Accelerating NLP Tasks on FPGA with Compressed BERT and a Hardware-Oriented Early Exit Method

Binjing Li, Siyuan Lu, Keli Xie, and Zhongfeng Wang

School of Electronic Science and Engineering, Nanjing University, P. R. China
Email: {bjli,sylu, kelixie}@smail.nju.edu.cn, {zfwang}@nju.edu.cn

***Abstract*—In recent years, Natural Language Processing (NLP) has gradually become a heated topic in research area, and Transformer-based pretrained models (the most widely used is BERT) have achieved state-of-the-art results in many NLP tasks. But since Transformer-based pretrained models always contain extensive parameters, and consume much time in computing, it is really difficult to employ them on resource-limited embedded platform or mobile devices. To resolve this issue, we utilize the ALBERT model with the improved early exit method (ELBERT) and propose an efficient VLSI architecture for it in an algorithm and hardware co-design way. First of all, by using the quantization and encoder-level parameter sharing techniques, the storage space for saving BERT is reduced from 1208.88 MB to 20.99MB with little accuracy loss, which makes it possible to store all the weights on-chip. Secondly, we present a hardware-friendly design for the improved early-exit method. In contrast to the original ALBERT implementation, the accelerator combining ALBERT model with improved early exit can obtain a speed-up of 4.59x. Thirdly, owing to our efficient hardware design, experiments demonstrate that our FPGA accelerator can achieve a performance-per-watt of 1.29 fps/W, which is 25.97x over NVIDIA 2080 Ti GPU. On the whole, we introduce an efficient accelerator on FPGA with compressed BERT and a hardware-oriented early exit method, which solves the challenge to deploy the BERT to resource-constrained edge platforms with strict latency and memory requirements.**

***Index Terms*—Natural Language Processing (NLP), BERT, Early Exit, Inference Accelerator, Hardware Design**

I. Introduction

In Natural Language Processing (NLP) tasks, large-scaled pre-trained models (like BERT [2], RoBERTa [5] and ERNIE [9]) have outperformed the previous solutions to a large extent. However, a large number of parameters can lead to high computation cost and huge memory requirements, making it difficult to apply these models on the mobile devices or embedding platforms.

In order to acquire the state-of-the-art accuracy in limited hardware resources, model compression methods have been proposed. ALBERT [4] gives a compression method from another perspective. By sharing parameters among different layers, it can reduce the number of parameters greatly but still maintain the prediction accuracy as the BERT model. With its sharing parameter technique, the ALBERT can attain faster prediction speed. From the perspective of reducing computation cost, early exit method can be applied to the ALBERT model. Early exit may enable one-for-all, which implies that after the training process finished, the model can exit at certain layers according to the complexity of inputs.

Nowadays, building ASIC or FPGA design for deep neural network has attracted a lot of attention in related research area. To employ large-scaled models on resource-limited hardware, specific and efficient hardware design is really indispensable. As far as we know, [6] presents one typical hardware acceleration design of BERT on FPGA. The current solution involves time-consuming data transportation from off-chip memory to on-chip memory because of the large amount of parameters in the BERT. Even if pipeline approach is taken to fully utilize the parallelism, the process of data transportation still cannot be avoided. Owing to parameter sharing technique and fully quantize the model, all the parameters can be stored on the chip and the time of data loading from external memory to internal memory will be saved.

In this paper, we put forward an efficient hardware design for ALBERT with the refined early exit method. ALBERT with the refined early exit approach can be divided into three parts, MHA (multi-head attention) and FFN (position-wise feed forward network) and the early exit part. Main contributions of this work can be summarized as follows:

- In order to reduce the burden on hardware storage, our work quantizes the BERT model with the method of symmetric linear quantization and cross-layer parameter sharing. The weights and activation results of MHA are quantized into 8 bits, while the weights and activation results of FFN are quantized into 16 bits. To recover the inference accuracy of the model, we have to do some fine-tune training.
- We present a hardware-friendly design for the refined early exit method. Entropy-based early exit and probability-trend-based early exit are all optimized to reduce the latency as much as possible.
- We propose an efficient hardware architecture for ALBERT combined with the refined early exit method in a algorithm and hardware co-design way. For algorithm part, the storage space for saving BERT is reduced from 1208.88 MB to 20.99 MB with little accuracy loss, which makes it possible to store all weights on-chip. For hardware part, the optimized architecture can achieve a speed-up of 2.98x per layer compared with the implementation on GPU when the batch size is set 1 and max sequence length is 64. in contrast to the original ALBERT implementation, the accelerator

combined ALBERT model with the refined early exit can obtain a speed-up of 4.59x. Owing to our efficient hardware design, experiments demonstrate that our FPGA accelerator can achieve a performance-per-watt of 1.29 fps/W, which is 25.97x over NVIDIA 2080 Ti GPU.

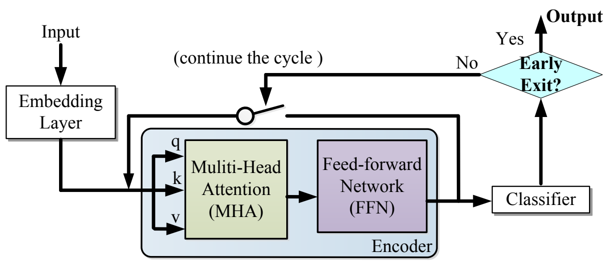

Fig. 1. The architecture of the ELBERT model. Owing to the cross-layer parameter sharing strategy, the output of the encoder will be the next input.

II. Background

A. The Model Architecture of the ELBERT

The ELBERT is composed of an ALBERT [4] model and the early exit unit. The overall architecture of the ELBERT can be described in Fig. 1. Using ALBERT as the backbone, the ELBERT model contains an embedding layer, an encoder layer, and a classifier. The early exit decision is put after each propagation processed by the encoder and the classifier.

B. Quantization of BERT

To achieve the goal of lightweight in BERT, [13] performs quantization-aware training during the fine-tuning phase. The accuracy of the quantized 8-bit network is only 1% lower than that of the float32 network, which achieves 4x compression ratio of the network. In [3], with almost no precision loss, a lightweight approximation method is taken for nonlinear operations in BERT, enabling 2.4-4.0x acceleration. [8] incorporates second-order Heyssen information into a fine-tuned BERT model. The result shows that the network is quantized to only 2 bits, with only 2.3% accuracy loss on NLP datasets. [1] proposes a binary BERT, which initializes a binary BERT network by splitting weights from a half-sized ternary network.

C. Refined Early Exit Method

As in [12] and [10], the first stage of the early exit focuses on confidence of the classifier. Given a probability distribution p_i, the normalized entropy of the current classifier is calculated.

If the entropy is less than a certain threshold δ, the calculation will be terminated. The second stage of early exit traces the historical trend of the classifier output in a time window. If the prediction probability p_i of a certain class varies monotonically, further computations will be skipped.

D. The efficient hardware design of transformer-based pre-trained model

1) Softmax: As listed in [11], further transformation replaces the exponential and logarithmic (exp-log) functions in softmax by some other simple operations including constant multiplication, shift and addition.

2) Layer Normalization (LN): The layer normalization function used in the two ResBlock is:

$$layernorm(i,j) = \frac{G(i,j) - E(G,i)}{\sqrt{var(G,i)+\epsilon}}\gamma_j + \beta_j, \tag{1}$$

where the variable $E(G,i)$ is the mean value of all the elements in the i-th row of matrix G $(s \times d_{model})$.

Instead of computing the $E(G,i)$ and $var(G,i)$ directly in serialization mode, we can first compute the $E(G,i)$ and $E(G*G,i)$ in parallel as stated in [7]. Then we calculate the $var(G,i)$ based on the given results.

III. Efficient Architecture for the ELBERT Model

A. Quantization Method for the ELBERT Model

In our work, we choose the ALBERT architecture as the backbone. The ALBERT model utilizes the method of cross-layer parameter sharing to improve the parameter efficiency.

Besides, we apply the symmetric linear quantization to the ELBERT model as follows:

$$W_q = Q_{m,f}(W) = \lfloor 2^f W_{clip} + \frac{1}{2} \rfloor \cdot 2^{-f}, \tag{2}$$

where the W and W_q represents the float32 weights and quantized weights, respectively. m and q are denoted as the bits of integer and decimal, respectively.

The W_{clip} is the clipping function to restrict the range of W. The value out of range will be clipped to the correct range which can be represented by m integer bits and f decimal bits. Here the W_{clip} is described as follows:

$$W_{clip} = min(|W|, 2^m - 2^{-f}) \cdot sign(W). \tag{3}$$

The weights of the MHA ResBlock are quantized to 8 bits. The others are quantized to 16 bits. To be more specific, the m integer bits are determined by the range of a certain value. Due to the total number of bits are fixed, the f decimal number is calculated by the total number of bits minus the m integer bits. In order to maintain the prediction accuracy, we have to do some finetune training.

In our experiment, after symmetric linear quantization and cross-layer parameter sharing, it is possible to store all the weights on-chip with lower power consumption and no data transportation.

B. The Refined Early Exit Hardware Unit

Before going through the early exit module, the $[CLS]$ position results whose dimension is $[1, d_{model}]$ will be extracted from the outputs of the FFN ResBlock. Then the $[CLS]$ position results will first go through a exit layer which contains two linear sublayers and a tanh activation between the two sublayers. These calculations can reuse the SA hardware.

Algorithm 1: The Computation Flow of the MHA ResBlock (d_{model}=n*64, (j) represents the index of each block whose size is s * 64)

```
if Calculating the MHA ResBlock then
  for i = 1; i ≤ h; i++ do
    for j = 1; j ≤ $d_{model}$/64; j++ do
      $Temp1^{(j,0)} = Q^{(j)}W_{Q_i} + Bias^{(j)}_{Q_i}$;
    end for
    for j = 1; j ≤ $d_{model}$/64; j++ do
      $Temp2^{(j)} = K^{(j)}W_{K_i} + Bias^{(j)}_{K_i}$;
    end for
    for j = 1; j ≤ $d_{model}$/64; j++ do
      for k = 1; k ≤ $d_{model}$/64; k++ do
        $Softmax\ \ input^{(j,k)} = Temp1^{(j,0)} \times Temp2^{(k)^T}$;
      end for
    end for
    $Temp_1 = Softmax(Softmax\ \ input)$;
    for j = 1; j ≤ $d_{model}$/64; j++ do
      $Temp2^{(j)} = V^{(j)}W_{V_i} + Bias^{(j)}_{V_i}$;
    end for
    Reset P to 0;
    for j = 1; j ≤ $d_{model}$/64; j++ do
      for k = 1; k ≤ s/64; k++ do
        $P^{(j)}_i + = Temp1^{(j,k)} \times Temp2^{(k)}$;
      end for
    end for
  end for
  for i = 1; i ≤ h; i++ do
    for j = 1; j ≤ $d_{model}$/64; j++ do
      $G^{(j)_i} = P^{(j)} \times W_{G_i} + Bias^{(j)}_{G_i} + Q^{(j)}_i$;
    end for
  end for
  Output = LayerNorm(G);
end if
```

Algorithm 2: The Computation Flow of the FFN ResBlock (d_{model}=n*64, the superscript represents the index of each block whose size is s * 64)

```
if Calculating the FFN ResBlock then
  for i = 1; i ≤ 4h; i++ do
    for j = 1; j ≤ $d_{model}$/64; j++ do
      $P^{(j)}_i = GELU(X^{(j)}W_{1_i}) + Bias^{(j)}_{1_i}$;
    end for
  end for
  for i = 1; i ≤ h; i++ do
    for j = 1; j ≤ $d_{model}$/64; j++ do
      $G^{(j)}_i = P^{(j)}W_{2_i} + Bias^{(j)}_{2_i} + X^{(j)}_i$;
    end for
  end for
  Output = LayerNorm(G);
end if
```

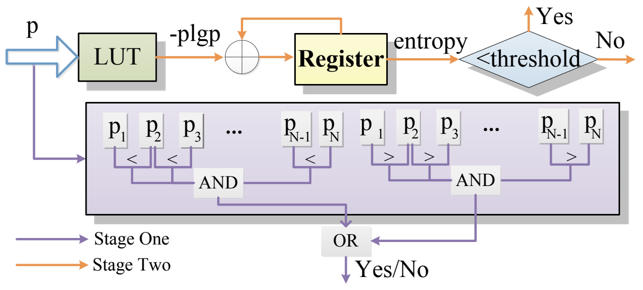

Fig. 2. The hardware module of the refined early exit method.

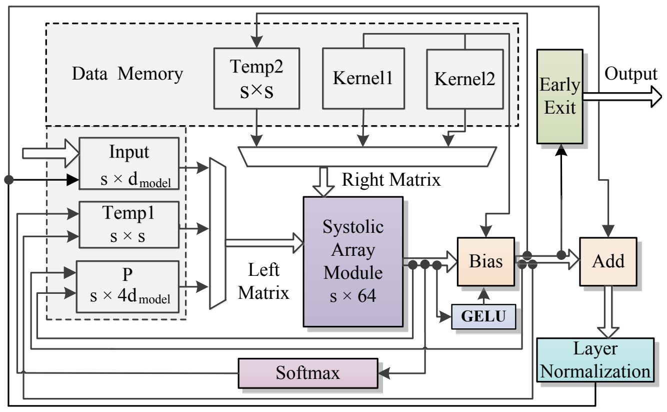

Fig. 3. The whole hardware architecture of the ELBERT model.

The refined early exit method includes two stages. The first stage is to calculate the normalized entropy of the current classifier given a probability distribution p_i. The entropy contains multiplications, logarithmic and summation operations. The multiplications and logarithmic operations are computed according to the lookup table (LUT). If the entropy value is less than δ, we use a termination flag to mark this situation.

C. The Overall Hardware Architecture Design for the Proposed Accelerator

As the batch size is set to 1 in our experiment, the calculation of these two ResBlock is sets of matrix-matrix multiplication. The General Matrix-matrix Multiplications (GEMM) can be done with one and the same SA, the size of which is limited to $s \times 64$.

The whole hardware architecture of the ELBERT model is illustrated in Fig. 3. The input of the MHA ResBlock and the outputs of the two ResBlocks are stored in the $input$ block. The weights of the MHA ReBlock and the FFN ResBlock are stored in the $kernel_1$ and $kernel_2$ block separately in Fig. 3. From the architecture in the picture and the Algorithm, we can see the detailed computation flow clearly.

IV. Experimental Results

From the TABLE I, our quantization method only costs less than 1% accuracy loss on ELBERT, which greatly saves the storage space for saving BERT.

TABLE I
ACCURACY (%) OF FLOAT32 AND QUANTIZED ELBERT MODEL WITH THE REFINED EARLY EXIT THRESHOLD SET TO 0.1

Dataset	SST-2	ag	IMDB
Float32	89.44	93.64	89.61
Quantized	88.99	92.80	89.12
Comparison	0.45	0.84	0.49

All the experiments are running with batch size of 1, and the length of sentence (denoted as s) is set to 64. The proposed architecture is evaluated on Xilinx xcvu13p-fhga2104-3-e FPGA by using the Vivado 2018.2. The Vivado implementation results-show that our design can run up to 200MHz. The total on-chip power is 9.251W (5.940W dynamic power and 3.311W device static power). The utilization report is presented in TABLE II.

To validate the performance of the hardware accelerator, we take SST-2 dataset as an example. When the refined early exit threshold set to 0.4, the average exit layer is 2.96. In contrast to the original ALBERT implementation, the accelerator combining ALBERT model with improved early exit can obtain a speed-up of 4.59x.

TABLE II
UTILIZATION REPORT FOR THE PROPOSED HARDWARE ACCELERATOR AND ITS PRIMARY MODULES

Resource	LUT	FF	BRAM	URAM	DSP
Utilization	412589	216459	196.5	960	321
Available	1728000	3456000	2688	1280	12288
Utilization(%)	23.88	6.26	7.31	75	2.61

The latency of per layer in ELBERT on NVIDIA 2080 Ti GPU is 13.79 ms. While the latency of hardware accelerator per layer in ELBERT is 4.63 ms. The power consumption of the GPU is 61 W. Owing to our efficient hardware design, experiments demonstrate that our FPGA accelerator can achieve a performance-per-watt of 1.29 fps/W, which is 25.97x over NVIDIA 2080 Ti GPU.

V. CONCLUSION

In conclusion, we propose an efficient hardware architecture for the ALBERT model with refined early exit method. We optimize the whole design in an algorithm and hardware co-design way. From the algorithm side, we make use of the parameter sharing technique and symmetric linear quantization method to quantize the BERT model to a large extent. From the hardware side, we design the refined early exit module using few hardware resources and optimize the other modules comprehensively. The results show that in contrast to the original ALBERT implementation, the accelerator combining the ALBERT model with the improved early exit can obtain a speed-up of 4.59x. Owing to our efficient hardware design, experiments demonstrate that our FPGA accelerator can achieve a performance-per-watt of 1.29 fps/W, which is 25.97x over NVIDIA 2080 Ti GPU. Based on compressed BERT and a hardware-oriented early exit method, we provide a novel solution for the design of FPGA accelerator which reaches the goal of employing the BERT to resource-constrained edge platforms with strict latency and memory requirements.

VI. ACKNOWLEDGEMENT

This work was supported in part by the National Natural Science Foundation of China under Grant 62174084, 62104097 and in part by the High-Level Personnel Project of Jiangsu Province under Grant JSSCBS20210034, the Key Research Plan of Jiangsu Province of China under Grant BE2019003-4.

REFERENCES

[1] Haoli Bai, Wei Zhang, Lu Hou, Lifeng Shang, Jing Jin, Xin Jiang, Qun Liu, Michael Lyu, and Irwin King. Binarybert: Pushing the limit of bert quantization. *arXiv preprint arXiv:2012.15701*, 2020.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[3] Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. I-bert: Integer-only bert quantization. In *International conference on machine learning*, pages 5506–5518. PMLR, 2021.

[4] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

[5] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[6] Zejian Liu, Gang Li, and Jian Cheng. Hardware acceleration of fully quantized bert for efficient natural language processing. In *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 513–516, 2021.

[7] Siyuan Lu, Meiqi Wang, Shuang Liang, Jun Lin, and Zhongfeng Wang. Hardware accelerator for multi-head attention and position-wise feed-forward in the transformer. In *2020 IEEE 33rd International System-on-Chip Conference (SOCC)*, pages 84–89, 2020.

[8] Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Q-bert: Hessian based ultra low precision quantization of bert. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8815–8821, 2020.

[9] Yu Sun, Shuohuan Wang, Yukun Li, Shikun Feng, Xuyi Chen, Han Zhang, Xin Tian, Danxiang Zhu, Hao Tian, and Hua Wu. Ernie: Enhanced representation through knowledge integration. *arXiv preprint arXiv:1904.09223*, 2019.

[10] Meiqi Wang, Liulu He, Jun Lin, and Zhongfeng Wang. Rethinking adaptive computing: Building a unified model complexity-reduction framework with adversarial robustness. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–8, 2021.

[11] Meiqi Wang, Siyuan Lu, Danyang Zhu, Jun Lin, and Zhongfeng Wang. A high-speed and low-complexity architecture for softmax function in deep learning. In *2018 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, pages 223–226, 2018.

[12] Keli Xie, Siyuan Lu, Meiqi Wang, and Zhongfeng Wang. Elbert: Fast albert with confidence-window based early exit. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7713–7717, 2021.

[13] Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. Q8bert: Quantized 8bit bert. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition (EMC2-NIPS)*, pages 36–39, 2019.