Project Proposal: A FPGA accelerator for YOLO CNN based on weight quantization and data flow optimization

Botong Xiao bx2197, Haoran Jing hj2588, Terry Zhang tz2477, Yunran Zhou yz3985

***Abstract*—Real-time object detection requires high throughput and power efficiency. However, many convolutional neural networks (CNNs) have frequency access to off-chip memory which causes slow processing and undesired power dissipation. In this project, we want to implement a streaming hardware accelerator with a YOLO(You-Only-Look-One) CNN. In addition, the parameters of the CNN will be quantized using binary weight and low-bit activation. Quantization would allow us to store the whole CNN in the on-chip block DRAM. Moreover, the hardware implementation of the CNN will be fully pipelined to improve hardware utilization and reusability of intermediate data. In all, the goal of this project is to eliminate off-chip memory access, improving hardware utilization to better throughput and energy efficiency.**

I. Introduction

DEEP learnging has been the most prevalent method in various task in computer vision due to the support of powerful computation devices such as GPU. Among the state-of-the-art methods, YOLO and the subsequent Sim-YOLO, YOLO-V2 demonstrate the most promising trade-off between speed and accuracy. While GPU is widely used for the training and inference of deep learning algorithms such as YOLO, recent researches have proved its inefficiency in optimization such as the the quantization of weight/activation and data access schedule. For instance, [2] has demonstrated that the training and inference of CNNs can be quantized to a very low-bit precision with insignificant loss in accuracy. This quantization enables a fast , memory efficient, and power efficient FPGA accelerator.

Based on this, [1] proposed a optimized data path to reduce the frequency of off-chip memory access. In this project, we target to replicate the design of [1]. In the following sections, the hardware and software are explained in detail. Milestones of this project is listed in the last section.

II. Specification

Fig.1 presents the overall architecture of the whole FPGA implementation. The data will be input from peripherals such as the camera through PCIE. Then the data will directly be sent to the DRAM through the YOLO DMA. The accelerator will fetch the input image data from the DRAM and perform the computation and then send the detection result back to the DRAM. Eventually, the detection result will output from the DRAM and be sent back to other peripherals such as the monitor through PCIE. The main design idea and details will be illustrated in hardware specification and software specification as follows.

A. Hardware Specification

The YOLO accelerator mainly consists of a controller, convolution layers, and buffers between each layer. Each convolution layer consists of three layers performing convolution computation, batch normalization, and max pooling, respectively.

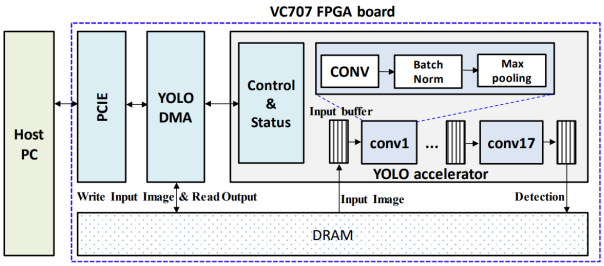

Fig. 1: The Streaming Architecture

The overall proposed structure of the YOLO accelerator is shown in Fig.2. The data input from the previous layer will first be stored in a circular buffer. The circular buffer includes four lines of SRAM. Three of them are the partial inputs from the previous layer and will be divided into sliding cubes and performed 3×3 convolution with the 3×3 kernel. Another additional line is to enable the overlapping of the computation of the current layer and the previous layer. The partial output result of convolution will be sent to the adder tree which consists of two stages of ternary adders. The partial results from adders will be stored in the line buffers, and be sent to perform batch normalization and max-pooling when the computation is completed. Eventually, the final results will be transferred to the next layer and sent back to the DRAM after output from the last convolution layer.

B. Software Specification

For the software part, our initial thought of the program will roughly mainly focus on 3 parts: I/O interface, accelerator and user interface. At the I/O interface part, the software needs to make the connection to the hardware, which means there should be an interface for the camera for object detecting, an interface for video output, making those devices communicate and transfer data. The second part is the accelerator, containing

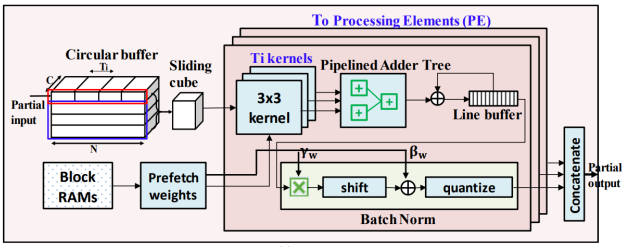

Fig. 2: FPGA implementation of convolutional layers

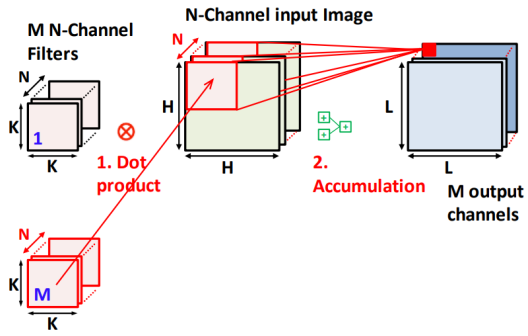

Fig. 3: Operation of convolutional layers

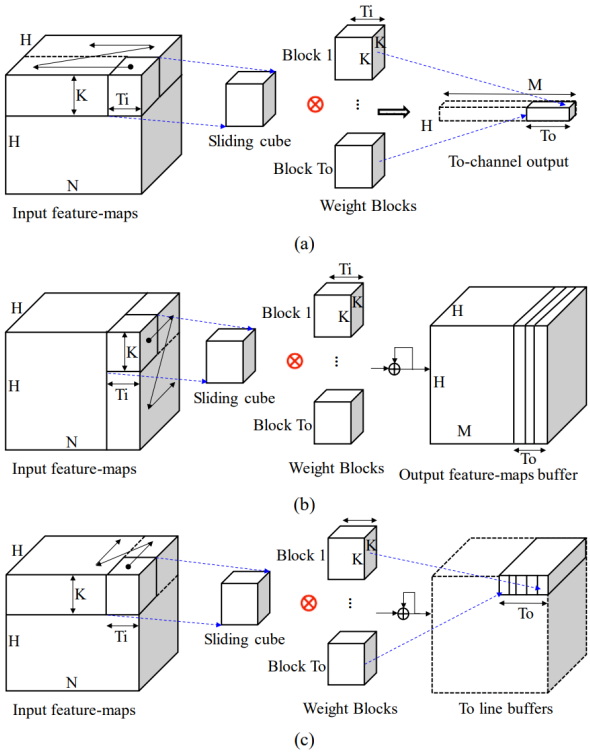

Fig. 4: Different streaming schedule (a) No weight reuse (b) Full weight reuse (c) Line based weight reuse

the convolution layer and buffer which has mentioned above. The work for software is to create those buffers from RAM and also for the algorithm of the accelerator algorithm. The basic principle of the CNN network is shown below at left:

For the CNN network, weight must be used for learning, which could cause a lot of computational resources. Our plan is to calculate the weight off-board, then add the weight to the FPGA. Due to limited RAM resources, those algorithms need optimization in order to make room for buffer. Our plan is to use the method from [1] to reduce the weight re-use, the abstract method from [1] is shown above at right. After optimization, we should be able to put the buffer and algorithm onto the FPGA RAM. And the last part is the user interface. Because it is a project for object detection, it should have a direct image or video output to show the result of the object detection on a screen. Users can choose to activate and deactivate the function, where the object detected should be highlighted and labeled.

III. Milestones

TABLE I: Milestones

Fully understand the CNN algorithm	Mar. 4
Design the RTL implementation for the accelerator	Apr. 22
Verify the accelerator RTL implementation functionality	Apr. 29
Develop the software to connect the accelerator with peripherals	May. 6
Perform verification for the whole design	May. 13

References

[1] D. T. Nguyen, T. N. Nguyen, H. Kim and H. Lee, "A High-Throughput and Power-Efficient FPGA Implementation of YOLO CNN for Object Detection," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 27, no. 8, pp. 1861-1873, Aug. 2019, doi: 10.1109/TVLSI.2019.2905242.

[2] D. T. Nguyen, H. Kim, H. -J. Lee and I. -J. Chang, "An Approximate Memory Architecture for a Reduction of Refresh Power Consumption in Deep Learning Applications," 2018 IEEE International Symposium on Circuits and Systems (ISCAS), 2018, pp. 1-5, doi: 10.1109/ISCAS.2018.8351021.