A 0.57-GOPS/DSP Object Detection PIM Accelerator on FPGA

Bo Jiao, Jinshan Zhang, Yuanyuan Xie, Shunli Wang, Haozhe Zhu, Xiaoyang Kang, Zhiyan Dong, Lihua Zhang and Chixiao Chen
Shanghai Engineering Research Center of AI & Robotics, Fudan University, Shanghai, PR China
Email:{cxchen,xiaoyang_kang}@fudan.edu.cn

ABSTRACT

The paper presents an object detection accelerator featuring a processing-in-memory (PIM) architecture on FPGAs. PIM architectures are well known for their energy efficiency and avoidance of the memory wall. In the accelerator, a PIM unit is developed using BRAM and LUT based counters, which also helps to improve the DSP performance density. The overall architecture consists of 64 PIM units and three memory buffers to store inter-layer results. A shrunk and quantized Tiny-YOLO network is mapped to the PIM accelerator, where DRAM access is fully eliminated during inference. The design achieves a throughput of 201.6 GOPs at 100MHz clock rate and correspondingly, a performance density of 0.57 GOPS/DSP.

1 INTRODUCTION

Recent advances in artificial intelligence have driven many ASIC and FPGA implementations, among which tiny machine learning (tinyML) on terminal devices draws much attention. In terms of energy efficiency and memory wall issues, PIM architectures overweight traditional Von-Neumann and coarse-grained reconfigurable array (CGRA) architectures [1], more suitable for tinyML applications. However, PIM accelerators normally require particular technology of emerging memory devices (ReRAM, MRAM, PCRAM etc.) and custom analog computing circuits. In other words, PIM implementations are regarded as infeasible on FPGAs.

In this paper, we propose an FPGA based PIM tinyML accelerator, where block RAMs (BRAMs) are utilized as PIM memories and look-up-table(LUT)-based custom circuits are generated for PIM computing. The memory wall is mitigated because both inter-layer activations and weights can be fully stuffed in the accelerator. Moreover, DSP blocks, as the most constrained FPGA resource in data-intensive applications, are not involved in tensor computing circuits [2] here. A co-designed object detection algorithm is deployed on the accelerator to verify the proposed accelerator. The scheme achieves performance density of 0.57GOPS/DSP, 3.4x better than traditional CGRA implementation on FPGAs [3].

2 FPGA BASED PIM UNIT

PIM architectures and tinyML implementations desire low bit-width quantized networks. They reduce both memory space and computing power, and they are capable of good accuracy. Learned step-size quantization (LSQ) in [4] achieves almost the same accuracy as floating point on ResNet-18, but only adopts 3-bit quantized activations and weights. Figure 1(a) illustrates a single layer operation of the LSQ algorithm. Both weights and activations are quantized into 3 bits by scaling factors of s_w and s_A, respectively. A tensor computing block completes the convolution or full connection operation afterward. The output of the tensor block is processed by rescaling, normalization, and ReLU, which can be merged into one operator, known as RBR here. The RBR results are quantized as the input activations of the next layer.

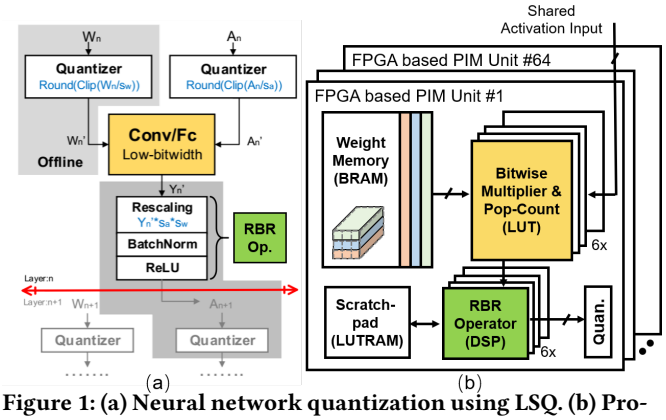

Figure 1: (a) Neural network quantization using LSQ. (b) Proposed PIM unit on FPGA.

Figure 1(b) demonstrates a hardware PIM unit corresponding to 1(a). In each unit, network weights are stored into a 72-Kb BRAM. The BRAM is configured as a 144bx512 one. For 3x3 kernels, each 144 bits contains 48 3-bit weights. To accommodate row-stationary data flow, the 48 weights are assigned to 3 consecutive weights in a row from 16 input channels. The bitwise multiplier performs a logical AND between 1-bit of activation and 1-bit of weight. A pop-counter, implemented with LUTs, accumulates all 1-bit products, and the overall 3-bit results are shift added through a DSP unit. Intermediate partial sums are stored in a local scratchpad, implemented with LUTRAM cells. Another DSP unit completes the RBR operations.

There are 64 PIM units in the proposed accelerator. All PIM units share one 16-bit input activation and perform 192 1bx3b multiplications simultaneously. The partial sums are accumulated according to the data flow configuration.

3 ACCELERATOR ARCHITECTURE

The overall accelerator architecture facilitates the PIM design on an FPGA SoC platform, as depicted in Fig. 2. The key strategy of the tinyML accelerator is to avoid off-chip DRAM data access during inference, thus completely eliminating the memory wall. By adopting low-bit-width quantization, the overall weight memory is reduced to 0.7MB and can be fully stored into the 64 BRAMs. There are three 876Kb RAMs assigned as activation buffers, from which PIM units load data and to which PIM units store inter-layer

ASPDAC '21, Jan. 18–21, 2021, Tokyo, Japan

ACM ISBN 978-1-4503-7999-1/21/01…$15.00
https://doi.org/10.1145/3394885.3431659

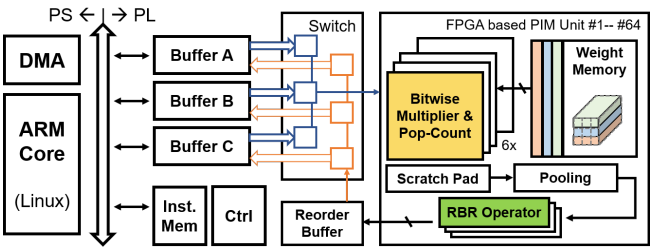

Figure 2: Overall PIM accelerator architecture

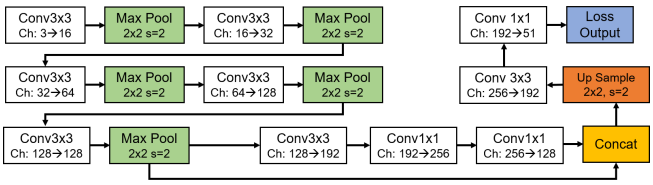

Figure 3: Deployed tiny-YOLO network

results. Therefore, there is no intermediate data accessed from/to the off-chip DRAM.

The accelerator is equipped on an FPGA SoC platform, where the programmable logic gates (PL) are controlled by an on-chip Linux-operating ARM core. All memories can be accessed by the core and the DMA via the AXI bus. A switch box connects the PIM units and the activation buffers according to the source and destination configuration. In each PIM unit, an additional pooling block is inserted between the scratchpad and the RBR block. It can reduce the activity ratio of the RBR block in those layers followed by a max-pooling layer. The accelerator also contains a reorder buffer, which converts the data order of different data flows into a sequential order required by the next layer.

4 EXPERIMENTAL RESULTS

To verify the design, we deployed a shrunk tiny-YOLO network [5] on the proposed accelerator to demonstrate a real-time object detection task. Figure 3 details the deployed network. The network includes seven 3x3 convolutional layers, three 1x1 layers, five max-pooling down-sampling layers, and one up-sampling layer. Among these layers, the first and last layer are not quantized and implemented on the ARM core, and all rest layers are on the accelerator.

The accelerator is implemented on an Ultra96v2 FPGA board, and described in Verilog RTL, rather than high-level synthesis. Thanks to the PYNQ framework, the accelerator can be easily controlled with a Jupyter notebook. Figure 4 illustrates the hardware and software environment while the accelerator is operating. A green bounding box can be drawn on the input image according to the accelerator results. A red box, indicated as ground truth, is also drawn for intersection over union (IoU) analysis. Table 1 shows the usage of various hardware resources. The PIM design almost exhausts LUT, BRAM and DSP resources to maximize the performance density. Table 2 shows the overall operation details. It is found that the peak throughput and performance density are obtained during the first two layers, which is 201.6 GOPS and 0.57GOPS/DSP respectively at a clock rate of 100MHz. The total processing time is 10.2ms, namely achieving a frame rate of 91.7fps.

Figure 4: Accelerator software/hardware deployment.

Table 1: Resource Utilization

Resource	LUT	LUTRAM	FF	BRAM	DSP
Utilization	69.9k	9.3k	43.6k	210.5	353
Per Centerage	99.1%	32.3%	30.9%	97.5%	98.1%

Table 2: Measured Latency Breakdown for YOLOv3-Tiny

Layer	Kernel Size	Input Size $C \times W \times H$	Output Size $C \times W \times H$	#MOp	Latency (ms)
CONV2	3 s1	16×256×144	32×256×144	339.74	1.68
CONV3	3 s1	32×128×72	64×128×72	339.74	1.68
CONV4	3 s1	64×64×36	128×64×36	339.74	1.71
CONV5_1	3 s1	128×32×18	128×32×18	169.87	0.88
CONV5_2	3 s1	128×32×18	128×32×18	169.87	0.88
CONV6	3 s1	128×16×9	192×16×9	63.70	0.35
CONV7	1 s1	192×16×9	256×16×9	14.16	0.21
CONV8	1 s1	256×16×9	128×16×9	9.44	0.14
CONV9	3 s1	256×32×18	192×32×18	509.61	2.70

5 CONCLUSION

This paper presents an object detection PIM accelerator on Ultra96v2 FPGA. The accelerator features FPGA based PIM implementation. Together with multiple activation buffers, the accelerator fully eliminates DRAM access, i.e. the memory wall, during inference. A quantized and shrunk tiny-YOLO network is deployed on the accelerator to verify the effectiveness of the design.

ACKNOWLEDGEMENT

This work was supported by The National Key Research and Development Program of China under Grants 2019YFB2205000, Shanghai Rising-Star Program under Grants 20QA1407300, Shanghai Sailing Program under Grants 18YF1402300, and Science and Technology Commission of Shanghai Municipality, under Grants No. 19511132000.

REFERENCES

[1] Y. Cai, et al., "Training low bitwidth convolutional neural network on RRAM," *Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 117-122, 2018.
[2] D. Wang, et al., "ABM-SpConv: A Novel Approach to FPGA-Based Acceleration of Convolutional Neural Network Inference," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2019.
[3] H. Zhu, Y. Wang, and C.-J. R. Shi, "Tanji: A General-Purpose Neural Network Accelerator with a Unified Crossbar Architecture," in *IEEE Design & Test*, vol. 37, no. 1, pp. 56–63, Feb. 2020.
[4] S.K. Esser, et al., "Learned Step Size Quantization," in *International Conference on Learning Representations 2020 (ICLR)*, 2020.
[5] J. Redmon, and Ali Farhadi. "YOLOv3: An Incremental Improvement," [Online]Available: https://arxiv.org/abs/1804.02767, 2018.