Proj.2

Zexian Deng, Yaotian Liu
51902191xxxx, 519021910090

I. INTRODUCTION

FOR Proj.2, we implement a CNN accelerating architecture based on *Optimizing the Convolution Operation to Accelerate Deep Neural Networks*[2].

The research question of the paper relies on the fact that prior works lack fully studying the convolution loop optimization before the hardware design phase, resulting in accelerator which hardly exploit the data reuse and manage data movement efficiently. So the author of this paper studies CNN-related techniques, e.g. loop unrolling, tiling and interchange, by quantitatively analyzing and optimizing the design objectives of the CNN accelerator based on multiple design variables. At the same time, based on conclusions above, the author proposed a hardware CNN accelerator which concerns:

- limited computational resources
- storage capacity
- off-chip communication

providing simultaneous maximization of resource utilization and data reuse, and minimization of data communication.

The reason we choose this paper is that, it provides not only a smart new design, but presents a very detailed and in-depth analysis of the basis of convolution loop optimization as well, making it a perfect choice for us to study.

Below is the scheme proposed:

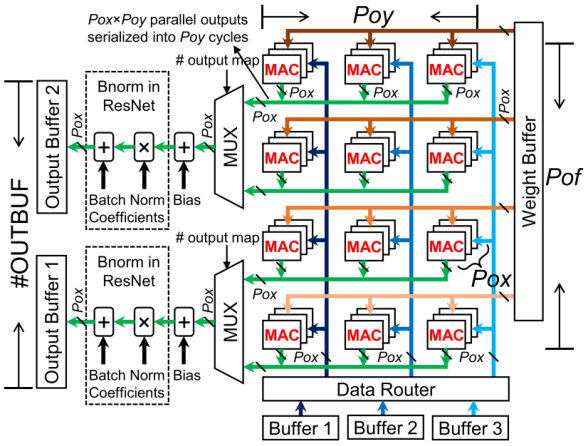

Fig. 1. Convolution acceleration architecture with Pox × Poy × Pof MAC units.

During the process of our reproduction, we encounter some difficulties and also discover some flaws that should be corrected or could be further optimized.

II. DIFFICULTIES & ANALYSIS

A. MUX

The MUX is a module, acting like a regular multiplexer but with a more developed function, providing a further serialization of the outputs from MAC. However, relevant introduction in the paper is far from accurate and detailed, and the figure is also a little deceptive.

According to the paper, *"Since Poy < Nkx × Nky × Nif for all the layers, we serialize the Pox × Poy × Pof MAC outputs into Poy cycles."* But actually, if

$$N \times Poy < \mathrm{Min}(Nkx \times Nky \times Nif)$$

where Nkx, Nky, Nif are the shape of filters, then there can be a N times folded serialization.

So in our final design, according to our design variables, where N can be equal to Pof, only one MUX is used to greatly save the use of following calculation units.

B. BN & ReLU

According to the paper, ReLU is applied after batch normalization. This is also the original proposition of BN[1]. However, in recent study, this may not be promising compared to applying ReLU before BN.

This question is discussed in Adrian Rosebrock's *Deep Learning for Computer Vision with Python*[3]. In Adrian's view, using BN ahead of ReLU does not make sense.

Under the original setting, a BN layer is normalizing the features coming out of a CONV layer, causing nearly half of the features to be negative due to normalization. Then these negative features will be clamped by a nonlinear activation such as ReLU under our design. That is to say, no matter what comes from CONV layer, even all of the outputs are positive, there are always nearly half of total features be clamped to zero.

Instead, if we place the BN after ReLU, we normalize the positive features without statistically biasing them with features that would have otherwise not make it to the next CONV layer.

What's more funny, according to François Chollet:

"I can guarantee that recent code written by Christian [from the BN paper] applies ReLU before BN."

According to some posts online, they also find out using ReLU before BN can yield a better result.

Of course, there are other opinions supporting using BN first. In this way, the parameters of CNN, like bias, can be merged into BN, resulting in a faster inference. But from my

point of view, I prefer a better logic to "tricks" to speed up inference.

Thus, in our reproduction, we apply ReLU before BN.

$$
\begin{aligned}
x' &= x + b \\
x'' &= \gamma \cdot \frac{\mathrm{ReLU}(x') - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} + \beta \\
\Rightarrow x'' &= \frac{\gamma}{\sqrt{\mathrm{Var}[x] + \epsilon}} \cdot \mathrm{ReLU}(x) + (\beta - \gamma \cdot \frac{\mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}}) \\
x'' &= K \cdot \mathrm{ReLU}(x) + B, \\
\text{where } K &= \frac{\gamma}{\sqrt{\mathrm{Var}[x] + \epsilon}},\ B = \beta - \gamma \cdot \frac{\mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}}
\end{aligned}
$$

REFERENCES

[1] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. Mar. 2, 2015. DOI: 10.48550/arXiv.1502.03167. arXiv: 1502.03167 [cs]. URL: http://arxiv.org/abs/1502.03167 (visited on 09/23/2022).

[2] Yufei Ma et al. "Optimizing the Convolution Operation to Accelerate Deep Neural Networks on FPGA". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26.7 (July 2018), pp. 1354–1367. ISSN: 1557-9999. DOI: 10.1109/TVLSI.2018.2815603.

[3] Adrian Rosebrock. *Deep Learning for Computer Vision with Python*. September 2017.