Hardware Acceleration of Fully Quantized BERT for Efficient Natural Language Processing

Zejian Liu[1,2], Gang Li[1], Jian Cheng[1,2]
[1]National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences
[2]School of Future Technology, University of Chinese Academy of Sciences
Beijing, China
liuzejian2018@ia.ac.cn, {gang.li, jcheng}@nlpr.ia.ac.cn

***Abstract*—BERT is the most recent Transformer-based model that achieves state-of-the-art performance in various NLP tasks. In this paper, we investigate the hardware acceleration of BERT on FPGA for edge computing. To tackle the issue of huge computational complexity and memory footprint, we propose to fully quantize the BERT (FQ-BERT), including weights, activations, softmax, layer normalization, and all the intermediate results. Experiments demonstrate that the FQ-BERT can achieve 7.94×compression for weights with negligible performance loss. We then propose an accelerator tailored for the FQ-BERT and evaluate on Xilinx ZCU102 and ZCU111 FPGA. It can achieve a performance-per-watt of 3.18 fps/W, which is 28.91× and 12.72× over Intel(R) Core(TM) i7-8700 CPU and NVIDIA K80 GPU, respectively.**

I. Introduction

In recent years, transformer-based models [1] have been ubiquitous in various natural language processing (NLP) applications, including machine translation [2], question answering and document analysis [3]. Compared with traditional recurrent neural network (RNN) and long short-term memory (LSTM) [4], transformer-based model uses multi-head self-attention mechanism to encode and decode the information of input sentence to achieve better representative ability, and the Bidirectional Encoder Representations from Transformers (BERT) [5] is the state-of-the-art transformer-based model.

The architecture of BERT is depicted in Figure 1. Different from the original transformer, BERT only consists of stacked encoder layers as it focuses on learning language representations, which is similar to convolutional neural network for computer vision. The building block of self-attention is essentially matrix-matrix multiplication, which is suitable for massive parallel computing, therefore high-performance GPUs are preferred to accelerate the training and inference in the cloud. However, the vast computational complexity (>20GFLOPs) and memory requirement (>320MB floating-point parameters) of BERT poses a significant challenge for resource-constraint platforms, which hinders the applications on edge devices.

In this paper, we investigate the acceleration of BERT on resource-constraint FPGAs through algorithm/hardware co-design. Specifically, to reduce memory footprint, we adopt quantization to compress the original model. Different from Q8BERT [6] and Q-BERT [7], which only quantize part of parameters, we propose a fully quantized BERT (FQ-BERT). FQ-BERT represents weights, activations, scale factors, softmax, layer normalization, and all the other intermediate results with integer or fixed-point format. Then we present an accelerator tailored for the proposed FQ-BERT and evaluate on two separate FPGAs. Experiments demonstrate that the proposed accelerator outperforms CPU and GPU by a large margin in terms of performance-per-watt with less than 1% accuracy degradation on SST-2 dataset. The main contributions of this paper can be summarized as follows:

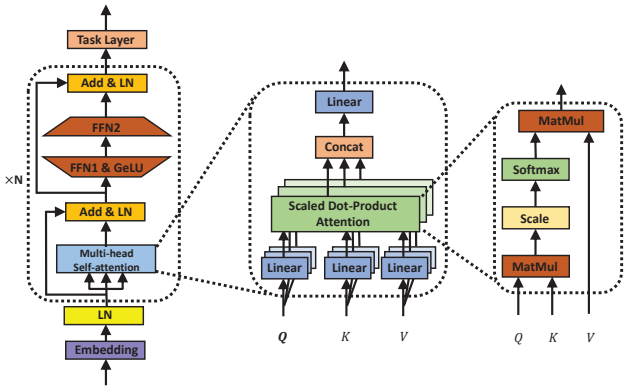


Fig. 1. Left: the overall BERT architecture; middle: the architecture of multi-head self-attention module; right: the architecture of scaled dot-product attention layer.

- We propose an efficient FPGA-based BERT accelerator via algorithm/hardware co-design. On the algorithm side, we propose a hardware-friendly FQ-BERT that represents all parameters and intermediate results with integer or fixed-point data type. On the hardware side, to support different operations during inference, we present an accelerator with dot-product based PE and bit-level reconfigurable multiplier for 8/4-bit and 8/8-bit multiplication at run-time.

- We evaluate 8/4-bit FQ-BERT on SST-2 and MNLI dataset, which achieve a compression ratio of 7.94×, with 0.7% and 3.5% accuracy drop respectively. Then we implement accelerator on Xilinx ZCU102 and ZCU111 MPSoC. Experiments demonstrate that our accelerator can achieve a performance-per-watt of 3.18 fps/W, which is 28.91× and 12.72× over Intel(R) Core(TM) i7-8700 CPU and NVIDIA K80 GPU, respectively.

978-3-9819263-5-4/DATE21/©2021 EDAA

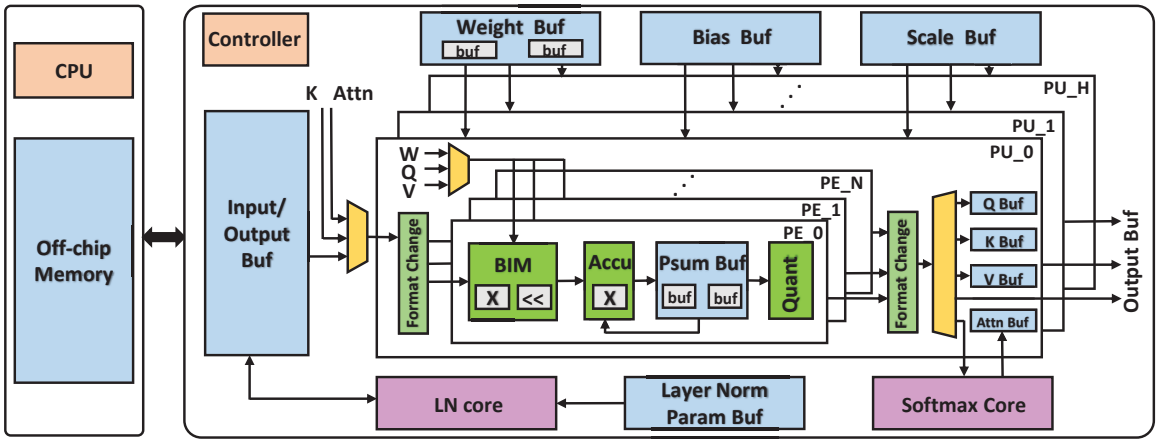


Fig. 2. The overall architecture of the proposed accelerator for fully quantized BERT.

II. Fully Quantized BERT

In this section, we introduce the methods and tricks that we used for quantizing parameters (weights, activations, biases and others) of the original BERT to obtain FQ-BERT.

A. Weights and Activations

To reduce the computational complexity and memory footprint, we adopt quantization to compress the BERT. In this work, we use symmetric linear quantization strategy for both weights and activations, because symmetric quantization is more hardware friendly for the lack of zero-point. Specifically, for each element x of an input tensor, the k-bit quantization function is:

$$\begin{aligned} x_c &= clamp(x, MIN, MAX) \\ s &= scale(x_c, k) \\ x_I &= \lfloor x_c \times s \rceil \\ x_q &= x_I / s \\ clamp(x, a, b) &= min(max(x, a), b) \end{aligned} \tag{1}$$

Figure 3 shows the quantization bitwidth of weights and corresponding accuracy on two document clssification datasets. It can be seen that when the bitwidth of weights is lower than 4, the classification accuracy drops dramatically, which indicates that 4-bit quantization for weights is a reasonable trade-off to maintain accuracy and considerable compression ratio. MIN and MAX are clip thresholds, which need to be carefully tuned during training. Its impact on performance is also depicted in Figure 3. Note that for symmetric quantization, $MIN = -MAX$. s is a scaling factor, which is determined by the value of weights or activations and the quantization bitwidth k. For weights, the scaling factor is calculated according to:

$$s_w = scale(W, k) = \frac{2^{k-1} - 1}{max(|W|)} \tag{2}$$

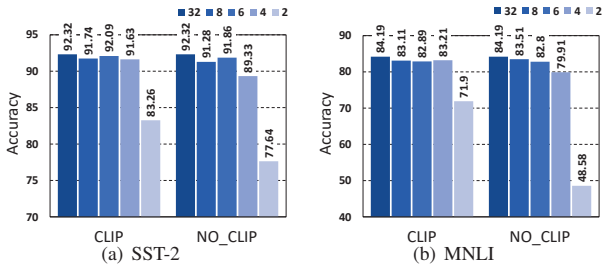


Fig. 3. Impact of different quantization bitwidth of weights on accuracy.

We use exponential moving average (EMA) to collect the statistics of activations to determine scaling factor during inference:

$$s_a = scale(A, k) = \frac{2^{k-1} - 1}{EMA(max(|A|))} \tag{3}$$

B. Biases and Others

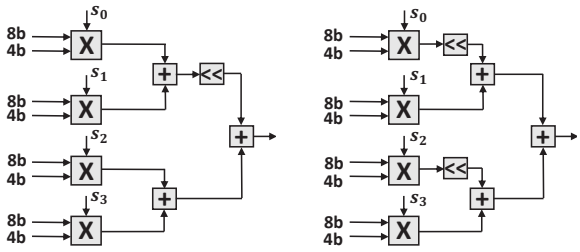
Different from previous work that only weights and activations are quantized, our goal is to obtain a hardware-friendly BERT that can facilitate the deployment on hardware and improve the utilization of computing units. To this end, we further quantize all the biases to 32-bit integers with the help of weights' and activations' scaling factors:

$$\begin{aligned} bias_q &= \lfloor bias \times s_{bias} \rceil / s_{bias} \\ s_{bias} &= s_a \times s_w \end{aligned} \tag{4}$$

As weights, activations and biases are all quantized, the calculation of the output can be rewritten as:

$$\begin{aligned} y_I &= \lfloor y \times s_y \rceil = (\sum a_I \times w_I + b_I) \times s_f \\ s_f &= \frac{s_y}{s_a \times s_w} \end{aligned} \tag{5}$$

where s_a, s_w and s_y are quantized to 8-bit values, and s_f is a 32-bit integer. In addition, we also quantize the numerator and output of softmax module, and parameters of layer normalization to 8-bit fixed-point values.



(a) Type A (b) Type B

Fig. 4. Two type of BIM. $M = 4$.

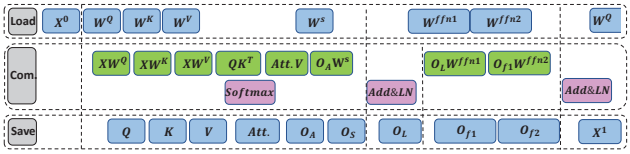


Fig. 5. The dataflow of proposed accelerator.

III. Accelerator

A. System Overview

The overall architecture of our proposed accelerator is shown in Figure 2. It can be divided into two parts: the software program on CPU and the hardware accelerator on FPGA.

As shown in Figure 1, the whole model consists of embedding layers, encoder layers and task-specific layers. Different from encoder layers, the computation cost of other layers is small, but the memory cost is large, so these computations are completed on CPU, and the results are sent to FPGA. The weights are stored in off-chip memory and sent to FPGA through AXI4 interface.

On the FPGA side, the accelerator consists of Processing Units (PUs), Layer Normalization (LN) Core, Softmax Core and On-chip Buffers. Each PU contains multiple Processing Elements (PEs), which is used for variable bitwidth matrix-vector multiplication. The On-chip Buffers include input/output buffer, weight buffer, parameter buffer and intermediate buffer. Specifically, the weight buffer is double buffered to overlap off-chip data transfer. The intermediate buffer is used for storing all the intermediate data, including Q, K, V and attention matrix. The parameter buffer is used for caching data for special computation, such as scaling factors and softmax lookup table values. These data are loaded in the initialization procedure.

B. Computing Unit

PE. As different part of FQ-BERT has different bitwidth, the input bitwidth of multiplication has different combinations, such as 8-bit $\times$ 4-bit for XW^Q and 8-bit $\times$ 8-bit for QK^T. In order to reuse the same module to support different operations, we design a Bit-split Inner-product Module (BIM), which is similar to Bit-fusion [8], as depicted in Figure 4. Each BIM contains $M = 2m$ 8-bit $\times$ 4-bit multipliers, two m-adder tree and several shift-add logic. Each multiplier has a sign signal to indicate the input number is signed or unsigned. The shift-add logic is used for shifting the partial sum when the bitwidth of input is larger than 4 bits. The position of shift-add logic has two choices, as shown in Fig.4, and using shift logic at adder tree's output (Type A) can save more resources, though this need to rearrange the input data.

The output of BIM is a signed partial sum, which will be sent to an accumulator, as shown in Figure 2. The accumulator sums this input and previous data and send the results to the Psum Buf. When the calculation is completed, the results will be sent to quantization module to get final value incorporating with biases and scaling factors. Since the quantization process spends more than one cycle to complete, the Psum Buf is double buffered to ensure the calculation can be pipelined.

Softmax Core. BERT utilizes the softmax function to convert QK^T to attention matrix, the expression of softmax is:

$$Softmax(\boldsymbol{x})_i = \frac{exp(x_i)}{\sum_j exp(x_j)} \tag{6}$$

whele $\boldsymbol{x}$ is a row vector of QK^T matrix. As the exponential function $exp(x_i)$ is very expensive in terms of resource consumption, we use a lookup table instead. However, it is difficult to determine the number of entries of lookup table as the range of exponential function is very large and sensitive to input value. Fortunately, softmax is invariant to subtraction, i.e. subtract a constant for all the elements in the vector will not change the final results. If all the elements subtract the maximum value of the input vector, the output of exponential function will always be limited between 0 and 1. What's more, as we quantize $exp(x_i)$ to 8-bit, only 256 sampling points are needed.

LN Core. As most operations in layer normalization are element-wise multiplications and additions, with some non-linear operations, PE arrays are unsuitable for this computation pattern. We design a coarse-grained 3-stage pipelined SIMD unit to fully utilize its parallelism. The first stage consumes two input vectors with two scaling factors and generates one vector and a mean value. The second stage subtracts the mean value from each element in the input and calculates variance value. The third stage finish the element-wise multiplication.

C. Scheduling

The dataflow of the accelerator is shown in Figure 5. According to different computation units and required weights, the computing process can be divide into several stages. Although the weights have been quantized to 4-bit, loading all the weights that needed per stage is still unpractical. So we further divide each stage into several sub-stages, and only load part of weights that will be used in next sub-stage. Through task-level scheduling, the off-chip transfer can be completely overlapped by computing.

IV. Evaluation

In this section, we evaluate the performance of proposed accelerator. First, we introduce the software and hardware

TABLE I
ACCURACY OF FQ-BERT AND BASELINE BERT.

	w/a	SST-2	MNLI	MNLI-m	Comp. Ratio
BERT	32/32	92.32	84.19	83.97	1×
FQ-BERT	4/8	91.51	81.11	80.36	7.94×

TABLE II
ABLATION STUDY OF QUANTIZATION FOR DIFFERENT PARTS OF BERT.

w/a	scale	softmax	layer norm	accuracy
-	-	-	-	92.32
✓	-	-	-	91.63
✓	✓	-	-	91.28
✓	✓	✓	-	91.86
✓	✓	✓	✓	91.51

environment setup of our experiments. Then, the performance of FQ-BERT is given. Finally, hardware resource utilization, latency and power consumption are provided.

A. Experimental Setup

We implement the FQ-BERT using PyTorch and evaluate on two tasks, SST-2 and MNLI, of GLUE benchmark [9]. To get the quantized model, we first train the original model for 3 epochs with default hyper-parameters. Then we fine-tune the model with quantization function.

The proposed hardware accelerator is built on Xilinx ZCU102 FPGA and ZCU111 MPSoC, and the frequency of FPGA part is set to 214 MHz. We implement the accelerator using High Level Synthesis. Vivado HLS 2019.1 is used for compiling the C++ code to generate synthesizable RTL.

The baseline program runs on Intel(R) Core(TM) i7-8700 CPU @ 3.2GHz and NVIDIA K80 GPU with CUDA 10.1. As we only consider the latency, all the experiments are running with batch size of 1, and the length of sentence is set to 128.

B. Accuracy

The accuracy of FQ-BERT is shown in Table I. For SST-2 task, the accuracy drop is 0.81%, and for MNLI task, the accuracy drop is 3.08% and 3.61%, respectively. The accuracy loss on MNLI is larger than SST-2 as it is a more difficult task. After quantization, the model size is compressed 7.94×. We also investigate the impact of different parts of BERT when they are quantized, the results are shown in Table II. We observe that the accuracy drop is not always increasing when more parameters are quantized, as quantizing softmax can improve the accuracy from 91.28% to 91.86%.

C. Resource Consumption

Table III shows the resource consumption of the proposed accelerator for different configurations. Specifically, the number of PEs N and the number of multipliers in BIM M are examined. We can see that the DSP usage is very high for the targeted FPGA. To demonstrate the scalability of our accelerator, we double the number of total multipliers and implement it on ZCU111, and we get nearly twice the performance improvement.

TABLE III
RESOURCE CONSUMPTION AND LATENCY FOR DIFFERENT NUMBER OF PES AND MULTIPLIERS IN BIM. THE NUMBER OF PUS IS 12.

(N, M)	BRAM18K	DSP48E	FF	LUT	Latency(ms)
ZCU102	1824	2520	548160	274080	-
(8, 16)	838	1751	124433	123157	43.89
(16, 8)	877	1671	151010	154192	45.35
ZCU111	2160	4272	850560	425280	-
(16, 16)	679*	3287	201469	189724	23.79

*Some memory are implemented using URAM, which are not reported

TABLE IV
PERFORMANCE COMPARISON ON CPU, GPU AND FPGA.

	CPU	GPU	ZCU102	ZCU111
Latency(ms)	145.06	27.84	43.89	23.79
Power(W)	65	143	9.8	13.2
fps/W	0.11	0.25	2.32	3.18

D. Performance and Energy Efficiency

We further compare the performance and energy efficiency of our accelerator with CPU and GPU, results are shown in Table IV. Compared with CPU, our accelerator achieves 6.10× and 28.91× improvement in latency and energy efficiency. For GPU, our accelerator achieves 1.17× and 12.72× improvement.

V. CONCLUSION

In this paper, we investigate the acceleration of BERT on FPGAs. To reduce memory footprint, we propose to fully quantize all parameters of BERT, including weights, activations, scale factors, softmax, layer normalization and other intermediate results. Then we present an energy-efficient accelerator and evaluate it on two separate FPGAs. Experiments show that our accelerator outperforms CPU and GPU by factors of 28.91× and 12.72× on performance-per-watt respectively.

VI. ACKNOWLEDGMENT

This work was supported in part by National Natural Science Foundation of China (No.61972396, 61876182, 61906193), National Key Research and Development Program of China (No. 2020AAA0103402), the Strategic Priority Research Program of Chinese Academy of Sciences (No. XDB32050200).

REFERENCES

[1] A. Vaswani *et al.*, "Attention is all you need," in *NeurIPS*, 2017.
[2] Q. Wang *et al.*, "Learning deep transformer models for machine translation," in *ACL*, 2019.
[3] B. van Aken *et al.*, "How does bert answer questions? a layer-wise analysis of transformer representations," in *CIKM*, 2019.
[4] S. Hochreiter *et al.*, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[5] J. Devlin *et al.*, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *NAACL*, 2019.
[6] O. Zafrir *et al.*, "Q8BERT: Quantized 8Bit BERT," in *NIPS workshop*, 2019.
[7] S. Shen *et al.*, "Q-BERT: Hessian Based Ultra Low Precision Quantization of BERT," in *AAAI*, 2019.
[8] H. Sharma *et al.*, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural networks," in *ISCA*, 2018.
[9] A. Wang *et al.*, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," in *ICLR*, 2019.