FTRANS: Energy-Efficient Acceleration of Transformers using FPGA

Bingbing Li\textsuperscript{1}, Santosh Pandey\textsuperscript{2}, Haowen Fang\textsuperscript{3}, Yanjun Lyu\textsuperscript{1}, Ji Li\textsuperscript{4}, Jieyang Chen\textsuperscript{5}, Mimi Xie\textsuperscript{6}, Lipeng Wan\textsuperscript{1}, Hang Liu\textsuperscript{2} and Caiwen Ding\textsuperscript{1}

\textsuperscript{1}University of Connecticut \textsuperscript{2}Stevens Institute of Technology \textsuperscript{3}Syracuse University
\textsuperscript{4}Microsoft Corporation \textsuperscript{5}Oak Ridge National Laboratory \textsuperscript{6}University of Texas at San Antonio
\textsuperscript{1}\{bingbing.li, l.yu.yanjun, caiciwen.ding\}@uconn.edu \textsuperscript{2}\{spandey, Hang.liu\}@stevens.edu \textsuperscript{3}fang02@syru.edu
\textsuperscript{4}\texttt{changzhouliji@gmail.com} \textsuperscript{5}\texttt{[chenj3, wanl]@ornl.gov} \textsuperscript{6}mimi.xie@utsa.edu

ABSTRACT

In natural language processing (NLP), the “Transformer” architecture was proposed as the first transduction model relying entirely on self-attention mechanisms without using sequence-aligned recurrent neural networks (RNNs) or convolution, and it achieved significant improvements for sequence to sequence tasks. The introduced intensive computation and storage of these pre-trained language representations has impeded their popularity into computation and memory constrained devices. The field-programmable gate array (FPGA) is widely used to accelerate deep learning algorithms for its high parallelism and low latency. However, the trained models are still too large to accommodate to an FPGA fabric. In this paper, we propose an efficient acceleration framework, FTRANS, for transformer-based large scale language representations. Our framework includes enhanced block-circulant matrix (BCM)-based weight representation to enable model compression on large-scale language representations at the algorithm level with few accuracy degradation, and an acceleration design at the architecture level. Experimental results show that our proposed framework significantly reduce the model size of NLP models by up to 16 times. Our FPGA design achieves 27.07× and 81× improvement in performance and energy efficiency compared to CPU, and up to 8.80× improvement in energy efficiency compared to GPU.

ACM Reference Format:

Bingbing Li\textsuperscript{1}, Santosh Pandey\textsuperscript{2}, Haowen Fang\textsuperscript{3}, Yanjun Lyu\textsuperscript{1}, Ji Li\textsuperscript{4}, Jieyang Chen\textsuperscript{5}, Mimi Xie\textsuperscript{6}, Lipeng Wan\textsuperscript{1}, Hang Liu\textsuperscript{2} and Caiwen Ding\textsuperscript{1}. 2020. FTRANS: Energy-Efficient Acceleration of Transformers using FPGA. In ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED ’20), August 10–12, 2020, Boston, MA, USA. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3370748.3406567

1 INTRODUCTION

RNN and its variant Long Short-Term Memory (LSTM) unit [6] and Gated Recurrent unit (GRU) [3] used to dominate in sequence modeling, language modeling and machine translation, etc. However, they in general lack efficiency in transmitting global information, due to the bottleneck in the memory (hidden state) and complicated bypassing logic (additive and derivative branches) where long range information is passed. In addition, the inherently sequential nature precludes parallelization within training examples through backpropagation, which is critical at longer sequence lengths [9].

To overcome the shortcomings in RNNs, the “Transformer” architecture was proposed as the first transduction model relying entirely on self-attention mechanisms without using sequence-aligned RNNs or convolution. It achieved notable improvements for sequence to sequence tasks [18]. The breakthroughs and developments of new models have accelerated at an unprecedented pace since the attention mechanisms have become the mainstream in NLP domain with the invention of Transformer. Many transformer-based NLP language models like BERT [4] and RoBERTa [10] introduced pretraining procedures to the transformer architecture and achieved record-breaking results on major NLP tasks, including question answering, sentiment analysis, and language inference.

Nevertheless, the introduced intensive computation and power footprint of these pre-trained language representations has impeded their popularity into computation and energy constrained edge devices. Moreover, despite of the rapid advancement achieved by the recent transformer-based NLP models, there is a serious lack of studies on compressing these models for embedded and internet-of-things (IoT) devices.

In this paper, we propose an energy-efficient acceleration framework, FTRANS, for transformer-based large scale language representations using FPGA. FTRANS is comprised of an enhanced BCM-based method enabling model compression on language representations at the algorithm level, and an acceleration design at the architecture level. Our contributions are summarized as follows:

- **Enhanced BCM-based model compression for Transformer.** We address the accuracy degradation caused by traditional BCM compression, and propose an enhanced BCM-based compression to reduce the footprint of weights in Transformer. With small accuracy loss, FTRANS achieves up to 16 times compression ratio.
- **Holistic optimization for Transformers on FPGA.** Given the large size and complex data flow of transformer-based models, even with model compression, we still need to schedule the computation resources carefully to optimize latency and throughput. We propose a two stage optimization approach to mitigate the resource constraints and achieve high throughput.
- **Low hardware footprint and low power (energy) consumption.** We propose an FPGA architecture design to support the model compression technique and we develop a design automation and optimization technique. Overall, the proposed FTRANS achieves the lowest hardware cost and energy consumption in...
implementing Transformer and RoBERTa compared to CPU and GPU references.

Experimental results show that our proposed framework significantly reduce the size of NLP models by up to 16 times. Our FPGA design achieves 27.07× and 81× improvement in performance and energy efficiency compared to CPU. The power consumption of GPU is up to 5.01× compared to that of FPGA, and we achieve up to 8.80× improvement in energy efficiency compared to GPU.

2 RELATED WORK
Attention mechanisms have become an integral part of compelling sequence modeling and transduction models in various tasks [9]. Evidence of NLP community moving towards attention-based models can be found by more attention-based neural networks developed by companies like Amazon [8], Facebook [16], and Salesforce [2].

The novel approach of Transformer is the first model to eliminate recurrence completely with self-attention to handle the dependencies between input and output. BERT [4] and RoBERTa [10] extend Transformer’s capacity from a sequence to sequence model to a general language model by introducing the pretraining procedure, and achieved state-of-the-art results on major NLP benchmarks. Although RNNs and Convolutional Neural Networks (CNNs) are being replaced by Transformer-based models in NLP community, there are only a few works that accelerate Transformers and focus on reducing the energy and power footprint, e.g., a case study of Transformer is presented in [1] using one of the cutting-edge FPGA boards. However, it is noteworthy that [1] targets at a special use case, e.g., a case study of Transformer is presented in [1] using one of the cutting-edge FPGA boards. However, it is noteworthy that [1] targets at a special use case, e.g., a case study of Transformer is presented in [1] using one of the cutting-edge FPGA boards. However, it is noteworthy that [1] targets at a special use case, e.g., a case study of Transformer is presented in [1] using one of the cutting-edge FPGA boards. However, it is noteworthy that [1] targets at a special use case, e.g., a case study of Transformer is presented in [1] using one of the cutting-edge FPGA boards.

3 TRANSFORMER WORKLOAD ANALYSIS
3.1 Attention
The attention function can be described as mapping a query q and a set of keys k and values v pairs to an output o as shown in Figure 2 (a), named scaled dot-product attention, or single head attention.

\[
O_{att} = \text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)
\]  

3.2 Multi-head Attention
Multi-head attention are then concatenated as multi-head attention, as shown in Figure 2 (b). MultiHead (Q, K, V) = Concat (Head1, \cdots, Headh)\times W^O, where the Head is defined as:

\[
\text{Head}_i = \text{Attention}(QW^Q_i, KW^K_i, VW^V_i)
\]  

where the projections are parameter matrices W^Q_i \in \mathbb{R}^{d_{model}\times d_k}, W^K_i \in \mathbb{R}^{d_{model}\times d_k}, and W^V_i \in \mathbb{R}^{d_{model}\times d_v}. Multi-head attention enables the model to jointly attend to information from different representation subspaces at different positions [18].

In this work, we implement a shallow Transformer and a large scale Transformer, i.e., RoBERTa. The shallow Transformer has h = 2 parallel attention layers with 4 attention heads and RoBERTa (base configuration) has 12 layers with 12 heads. For each head we use d_k = d_v = d_{model}/h = 200 and 768 for Transformer and RoBERTa, respectively.

Encoder: The encoder consists of a stack of N identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a FC feed-forward network. There is a residual connection around each of the two sub-layers, followed by layer normalization.

Decoder: The decoder contains a stack of N identical layers. Within each layer, there are three sub-layers, where the third sub-layer is the same as the encoder. The inserted second sub-layer performs multi-head attention over the output of encoder stack. The first sub-layer utilizes masked multi-head attention, to ensure that predictions for position i only depends on its previous positions.

Figure 2: (a) Scaled Dot-Product Attention. (b) Multi-Head Attention.
4 TRANSFORMER COMPRESSION USING ENHANCED BLOCK-CIRCULANT MATRIX

The introduced intensive computation and weight storage of large pre-trained language representations have brought challenges in hardware implementation. Therefore, model compression is a natural method to mitigate the these challenges.

4.1 Enhanced BCM-based Transformer

CirCNN [5] and C-LSTM [19] have adopted BCM for model compression on small to medium scale datasets in image classification and speech recognition, respectively, and achieved significant improvement in terms of performance and energy efficiency compared to the prior arts. Using this method, we can reduce weight storage by replacing the original weight matrix with one or multiple blocks of circulant matrices, where each row/column is the cyclic reformulation of others. We use $b$ to represent the row/column size of each circulant matrix (or block size, FFT size). Suppose the shape of a weight matrix in Transformer (e.g., $W^Q_i$, $W^K_i$, $W^V_i$) is $W \in \mathbb{R}^{m \times n}$, there will be $f \times g$ blocks after partitioning, where $f = m + b$ and $g = n + b$. Then $W = [W_{ij}], i \in \{1 \ldots f\}, j \in \{1 \ldots g\}$.

The input $x$ is also partitioned as $x = [x^1, x^2, \ldots, x^n]^T$. In each BCM, only the first column/row is needed for storage and computation, and is termed the index vector, $p_{ij}$. The theoretical foundation is derived in [20], which demonstrates the universal approximation property and the error bounds of BCM-based neural networks are as efficient as general neural networks.

Prior works [5, 19] have not investigated large-scale language representations. To further maintain the prediction accuracy, we use an enhanced BCM-based model compression. We modify the formulation of the index vector as follows:

$$p_{ij} = \frac{x^i \odot W_{ij}}{\sum_{j=1}^{b} W_{ij}}$$

where $W_{ij}$ is a circulant matrix. We observe that in this way, we can better preserve the parameter information and maintain the overall prediction accuracy. The main reason is that prior works take the first column/row as the index vector, missing the effective representations for other rows/columns.

Based on the circulant convolution theorem [14, 17], instead of directly performing the matrix-vector multiplication, we could use the fast Fourier transform (FFT)-based multiplication method, and it is equivalent to matrix-vector multiplication. The calculation of a BCM-based matrix-vector multiplication $W_{ij}x_j$ is: $W_{ij}x_j = p_{ij \odot x_j} = \text{IFFT}(\text{FFT}(p_{ij}) \odot \text{FFT}(x_j))$, where $\odot$ represents circular convolution, and $\circ$ is element-wise multiplication. Therefore, the computational complexity is reduced from $O(b^2)$ to $O(b \log b)$.

5 ARCHITECTURE

FPGA is widely used to accelerate deep learning models for its high parallelism and low latency. As large amount of transformer parameters exceed the on-chip memory or block RAM (BRAM) capacity on FPGA fabric, even with model compression technique, the full model cannot be stored on chip. To address the challenge, we partition a model into embedding layer and encoder/decoder stacks. The embedding layer contributes 30.89% of parameters. Essentially it is a look-up table which transforms discrete tokens into continuous space, the computation is less than that of encoder and decoder. Therefore, our basic idea is to off-load embedding layer to off-chip memory, thus it is possible to deploy the most computational intensive part, i.e. the encoder and decoder stack on chip, avoiding frequently access off-chip weights, hence to accelerate computation. Second, to mitigate the I/O constrain, we developed the inter-layer coarse grained pipelining, intra-layer fine grained pipelining, and computation scheduling.

5.1 Overall Hardware Architecture

As shown in Figure 3, the proposed hardware architecture consists of computation units for encode/decoder computation, on-chip memory banks, a transformer controller, and an off-chip memory (DDR) and DDR controller. The transformer controller communicates with the host and controls all the modules in FPGA. The host PC loads the inputs (i.e., sentence pairs) to the FPGA for inference through PCIe. On the FPGA part, given the tokenized sentences, the embedding look up module accesses DDR to fetch embeddings. Next, the embeddings will be fed into the pipelined encoder/decoder stacks to perform inference.

The computing units consist of multi-head attention, scaled dot product attention, point wise feed forward layer, linear, and add/norm. The transformer controller orchestrates the computing flow and data flow of inputs from PCIe, BRAMs and computing units on the FPGA fabric. Since the encoder and decoder share same type of operations, so we first decompose them into three different computing primitives, including matrix multiplication of different sizes, vectorized exponentials etc. The multi-head attention, linear, and add/norm modules are reconfigured to form as encoder or decoder under the transformer control logic. We have two pipeline strategies. For shallow networks, the entire network can be straightforwardly implemented, i.e. all layers can be implemented by dedicated FPGA resources. For the state-of-the-art designs such as BERT and RoBERTa, there are multiple encoders/decoders, hence the resource such as DSPs may not enough. In such cases, reuse of certain PE or entire encoder/decoder module are necessary.

5.2 Multi-Head Attention Design

Multi-head attention includes multi-processing elements (named PE) banks, for matrix multiplication), buffers (K buf, Q buf, and V buf), a normalization module (Norm), a masking function for masked multi-head attention, and a softmax module as described in Equation (2) and shown in Fig. 4.
units to support scaling and softmax required by multi-head attention. The output of multipliers are fed into divider or accumulator as stream, hence scaling and softmax layer can be overlapped with matrix multiplication.

5.3.2 FFT/IFFT-based PE. Figure 5 shows the design of FFT/IFFT-based PE and softmax, including a FFT/IFFT kernel, an accumulator, and an adder. The accumulator is an adder tree with N inputs (the size is chosen the same as the FFT/IFFT kernel size). We select Radix-2 Cooley Tukey algorithm [7] for FFT implementation.

5.3.3 Softmax Module. Figure 5 (b) shows the implementation of the softmax function $\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$. The exponential function $\exp(x_i)$ or $\exp(x_j)$ is expensive in resource consumption for FPGAs. We adopt piece-wise linear functions to estimate their outputs, in order to simultaneously reduce the resource consumption and maintain the accuracy. A buffer is used to store $\exp(x_i)$ and an accumulator is used to compute the summation of $\exp(x_j)$. Next, we perform the division and generate the softmax results.

6 DESIGN AUTOMATION & OPTIMIZATION

We developed a workflow to prototype and explore the hardware architecture. First, we generate a data dependency graph based on trained models to illustrate the computation flow: The operators in graph are scheduled to compose the pipeline under the design constraints, to achieve maximum throughput. At last, a code generator receives the scheduling results and generates the final C/C++ implementation, which can be fed into the commercial HLS tool for synthesis. Our target synthesis backend is Xilinx SDx.

The major computationally intensive operations are shown in Figure 6. Other operations such as division and softmax consume much less time, and can be merged/overlapped with these major operations. The computation in different layers can be decomposed into common computing elements, i.e., PEs. The layers in same color can be performed by same PE, however, with unbalanced operations. For example, the time consumed by the $\mathbf{K}^W \mathbf{K}^V$ and $\mathbf{V}^W \mathbf{V}^V$ is roughly 4 times of computation required by the $n$ heads. To improve the utilization of pipeline, it is desirable to let
each layer consumes roughly the same time. This can be achieved by allocating more resources to the slowest layer. We adopt a two-stage optimization flow. In first stage, we find a resource scheme that can minimize the maximum time required by layers. In second stage, under such resource constrains, we optimize the scheduling of an individual encoder/decoder.

The optimization starts from a basic implementation of an individual encoder/decoder, i.e. no parallelization nor resource reusing, such that we can obtain an estimation of resource consumption, number of operations and execution time of each layer, throughput put obtained by unit number of resources. Then we will examine how much resource can be allocated to each encoder/decoder to minimize the execution time of the slowest layer:

$$\text{minimize} \quad \max(T_1, T_2, ..., T_n),$$

subject to

$$R_f[i] \geq M \sum_j R_j[i] + R_{misc}[i]$$

where $i \in (0, ..., 3)$, $j \in n$, $n$ is the number of layers, $M$ is the total number of encoder/decoder, $R_f = [R_{FF}, R_{LUT}, R_{DSP}, R_{BRAM}]$ is on-chip resource constraints for look-up table (LUT), flip-flop (FF), digital signal processing unit (DSP), and BRAM, respectively. $T_j$ is the time required by the $j$-th layer. $R_j$ is resource utilization of the $j$-th layer, which is also represented as a vector: $R_j = [R^i_{FF}, R^i_{LUT}, R^i_{DSP}, R^i_{BRAM}]$. $R_{misc}$ is the resource utilization of modules except encoder/decoder module, such as DDR controller, PCIe controller, etc. $T_j$ can be given as:

$$T_j = [N_1 \cdot (F_j \cdot K_j)], j \in n$$

where $N_{op}$ is the number of operations required by the $j$-th layer. $K_j$ is resource allocation factor of the $j$-th layer. $F_j$ is the throughput of non-optimized design, which can be obtained empirically. Therefore, the throughput is:

$$\text{Throughput} = \frac{freq}{(n \cdot \max(T_1, T_2, ..., T_j))}$$

It finds the slowest layer, allocates more resources, then updates the resource consumption and execution time. If resource constraints are satisfied, we repeat this procedure until no more speedup. Then the algorithm will examine the fastest layer. If it takes significantly less time than the slowest layer, it is possible to allocate less resources for that layer, hence more resources can be assigned to the slowest layer. After this procedure, we obtain resource constraints, e.g. the No. of different PEs of an encoder and decoder. Under resource constraints, each layer may not have dedicated computation resource, hence matrix multipliers, adders, etc. have to be shared. Therefore, the computation has to be carefully scheduled to minimize the computation latency. The encoder/decoder can be represented as a Directed Acyclic Graph (DAG) $G(V, E)$, where $V$ is a set of vertices representing different computation, edges $E$ indicate the dependency. The available computation units such as PEs and adders are represented by a set $Op = \{PE-A1, PE-A2, ..., Adder\}$. The algorithm used for operation scheduling takes $G$ and $Op$ as input, is shown in Algorithm 1.

### 7 EVALUATION

#### 7.1 Training of Transformer-Based Language Representation

In this section, we apply both enhanced BCM-based model compression on the linear layers, and adopt 16 fixed-point data representation for all the weights. We evaluate the accuracy impact with two representative Transformer structures, i.e., a shallow Transformer with both encoder and decoder, and a pretrained deep Transformer architecture - RoBERTa (base configuration) which only has encoder [10]. The shallow Transformer is evaluated in a language modeling task, which is an unsupervised sequence-to-sequence problem that requires the decoder part. On the other hand, we run a RoBERTa on a sentiment classification task that is a supervised classification problem without the requirement for decoder block. The software is implemented in PyTorch deep learning framework [15] and FairSeq sequence modeling toolkit [13]. Table 1 summarizes the key parameters of the shallow Transformer and RoBERTa models in the experiments.

| Model Configuration | Transformer Structure | Transformer Layers | Hidden Size | Attention Heads | Total Params |
|---------------------|-----------------------|-------------------|-------------|-----------------|-------------|
| Shallow Transformer  | encoder-decoder only   | 2                 | 768         | 12              | 125M        |
| RoBERTa (base config.) | encoder-decoder only   | 12                | 768         | 12              | 125M        |

#### 7.1.1 Finetuned RoBERTa for Sentiment Classification

We evaluate the proposed model compression method for finetuned RoBERTa [10] on IMDb movie review sentiment classification [11] to shed some light on training trial reductions. Starting from the saved state of pretrained models in work [10], we finetune the model until it reaches to its best validation accuracy at 95.7%. To maintain overall accuracy, we compress partial layers. The process suppresses randomness by using a deterministic seed. Thus the accuracy difference between the original RoBERTa and compressed version is solely contributed by the compression techniques.

#### 7.1.2 Shallow Transformer

Language modeling task takes a sequence of words as input and determines how likely that sequence is resource allocation factor of the $j$-th layer.

$$\text{minimize} \quad \max(T_1, T_2, ..., T_n),$$

subject to

$$R_f[i] \geq M \sum_j R_j[i] + R_{misc}[i]$$

where $i \in (0, ..., 3)$, $j \in n$, $n$ is the number of layers, $M$ is the total number of encoder/decoder, $K_j$ is the number of operations required by the $j$-th layer.

$$\text{Throughput} = \frac{freq}{(n \cdot \max(T_1, T_2, ..., T_j))}$$

It finds the slowest layer, allocates more resources, then updates the resource consumption and execution time. If resource constraints are satisfied, we repeat this procedure until no more speedup. Then the algorithm will examine the fastest layer. If it takes significantly less time than the slowest layer, it is possible to allocate less resources for that layer, hence more resources can be assigned to the slowest layer. After this procedure, we obtain resource constraints, e.g. the No. of different PEs of an encoder and decoder. Under resource constraints, each layer may not have dedicated computation resource, hence matrix multipliers, adders, etc. have to be shared. Therefore, the computation has to be carefully scheduled to minimize the computation latency. The encoder/decoder can be represented as a Directed Acyclic Graph (DAG) $G(V, E)$, where $V$ is a set of vertices representing different computation, edges $E$ indicate the dependency. The available computation units such as PEs and adders are represented by a set $Op = \{PE-A1, PE-A2, ..., Adder\}$. The algorithm used for operation scheduling takes $G$ and $Op$ as input, is shown in Algorithm 1.

### 7 EVALUATION

#### 7.1 Training of Transformer-Based Language Representation

In this section, we apply both enhanced BCM-based model compression on the linear layers, and adopt 16 fixed-point data representation for all the weights. We evaluate the accuracy impact with two representative Transformer structures, i.e., a shallow Transformer with both encoder and decoder, and a pretrained deep Transformer architecture - RoBERTa (base configuration) which only has encoder [10]. The shallow Transformer is evaluated in a language modeling task, which is an unsupervised sequence-to-sequence problem that requires the decoder part. On the other hand, we run a RoBERTa on a sentiment classification task that is a supervised classification problem without the requirement for decoder block. The software is implemented in PyTorch deep learning framework [15] and FairSeq sequence modeling toolkit [13]. Table 1 summarizes the key parameters of the shallow Transformer and RoBERTa models in the experiments.

| Model Configuration | Transformer Structure | Transformer Layers | Hidden Size | Attention Heads | Total Params |
|---------------------|-----------------------|-------------------|-------------|-----------------|-------------|
| Shallow Transformer  | encoder-decoder only   | 2                 | 768         | 12              | 125M        |
| RoBERTa (base config.) | encoder-decoder only   | 12                | 768         | 12              | 125M        |

#### 7.1.1 Finetuned RoBERTa for Sentiment Classification

We evaluate the proposed model compression method for finetuned RoBERTa [10] on IMDb movie review sentiment classification [11] to shed some light on training trial reductions. Starting from the saved state of pretrained models in work [10], we finetune the model until it reaches to its best validation accuracy at 95.7%. To maintain overall accuracy, we compress partial layers. The process suppresses randomness by using a deterministic seed. Thus the accuracy difference between the original RoBERTa and compressed version is solely contributed by the compression techniques.

#### 7.1.2 Shallow Transformer

Language modeling task takes a sequence of words as input and determines how likely that sequence is the actual human language. We consider the popular WikiText-2 dataset [12] in this experiment, which contains 2M training tokens with a vocabulary size of 35k. A shallow Transformer model with 4 attention heads and 200 hidden dimension is established.

The baseline and model compression results of shallow Transformer and RoBERTa on WikiText-2 and IMDb review are shown
Table 2: Comparison among different model configurations

| ID | Network Type | Block Size | Wt/Text-2 (ACC) % | Latency (ms) with BCM (s) | Latency (ms) with BCM & Quant. (%) |
|----|--------------|------------|-------------------|---------------------------|-----------------------------------|
| 1  | Shallow Transformer | 3       | 93.5               | 25.96                      | 30.5820                           |
| 2  | Shallow Transformer | 4       | 96.7               | 0.6                        | 0                                 |
| 3  | Shallow Transformer | 8       | 90.7               | 0.6                        | 0                                 |
| 4  | Shallow Transformer | 16      | 90.0               | 1.3                        | 0                                 |

Table 3: Comparison among different model configurations

| Batch Size | DSP | FF | LUT | Latency (ms) | Power (W) | Throughput (FPS) |
|------------|-----|----|-----|--------------|-----------|------------------|
| 1          | 5647| 304012| 268933 | 2.94 | 22.45 | 680.91 |
| 2          | 5647| 304296| 269361 | 11.59 | 22.52 | 690.50 |
| 3          | 5647| 305820| 269753 | 22.90 | 22.66 | 698.72 |
| 4          | 5647| 306178| 270449 | 55.94 | 22.73 | 702.54 |

8 CONCLUSION

In this paper, we propose an energy-efficient acceleration framework for transformer-based large scale language representations. Our framework includes an enhanced BCM-based method to enable model compression on large-scale language representations at the algorithmic level, and an acceleration design at the architecture level. We propose an FPGA architecture design to support the model compression technique and we develop a design automation and optimization technique to explore the parallelism and achieve high throughput and performance. Experimental results show that our proposed framework significantly reduces the size of NLP models with small accuracy loss on Transformer. Our FPGA-based implementation significantly outperforms CPU and GPU in terms of energy efficiency.

REFERENCES

[1] 2019. Supercharge Your AI and Database Applications with Xilinx's HBM-Enabled UltraScale+ Devices Featuring Samsung HBM2. Xilinx white paper, WP508 (v1.1.2) (2019).
[2] James Bradbury and Richard Socher. 2017. Towards Neural Machine Translation with Latent Tree Attention. EMNLP 2017 (2017), 12.
[3] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. arXiv preprint arXiv:1409.1259 (2014).
[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018).
[5] Caiwen Ding, Siyu Liao, Yanzhi Wang, Zhe Li, Ning Liu, Youwei Zhuo, Chao Wang, Xuehai Qian, Yu Bai, Geng Yuan, et al. 2017. Crcnn: accelerating and compressing deep neural networks using block-circulant weight matrices. In Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture. 395–408.
[6] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. Neural computation 9, 8 (1997), 1735–1780.
[7] S. Lehnert, T. Kretschmer, and Robert L. Krzak. 1992. Cooley-tukey FFT on the Connection machine. Parallel Comput. 18, 11 (1992), 1201–1221.
[8] Joo-Kyung Kim and Young-Bum Kim. 2018. Supervised Domain Enablement Attention for Personalized Domain Classification. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing. 894–899.

The accuracy drop on RoBERTa is slightly higher because its parameters are carefully pretrained on the Giga byte dataset (160GB of text) using a masked language model [10] and more sensitive to compression.
[9] Yoon Kim, Carl Denton, Luong Hoang, and Alexander M Rush. 2017. Structured attention networks. arXiv preprint arXiv:1702.00887 (2017).
[10] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Niki Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692 (2019).
[11] Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In ACL Association for Computational Linguistics, 142–150.
[12] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. [n. d.]. Pointer Sentinel Mixture Models. In 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017.
[13] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A Fast, Extensible Toolkit for Sequence Modeling. In Proceedings of NAACL-HLT 2019: Demonstrations.
[14] Victor Pan. 2012. Structured matrices and polynomials: unified superfast algorithms. Springer Science & Business Media.
[15] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. (2017).
[16] Peng Shi, Jinfeng Rao, and Jimmy Lin. 2018. Simple Attention-Based Representation Learning for Ranking Short Social Media Posts. arXiv preprint arXiv:1811.01013 (2018).
[17] Julius Orion Smith. 2007. Mathematics of the discrete Fourier transform (DFT): with audio applications. Julius Smith.
[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In Advances in neural information processing systems. 5998–6008.
[19] Shuo Wang, Zhe Li, Caiwen Ding, Bo Yuan, Qintu Qin, Yanzhi Wang, and Yun Liang. 2018. C-LSTM: Enabling Efficient LSTM Using Structured Compression Techniques on FPGAs. In FPGA’18.
[20] Liang Zhao, Siyu Liao, Yanzhi Wang, Zhe Li, Jian Tang, and Bo Yuan. 2017. Theoretical Properties for Neural Networks with Weight Matrices of Low Displacement Rank. In International Conference on Machine Learning, 4082–4090.