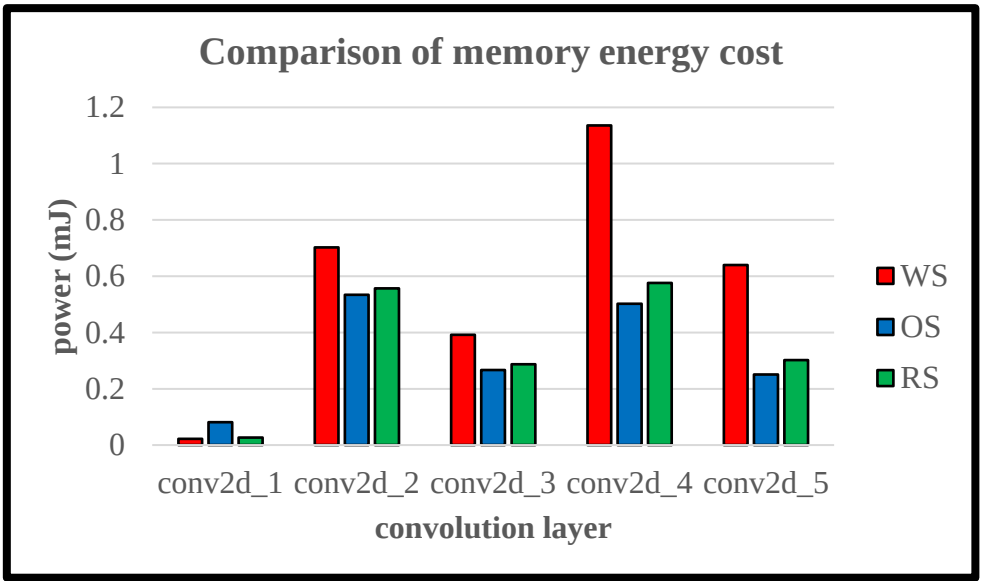


圖 4 不同 layer 下三種 dataflow 的 memory energy cost 直條圖

根據實驗結果，在我們的規格上，若 energy 最重要則選擇 OS，若 area 最重要則選擇 RS。

六、 心得

一開始做這個專題是希望能夠以一般課堂上學不到的方式接觸到數位硬體實作，加上其牽涉到的應用我們也很感興趣，因此毅然決然地報名黃朝宗老師的專題題目。

然而這個題目所需了解的背景知識我們涉略的還不夠深，所以第一學期我們先在寒假自行讀書、利用線上課程網站題目熟悉 python、pytorch 語法及架構，開學後緊接著練習數位電路的流程。後半學期則是閱讀關於深度學習與加速器的 paper，每次在學一個新知或技能時，我們都有不斷的練習，讀 paper 時老師也指導我們不是一昧地接受 paper 中的資訊，我們需要懷疑它，況且有些實驗是沒有說明完整的，對於有疑點的部份我們自己實作一次，可以解除我們的問題也能幫助我們更加了解這些架構，所以我們也實際練習了 paper 中的設計。

在第一學期期末時，我們開始思考我們想做的題目，這部分是最難的，上半學期雖然不像一般課程上課聽講回家完成作業，主要是自己要自動自發地去發現、了解新事物，但總歸來說仍然是學習，而思考題目我們需要藉由學到的內容、我們可以達到的程度還有我們做這個題目的意義以及與別人不同的地方在哪等等的條件來發想，這對我們來說是比較陌生的，所以這段時間我們跟教授討論，詢問學長可行度及需要考慮的內容，最終才在第二學期初真正確定題目，而中間的暑假我們對於之後可能的研究有個方向，所以這段時間自行找 paper，了解其使用的 model，並思考如何做成電路。

最後在第二學期的實作過程中，在設計過程遇到許多問題，沒有研究經驗造成起初沒有準確定義比較基準，也因為專題所做的數位電路設計的流程會比我們同時修的積體電路設計實驗的課程內容先完成，所以有些我們第一次使用的工具，加上改用 TSMC 40nm 製程合成電路，有些條件是我們沒有概念的，發現這些問題時再重新設計電路、摸索如何使用這些工具，使得我們無法達成預期的進度，在時間壓力下也懷疑過自己是否能夠完成，很感謝學長在這段時間不斷的引導我們該做的事、指導我們正確的概念，花了許多時間和我們一起討論，協助我們解決問題。也很感謝老師開會時提醒了許多我們忽略的細節，以及指導我們該如何思考或是我們在做某些事時其實應該要了解的意義，這類大方向的人生建議也讓我們受益良多。

回顧這一年，專題讓我們過得非常充實，我們從每次的錯誤中得到不少教訓並且銘記在心，老師、學長也分享了許多經驗及技巧，讓我們在這一年收穫滿滿！

七、 參考文獻

[1] Sze V, Chen YH, Yang TJ, Emer JS, “Efficient Processing of Deep Neural Networks: A Tutorial and Survey,” Proc. IEEE 2017.