Title
T-OPU: An FPGA-based Overlay Processor for Natural Language Processing

Author
JIAN, Yiheng

Publication Date
2022

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA
Los Angeles

T-OPU: An FPGA-based Overlay Processor for Natural Language Processing

A thesis submitted in partial satisfaction
of the requirements for the degree Master of Science
in Electrical and Computer Engineering

by

Yiheng JIAN

2022

ABSTRACT OF THE DISSERTATION

T-OPU: An FPGA-based Overlay Processor for Natural Language Processing

by

Yiheng JIAN
Master of Science in Electrical and Computer Engineering
University of California, Los Angeles, 2022
Professor Lei HE, Chair

There has been a rapid development of custom accelerators to speed up the training and inference of deep neural networks (DNNs) by using their parallel computing resources. Recently, most accelerators focused on convolutional neural networks (CNNs), which are composed of linear functions (matrix multiplication) in convolutional or fully connected layers. There is no publicly available study on accelerating the transformers. Transformers have achieved great success in many artificial intelligence fields and attracted lots of interest from academic and industry researchers. Bidirectional Encoder Representation from Transformers (BERTs) are the most recent Transformer-based model that achieves state-of-the-art performance in various Natural Language Processing (NLP) tasks. Unlike the CNNs, there are numerous nonlinear functions in BERT in softmax, layer normalization, and GELU layers. In this paper, we propose an FPGA-based accelerator of quantized BERT for NLP. The accelerator provides the end users with software-like programmability, which means it does not require hardware reconfiguration when the models are modified or updated. It can achieve state-of-the-art performance, power, and area (PPA) compared to existing studies.

The dissertation of Yiheng JIAN is approved.

Nader Sehatbakhsh

Jonathan Kao

Lei HE, Committee Chair

University of California, Los Angeles

2022

TABLE OF CONTENTS

1 Introduction **1**
1.1 Overview 1
1.2 Contribution 2
1.3 Outline 2

2 Background and related work **3**
2.1 Transformer 3
2.1.1 Vaswani Transformer 3
2.1.2 BERT 3
2.2 FPGA Accelerator 4
2.2.1 FPGA 4
2.2.2 FPGA accelerator for neural network 5

3 Architecture **8**
3.1 ACCELERATOR ARCHITECTURE 8
3.2 Matrix Multiplier Unit 8
3.3 Architecture of Memory 9
3.4 Nonlinear Vector Unit (NVU) Architecture 10
3.4.1 Softmax 10
3.4.2 Layer Normalization 12
3.4.3 GELU 13

4 Evaluation **15**
4.1 FPGA utilization 15
4.2 Inference time 16
4.3 Comparison with CPU, GPU, and FPGA 17
4.4 Comparison with CPU, GPU, and FPGA 18
4.5 Benchmarks Discussion 19

5 Conclusion **20**

References **21**

LIST OF FIGURES

2.1 Architecture of $\text{BERT}_{\text{BASE}}$ 4
2.2 A typical structure of an FPGA-based NN accelerator [GZY] 6
2.3 Architecture of OPU 7

3.1 Overview architecture 8
3.2 Architecture of Matrix Multiplication Unit 9
3.3 Architecture of NVU 10
3.4 Approximate and accurate curve of exponential 11
3.5 Approximate and accurate curve of inverse of square root 13
3.6 The plot of theory gelu function and approximated gelu function 14

4.1 BERT inference time percent overhead using T-OPU with 16-bit MMU for different sequence lengths and NVU variants. Overhead is relative to the minimum time using NVU-2048 16
4.2 BERT inference time (in ms) with different NVU widths and sequence lengths. Results are shown separately for T-OPU with 8-bit and 16-bit matrix multiplies 17

LIST OF TABLES

4.1 Overall FPGA resource Utilization on Zynq Z-7100 for T-OPU with 8-bit and 16-bit MMU with NVU-256, NVU-512, and NVU-1024 15

4.2 FPGA Resource Utilization for components of NVU-256, NVU-512, and NVU-1024 on Zynq-7100 16

4.3 Throughput (inference/sec) of NPE with NVU-1024 compared with CPU (i7-8700k), GPU (RTX 5000), and FPGA (VCU118). We also give relative throughput compared to FTRANS, throughput per DSP slice relative to FTRANS (for FPGA implementations), and approximate power consumption 18

ACKNOWLEDGMENTS

I would like to thank my supervisor Lei HE, as well as Hamza Khan, Chen Wu and Tiandong Zhao, for their guidance throughout this project. Also thanks to my committee members, Jonathan and Nadar, who offered guidance and support.

Chapter 4 is a version of ”NPE: An FPGA-based Overlay Processor for Natural Language Processing”, which is submitted. The authors are Hamza Khan, Jiawei Zhang, Yiheng Jian, Tiandong Zhao and Lei He.

CHAPTER 1

Introduction

1.1 Overview

Transformer is a promising deep learning model that have been widely utilized in natural language processing (NLP). It is originally introduced to solve the machine translation tasks. Later research shows representative potential in various tasks. As transformer develops, more and more researchers join and the application of transformer is rapidly expanding to other fields. Researchers in computer vision (CV) field utilize transformer in image recognition [DBK] and object detection [CMS] and achieve about the same results as that of CNN.

Transformer adopts a mechanism named self-attention to replace recurrence used in RNN and LSTM, which fail to capture long-term dependency in sentence. Its architecture contains various components, and they could be broadly divided into 2 parts: encoder and decoder.

Bidirectional Encoder Representations from Transformers (BERT) [DCL] is the state-of-the-art transformer-based model. It is simple encoder representation of transformer and neglect the decoder. The encoder consists of 2 parts. One of them is self-attention and the other is feed forward neural network. Unlike other transformer variants, BERT is dedicated to learning language representations. As soon as it introduced, it beat other NLP models and broke the existing results in 11 NLP tasks.

Field Programmable Gate Arrays (FPGAs) have been demonstrated to be an effective hardware platform to accelerate the training and inference of neural networks. Compared to CPU or GPU, whose hardware and software are designed independently, FPGAs could be dedicated for the target algorithms. This enables developers to implement only necessary logic in hardware. Besides, FPGAs are promising given its full capacity of parallelism in network computation to achieve low latency and high performance with better energy efficiency.

Current FPGA accelerators are mostly designed for traditional networks, including CNN, which consists mostly of linear matrix operations like convolution and batch normalization. However, BERT networks have a unique mix of computations, as nonlinear operations like softmax and layer normalization are introduced in attention layer in addition to standard matrix multiply based layers. FPGA accelerator designed for CNN cannot handle transformer or BERT networks.

Most existing accelerators include specialized units for computing each type of nonlinearity. For instance, FTRANS, the only previously published FPGA accelerator for transformers, includes separate softmax and layer normalization modules. Since NLP is a constantly evolving

field that may introduce different types of non-linearity, this specialized approach means that an FPGA design may need reconfiguration for additional NLP networks. It also leads to unnecessary area overhead and under-utilized resources across nonlinear operations.

1.2 Contribution

In this paper, an FPGA based overlay processor name T-OPU for BERT model inference is proposed. Unlike most other accelerators, T-OPU employ a common method for approximating different nonlinear functions efficiently and accurately without added overhead. The main contributions of our work are as follows:

1. We design a software-programmable domain-specific overlay processor with a matrix multiply unit and a multi-precision vector unit for NLP processing. We employ a unified piecewise polynomial approach for nonlinear function approximation to allow extensibility to future nonlinear functions that may be required.

2. We demonstrate that our proposed accelerator can meet the real-time latency constraints for conversational AI while maintaining 4x and 6x lower power than GPUs and CPUs, respectively. Our design utilizes 3x fewer FPGA resources.

1.3 Outline

This dissertation is organized as follows. The background and related work of this research is presented in Chapter 2. The architecture of our implementation is present in Chapter 3. Chapter 4 present the experiment design and results. Finally, conclusions and future work are given in Chapter 5.

CHAPTER 2

Background and related work

In this chapter, the background for T-OPU is presented. We first introduce the structure of BERT, then discuss the development of FPGA accelerators. After that, FPGA accelerators designed for BERT will be presented.

2.1 Transformer

2.1.1 Vaswani Transformer

Transformer, first introduced by Vanilla[VSP], is a sequence-to-sequence model and consists of an encoder and a decoder. Each encoder consists of Multi-head self-attention, feed-forward network, positional encoding, and embedding layers. The output of the encoder is connected to the decoder. The decoder structure of the transformer consists of output embedding, positional encoding, decoder layers (masked multi-head attention, multi-head attention, feed-forward). Together, these work as a transformer. Self-attention mechanism solves perfectly the problem of long-term dependency. It is not restricted to a fixed structure like RNN or troubled by limited by the size of receptive fields in CNN.

2.1.2 BERT

BERT[DCL] stands for Bidirectional Encoder Representation from Transformers. It is a language model learning representations with transformer by representing the relationships of the words in a language. Compared to Vaswani transformer, BERT is only an encoder, which is similar to that of original transformer. In the architecture of BERT, all recurrence is replaced by attention and feed-forward layers.

The structure of BERT is composed of Embedding layer, transformer-encoder layer, and output layer. The input is obtained by summing the three types of features: token embedding, segment embedding, and position embedding. The transformer-encoder layer is the key component of BERT. It is composed of N identical encoding layers. In Vaswani paper, N is set to 6. An encoding layer has 2 sub-layers: multi-head self-attention module and fully connected feed-forward network. These 2 sub-layers are connected by a residual layer followed by a layer normalization layer. In this way, the output of the sub-layer could be described as:

$$\text{sub-layer} = \text{LayerNorm}(x + \text{sub-layer(x)}) \tag{2.1}$$

Given the Query $\boldsymbol{Q}$, Key $\boldsymbol{K}$, and Value $\boldsymbol{V}$, the multi-head attention could be described as:

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{Softmax}\Big(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\Big)\boldsymbol{V} \tag{2.2}$$

Our implementation focuses on $\text{BERT}_{\text{BASE}}$, which is based on stacked layers of encoders. It contains 12 encoder layers and each has 12 attention heads and a hidden layer of 768. The structure of

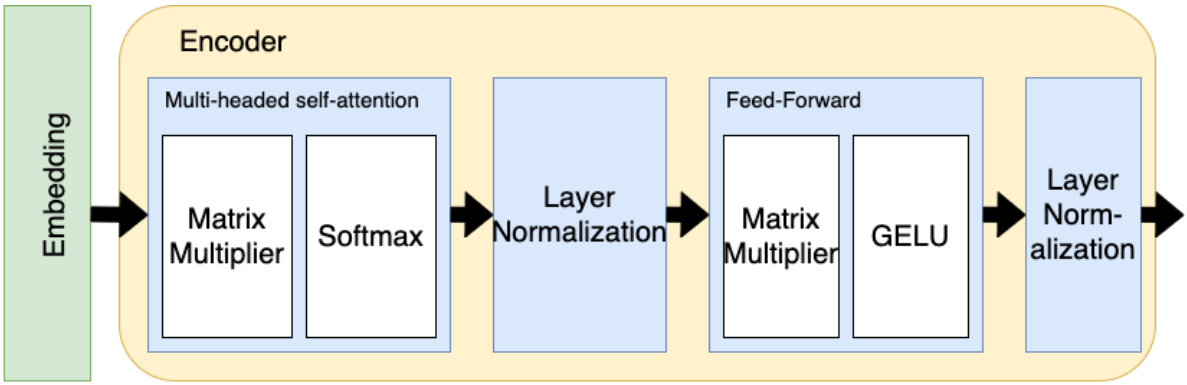

Figure 2.1: Architecture of $\text{BERT}_{\text{BASE}}$

$\text{BERT}_{\text{BASE}}$ is shown in figure 2.1. The model starts with an embedding layer that converts input language into features. An input language sequence of 512 tokens will be converted into a 512×768 matrix. Each token will be converted by a 768-length vector.

The embedding layer is followed by 12 encoders, each of which performs 4 operations: multi-headed self-attention, layer normalization, feed-forward layers, and layer normalization. Those operations could be decomposed into 4 computation units: matrix multiplication, softmax, layer normalization, and GELU.

2.2 FPGA Accelerator

2.2.1 FPGA

Field programmable gate arrays (FPGAs) [KTR] are semiconductor devices that can be re-configured by users even after manufacturing. It emerges as a semi-custom circuit in the field of application-specific integrated circuits (ASICs). An FPGA consists of a matrix of configurable logic blocks (CLBs) connected by programmable interconnections. The combination logic is implemented by small lookup tables (LUTs). Each lookup table is connect to the input of a D

flip-flop, which in turn drives other logic circuits or drives I/O. This constitutes a basic logic cell module that can implement both combination and timing logic functions. The logic of the FPGA is implemented by loading programming data into the internal static memory cells. The values stored in the memory cells determine the logic functions of the logic cells and the way modules are connected to each other.

In comparison to ASICs, FPGAs have lower performance, energy efficiency, and higher latency. For the same logic, the implementation of FPGAs requires a much larger area than that of ASICs. Despite these disadvantages, FPGAs still provide a compelling alternative to ASICs. Due to the programmability and abundant hardware resources, the development cycles of FPGAs is shorter than that of ASICs. The developers can implement their algorithms rapidly with provided logic cells, multipliers, and on-chip memory. In addition, the capability of reconfiguration enables users to update the hardware design when algorithms develop.

2.2.2 FPGA accelerator for neural network

Massively parallel processing, *distributed storage*, and *elastic topology* are 3 of the neural network characteristics. Neural networks have natural parallelism, which is determined by their structural features. Each neuron can perform independent operations and processing based on the received information and output the result. Different neurons in the same layer can perform operations simultaneously and then transmit to the next layer for processing. This means that neural networks can take full advantage of parallelism. The networks store the information in the adjusted values of each weight. During the training and inference process of neural networks, the processors need to read and write massive intermediate data from and to storage frequently.

Based on above information, CPUs of Von Neumann architecture are not the best solution for deep learning. Although general purpose processors are easy to program, they have to execute instructions in sequence and cannot take full advantage of the parallelism of neural networks. In addition, a lot of time is wasted in analyzing instructions, reading out and writing data. GPUs provide an alternative platform to accelerate the deep learning by offering flexible frameworks and highly programming interfaces. The frameworks are specially optimized for deep learning algorithms so that they can run in full parallel. The programming interfaces and compatible API allow developers to implement their algorithms with their preferred languages including C, python, and matlab. But, the high-energy consumption make it hard for GPUs to deploy on edge side.

FPGA is becoming an effective solution for deep learning algorithm acceleration and widely adopted to accelerate the training and inference process of neural network model. Unlike general processors like CPUs and GPUs, FPGA accelerators are dedicated for neural networks. Because they are required to support one or a series of algorithms, FPGAs only need to implement necessary logic. By eliminating redundant logic, FPGA accelerators can achieve high-performance and low latency with better energy efficiency.

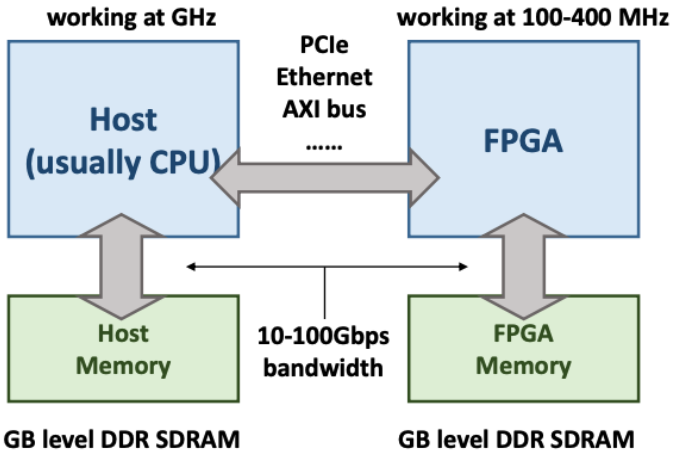

Figure 2.2: A typical structure of an FPGA-based NN accelerator [GZY]

A typical structure of an FPGA-based NN accelerator is shown in figure 2.2. The system usually consists of Host part and FPGA part. The host and FPGA could be connected through PCIe, Ethernet, and even USB. Both host and FPGA have external memory to store the model and kernel data. In general, the FPGA will handle the training or inference of the neural network and the host will serve to control the FPGA with software.

A domain-specific FPGA overlay processor named OPU [YWZ] is proposed to accelerate CNN networks. On the host side, a compiler is built to compile the deep learning model of popular framework like tensorflow and pytorch into executable instructions that could be decoded by OPU. The generated instructions along with model kernel data will be sent to and executed by OPU. By re-compiling switched or updated models, the OPU can accelerate different networks without reconfiguration.

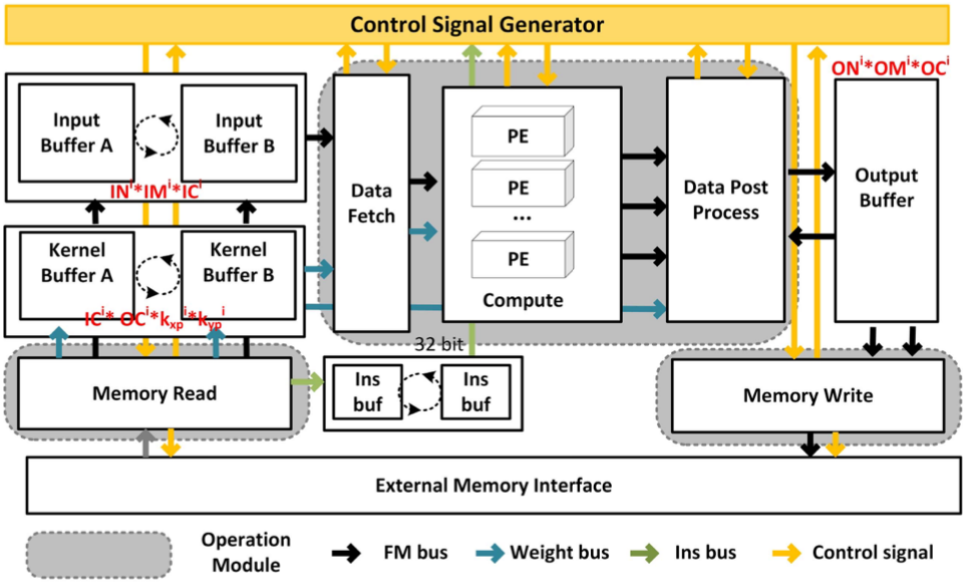

Figure 2.3: Architecture of OPU

The overview architecture of OPU is shown in figure 2.3. It can be decomposed into 6 main modules and 4 storage buffers. This paper demonstrates that OPU can achieve 2.9x faster compared with Jetson Tx2, which has a similar amount of computing resources.

CHAPTER 3

Architecture

3.1 ACCELERATOR ARCHITECTURE

In this section, the overall architecture of T-OPU is presented. T-OPU is consisted of data fetch unit, Matrix Multiply Unit (MMU), output control unit, Nonlinear Vector Memory (NVM) unit, and Nonlinear Vector (NV) unit. Among these units, MMU and output control unit are adopted from OPU. Figure 3.1 shows the architecture of the accelerator. In this picture, blue represents compute units and green for memory.

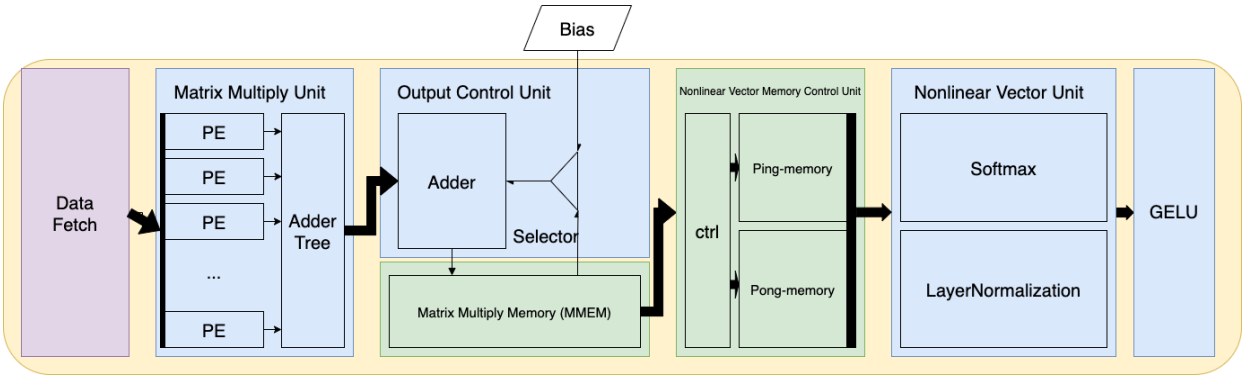

Figure 3.1: Overview architecture

3.2 Matrix Multiplier Unit

Our implementation of the matrix multiplication unit is shown in figure 3.2. It is composed of a process unit (PE) array and an adder tree, followed by a register array to store the result.

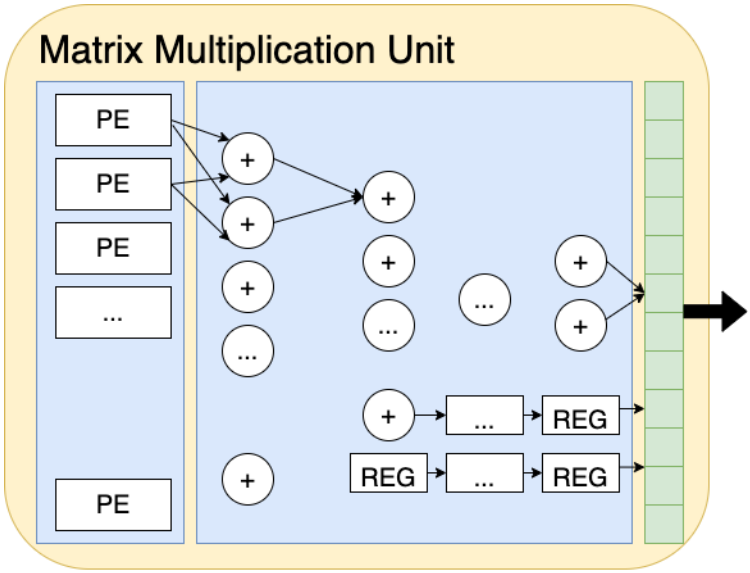

Figure 3.2: Architecture of Matrix Multiplication Unit

Among the PE array, each PE contains a multiplier with large bit-width input. With different data arrangement, a PE could support multiplication between different bit-width data. Assume the data width of input feature map and kernel is 8-bit, then a PE could handle the multiplication between 1 input feature map and 2 kernel data at the same time by concatenating 2 kernel data together. When we are processing NLP tasks, the data width is usually 16-bit. So, 1 PE unit can only handle the multiplication between 1 input feature map and kernel data. In our implementation of T-OPU, the supported data width choices include 4, 8, and 16-bit.

The MMU also supports multiple channel counters by limiting the depth of the adder tree. The outputs number choices include 64, 32, 16, 8, 4, and 2. This allows the computation unit to flexibly fit into the needs of different combination of input-output channels. Assume the number of input channel is 1024, the number of output channel should be 32 if it interrupts at the 5th layer of adder tree.

3.3 Architecture of Memory

Bandwidth is one of the critical issues that limit accelerator overall performance. To solve the data congestion and improve bandwidth, a ping-pong structure-based caching memory named NVU memory (NMEM) is adopted between the output control module and NVU to hide off-chip latency. It is implemented using single-port BRAMs on FPGA. While one of the buffer's data is being fetched by the NVU, the other buffer could get refilled and updated by the output control module. In this way, the maximum bandwidth utilization could be maintained.

3.4 Nonlinear Vector Unit (NVU) Architecture

The nonlinear vector unit is a critical component of our T-OPU. It implements non-linear function computation including softmax, layer normalization, and GELU with minimal resource. To support different scale of BERT model, our architecture is flexible enough. As shown in figure 3.3, the input data fed in will be first stored in the input buffer. After data transmission is completed, the data in input buffer will be loaded into vector compute unit for nonlinear function computation.

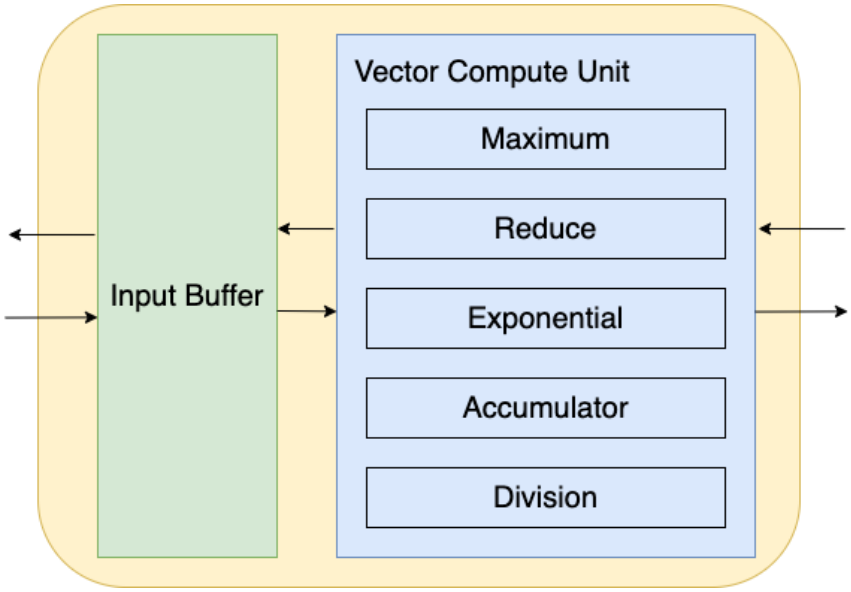

Figure 3.3: Architecture of NVU

To support different scales of BERT model, we adopt a piecewise architecture for input buffer. The data will come in package by package. Each package contains 32 numbers and the maximum capacity of the cache is 8 packages, which is 256 numbers. The input data length could range from 1 to 256, which is flexible for different models. Assume that the length of data from output control unit is 128, the data will be fully received in 4 cycles and the rest of memory will be filled with 0.

3.4.1 Softmax

The softmax function is a normalized exponential function. It always serves to be the last activation function in DNNs. Since it contributes little to run time of the inference process, little attention has been given to it. In our OPU, the softmax layer is processed on host side by software. However, as the transformer become more and more important, the infection of softmax can no longer be ignored. A transformer layer consists of a multi-headed self-attention block and a feed forward block. The softmax unit is performed nearly in every transformer layer.

$$P(y=j|x) = \frac{e^{x^T w_j}}{\sum_{k=1}^{K} e^{x^T wk}} \tag{3.1}$$

As shown in eq 3.1, the softmax function converts the input data into a vector of probabilities. To implement a softmax function, several operations including exponential, accumulator, and division are required. The softmax module is composed of 1 input buffer and 5 compute unit including maximum, subtraction, exponential, accumulator, and division. The data loaded from the input buffer will be sent through each compute unit in sequence.

To implement an exponential function with minimum hardware resources, we approximate it with a segmentation function. The function could be described as eq 3.2.

$$\text{Exp}(x) = \begin{cases} 0 & x < -6.2383 \\ 0.0156(x+6.2383)+0.002 & -6.2383 \leq x < -3 \\ 0.0859(x+3)+0.0488 & -3 \leq x < 2 \\ 0.2324(x+2)+0.1348 & -2 \leq x < -1 \\ 0.6328(x+1)+0.3672 & -1 \leq x \end{cases} \tag{3.2}$$

The curves of accurate function and approximate function are plotted in figure 3.4. As shown in the picture, the difference between accurate and approximate function is subtle. In this way, the nonlinear exponential function is replaced by a linear function with comparator, subtraction, addition, and constant multiplication. It takes 3 cycles to complete the computation.

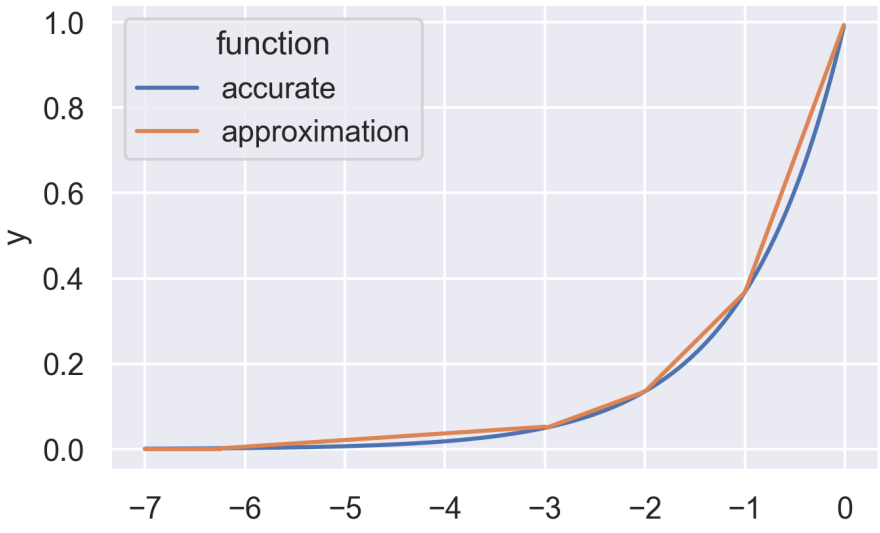

Figure 3.4: Approximate and accurate curve of exponential

3.4.2 Layer Normalization

Layer normalization function is widely used in transformer network. Unlike batch normalization, it is calculated across the feature dimension for each element and instance independently. It could be expressed as Eq. 3.3

$$\begin{aligned} \mu^l &= \frac{1}{H}\sum_{i=1}^{H} a_i^l \\ \sigma^l &= \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^l - \mu^l)^2} \\ \hat{x_{i,k}} &= \frac{x_{i,k} - \mu_i}{\sqrt{\sigma_i^2}} \end{aligned} \tag{3.3}$$

In eq 3.3, H represents the number of hidden elements in a layer and l is the short of the word "layer". The layer normalization calculates the standard variance of 1 layer data and the process could be divided into several parts including mean, subtraction, square, and square root.

To implement the standard variance with limited resources, a continuous piecewise linear approximation is adopted. We use a partial segmentation method to describe the non-linear function. The evaluation of it could be described as algorithm 1.

Algorithm 1: Continuous piecewise linear approximation method

Data: $\boldsymbol{X} = [x_0, x_1, \cdots, x_n]$
Input: x
Output: $v(x)$

1 $x_{i-1} \leq x < x_i$;
2 $\delta = (x - x_{i-1})/(x_i - x_{i-1})$;
3 $v(x) \approx (1-\delta)v(x_{i-1}) + \delta v(x_i)$

On a CPU or GPU, this operation may take couples of instructions, while it only take 4 cycles. Although piecewise linear approximation is not always accurate enough, by limiting the fixed point input and subsequent denormalization of the output, it could keep accurate with a few segments.

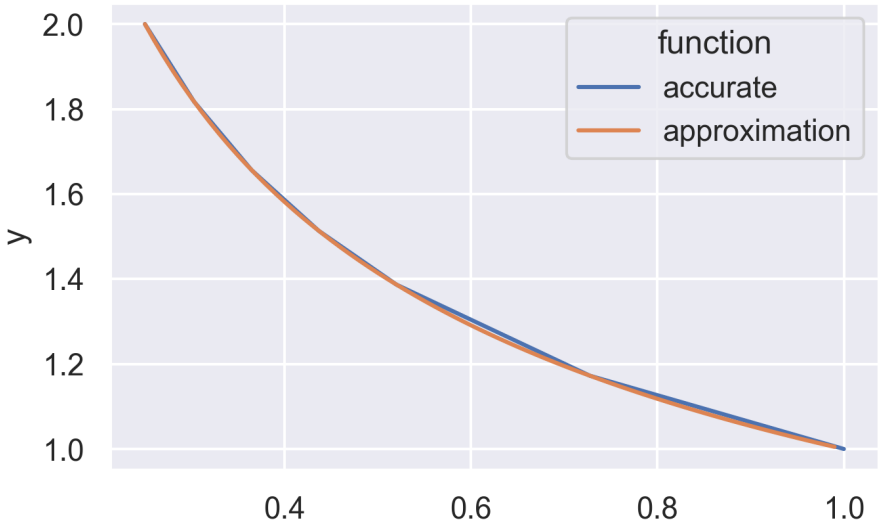

Figure 3.5: Approximate and accurate curve of inverse of square root

We implement a 7 segments approximation and the comparison with accurate function is shown in figure 3.5. Through the figure, we could find that the approximation curve is close to the accurate curve.

3.4.3 GELU

The Gaussian Error Linear Unit, or GELU, is an activation function. Different from ReLU, the GELU function weights inputs by their percentile.

The GELU function is defined by the following equation:

$$\text{GELU}(x) = xP(X \leq x) = x\Phi(x) = x \cdot \frac{1}{2}[1 + \text{erf}(x/\sqrt{2})] \tag{3.4}$$

In eq 3.4, *erf* is the short of error function. It is commonly to approximate the GELU with eq 3.5.

$$\text{GELU}(x) = 0.5x(1 + \tanh[\sqrt{2/\pi}(x + 0.044715x^3)]) \tag{3.5}$$

To make hardware implementation easier without accuracy loss, a segmentation function.

$$\text{GELU}(x) = \begin{cases} 0 & x < -3 \\ -0.0773(x+3) - 0.004 & -3 \leq x < -1 \\ 0.1587(x+1) - 0.1587) & -1 \leq x < 0 \\ 0.8413x & 0 \leq x < 1 \\ 1.0773(x-1) + 0.8413) & 1 \leq x < 3 \\ x & 3 \leq x \end{cases} \tag{3.6}$$

The curves of accurate gelu function and approximate gelu function are plotted in figure 3.6. Through the figure, we could find that approximate function curve is similar to actual function curve, which proves that the approximation is effective.

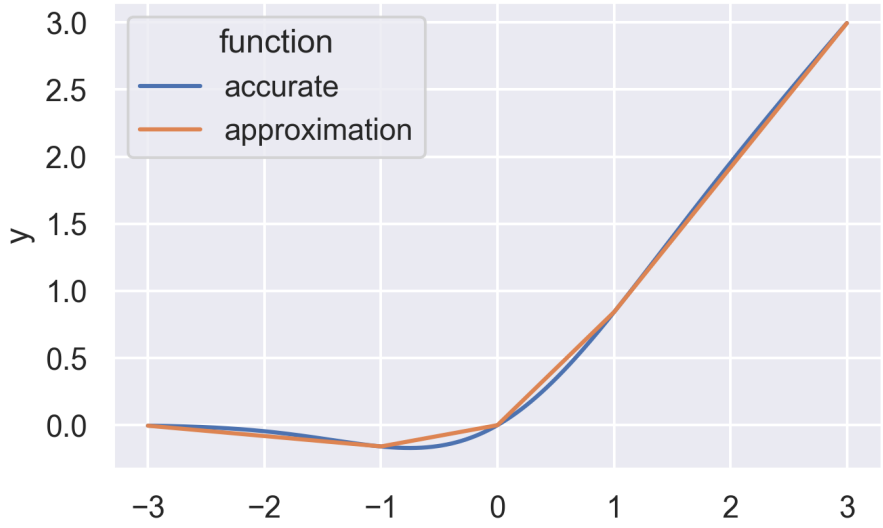

Figure 3.6: The plot of theory gelu function and approximated gelu function

CHAPTER 4

Evaluation

We implement T-OPU at 200MHz on the Xilinx Zynq Z-7100 FPGA, which has 2,020 DSP slices, 26.5 Mb RAM, and 277k LUTs. We examine FPGA utilization for each NVU variant separately, then show overall FPGA utilization of each of the six resulting T-OPU configurations. We calculate software-simulated inference times for BERT for these six configurations and compare them to the corresponding NVU-2048 reference inference time. Finally, we evaluate T-OPU's performance on BERT inference relative to other implementations'.

4.1 FPGA utilization

In Table 4.1, we individually show the FPGA utilization results for several components of the NVU: the NVU memory (NMEM), the vector register file (VRF), and the compute units (VCU and SCU). Then, in Table 4.2, we give the cumulative FPGA resource utilization for T-OPU using each NVU variant, both for 8-bit and 16-bit T-OPU.

Table 4.1: Overall FPGA resource Utilization on Zynq Z-7100 for T-OPU with 8-bit and 16-bit MMU with NVU-256, NVU-512, and NVU-1024

MMU	VRWIDTH	LUT	FF	DSP Slices	BRAM
8-bit	NVU-256	165776 (59.76%)	341151 (61.49%)	1994 (98.71%)	345 (45.70%)
8-bit	NVU-512	175701 (63.33%)	344385 (62.07%)	2002 (99.10%)	353 (46.75%)
8-bit	NVU-1024	192448 (69.37%)	351061 (63.28%)	2018 (99.90%)	369 (48.87%)
16-bit	NVU-256	129231 (46.59%)	250738 (45.19%)	1995 (98.76%)	502.5 (66.56%)
16-bit	NVU-512	139156 (50.16%)	253972 (45.78%)	2003 (99.16%)	510.5 (67.61%)
16-bit	NVU-1024	155903 (56.20%)	260648 (46.98%)	2019 (99.95%)	526.5 (69.73%)

From these results, we see that all the NVU variants are small relative to the overall T-OPU design. Even NVU-1024 uses less than three percent of overall flip-flop, DSP slice, and BRAM resources each. The larger NVU do use 7-15% of the overall LUT resources, much of which is due to the muxes required for shifting. Despite this, the overall design still has many LUTs left over.

Table 4.2: FPGA Resource Utilization for components of NVU-256, NVU-512, and NVU-1024 on Zynq-7100

Module	VRWIDTH	LUT	FF	DSP Slices	BRAM	F7 Mux	F8 Mux
NMEM	NVU-256	776 (0.28%)	1234 (0.22%)	0	4	0	0
VRF	NVU-256	156 (0.06%)	513 (0.09%)	0	4	0	0
VCU+SCU	NVU-256	10328 (3.72%)	1753 (0.32%)	8 (0.4%)	0	3 (<0.01%)	0
Total	NVU-256	**11260 (4.06%)**	**3500 (0.63%)**	**8 (0.4%)**	**8 (1.06%)**	**3 (<0.01%)**	**0**
NMEM	NVU-512	1330 (0.48%)	2268 (0.41%)	0	8 (1.06%)	0	0
VRF	NVU-512	306 (0.11%)	1025 (0.18%)	0	8 (1.06%)	0	0
VCU+SCU	NVU-512	19549 (7.05%)	3441 (0.62%)	16 (0.79%)	0	12 (<0.01%)	5 (<0.01%)
Total	NVU-512	**21185 (7.64%)**	**6734 (1.21%)**	**16(0.79%)**	**16 (2.1%)**	**12 (<-0.01%)**	**5 (<0.01%)**
NMEM	NVU-1024	2902 (1.05%)	4377 (0.79%)	0	16 (2.1%)	350 (0.25%)	0
VRF	NVU-1024	607 (0.22%)	2049 (0.37%)	0	16 (2.1%)	0	0
VCU/SCU	NVU-1024	34423 (12.41%)	6984 (1.26%)	32 (1.58%)	0	37 (0.03%)	5 (<0.01%)
Total	NVU-1024	**37932 (13.67%)**	**13410 (2.42%)**	**32 (1.58%)**	**32 (4.2%)**	**387 (0.28%)**	**5 (<0.01%)**

4.2 Inference time

The system simulation gives a cycle count estimate for a single inference of BERTBASE, which can be used to determine inference time given the operating clock speed. The relative inference times of T-OPU with 16-bit MMU and NVU-256, NVU-512, and NVU-1024 are compared to inference time with NVU-2048. For T-OPU with 16-bit MMU, NVU-2048 gives the ideal inference time because it always exceeds the MMU throughput.

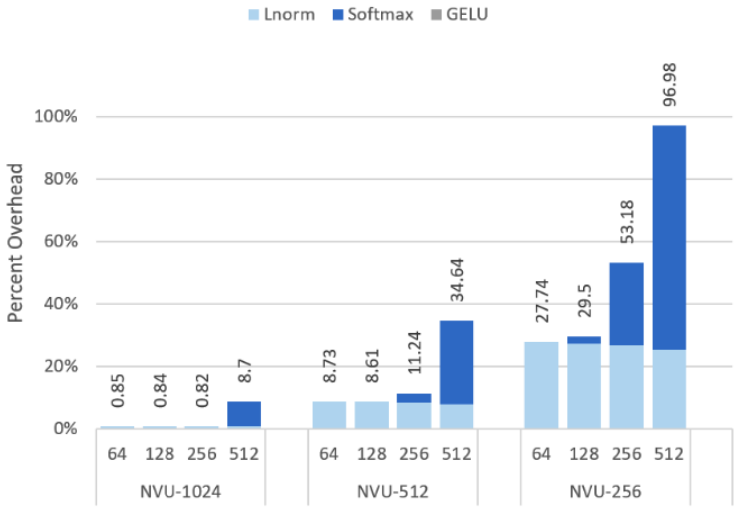

Figure 4.1: BERT inference time percent overhead using T-OPU with 16-bit MMU for different sequence lengths and NVU variants. Overhead is relative to the minimum time using NVU-2048

Figure 4.1 shows the percent inference time overhead of NVU of different for T-OPU with 16-

bit MMU. We see that in all cases, GELU does not add latency overhead for any sequence length. Overall, NVU-1024 has very little overhead compared to the baseline case. The small difference is because layer normalization throughput is slightly lower than that which is needed to match the MMU. For smaller sequence lengths, NVU-1024 adds less than 1% latency overhead, NVU-512 adds around 10%, and NVU-256 adds about 30%. Depending on the use case, these overheads may be acceptable given the reduced area costs. For higher sequence lengths, NVU-256 begins to show huge overheads of 53% and 97%. Note that inference time overhead alone is not the only criteria that should be used to evaluate these options. Even larger overheads may be acceptable, as long as the overall inference time including overhead is within the target for conversational AI. For this reason, the actual inference time is compared below.

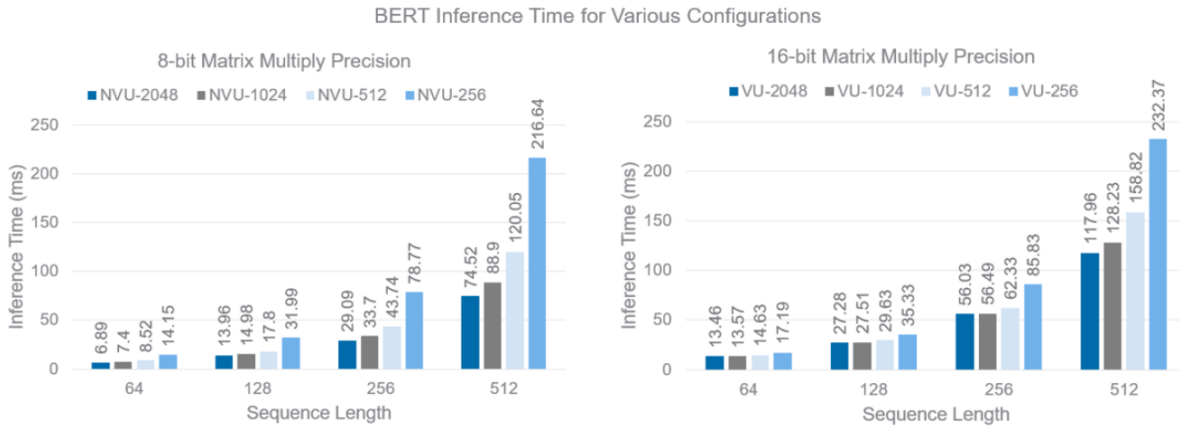

Figure 4.2: BERT inference time (in ms) with different NVU widths and sequence lengths. Results are shown separately for T-OPU with 8-bit and 16-bit matrix multiplies

The BERT inference time for T-OPU with 16-bit and 8-bit MMUs with each is shown in Figure 4.2. We see that T-OPU with 8-bit MMU can achieve sub-10 ms inference time with sequence length of 64 even with NVU-512, but that the inference time increases proportionally as sequence length increases. For typical applications, a sequence length of 64 is sufficient. For conversational AI, we require within 10-15 ms inference time, which we can clearly surpass with NVU-512 and NVU-1024 for both 8 and 16-bit.

4.3 Comparison with CPU, GPU, and FPGA

The authors of the FTRANS transformer FPGA accelerator [17] provide inference benchmarks by running RoBERTa, an optimized version of BERT with the same model architecture but trained more thoroughly. Since BERT and RoBERTa have the same architecture, we can compare our BERT accelerator's inference times with their RoBERTa benchmarks. We compare with our T-OPU with 16-bit and 8-bit MMUs with NVU-1024 on the Zynq Z-7100. The devices used in the

benchmark are an i7-8700k CPU, an RTX 5000 GPU, and an Ultrascale+ VCU118 FPGA (for FTRANS). The RTX 5000 has 1.52× more compute units than our Zynq FPGA and runs at 8.1× higher clock frequency. The VCU118 has 6,840 DSP slices and 2,586k logic cells (3.39× the DSP slices and 5.82× the logic cells on our board). The inference times and relative latencies are shown in Table 4.3. We also give the approximate power consumption of each device.

Table 4.3: Throughput (inference/sec) of NPE with NVU-1024 compared with CPU (i7-8700k), GPU (RTX 5000), and FPGA (VCU118). We also give relative throughput compared to FTRANS, throughput per DSP slice relative to FTRANS (for FPGA implementations), and approximate power consumption

	i7-8700k	RTX5000	FIRANS	T-OPU (16-bit)	T-OPU (8-bit)
Throughput	3.76	57.46	101.79	73.69	135.14
Relative Speedup	0.037x	0.56x	1x (baseline)	0.72x	1.33x
DSP Slices Utilized	-	-	6,840	2,020	2,020
Throughput per DSP	-	-	0.0148 (1x)	0.0365 (2.5x)	0.0669 (4.5)
Approximate Power (W)	80	120	25	20	20

From the results, we see that the CPU is far too slow for conversational AI. While the RTX 5000 GPU gets close, it does not meet the conversational AI latency targets. However, with a larger or more optimized GPU it could meet these requirements, albeit with much higher power consumption. Both FTRANS and T-OPU implementations stay within the range needed for conversational AI.

4.4 Comparison with CPU, GPU, and FPGA

The authors of the FTRANS transformer FPGA accelerator [17] provide inference benchmarks by running RoBERTa, an optimized version of BERT with the same model architecture but trained more thoroughly. Since BERT and RoBERTa have the same architecture, we can compare our BERT accelerator's inference times with their RoBERTa benchmarks. We compare with our T-OPU with 16-bit and 8-bit MMUs with NVU-1024 on the Zynq Z-7100. The devices used in the benchmark are an i7-8700k CPU, an RTX 5000 GPU, and an Ultrascale+ VCU118 FPGA (for FTRANS). The RTX 5000 has 1.52× more compute units than our Zynq FPGA and runs at 8.1× higher clock frequency. The VCU118 has 6,840 DSP slices and 2,586k logic cells (3.39× the DSP slices and 5.82× the logic cells on our board). The inference times and relative latencies are shown in Table **??**. We also give the approximate power consumption of each device.

4.5 Benchmarks Discussion

The biggest benefit of an FPGA implementation of BERT over CPU and GPU is with power consumption. From Table **??**, we see about a 4× power benefit over CPU and 6× over GPU. This difference in power consumption is especially important for NLP processing on edge devices. While FTRANS and T-OPU both have comparable performance and power, FTRANS uses over 3× more resources than T-OPU since it uses a much larger FPGA. We attribute some difference in resource consumption to the fact that FTRANS uses specialized modules for each transformer and each nonlinearity, which leads to additional area and under-utilized components.

In this paper, we propose T-OPU, an FPGA-based overlay processor that is domain-specialized for Natural Language Processing. T-OPU offers software-like programmability and provides a unified framework to process arbitrarily complex nonlinear functions. If a new state-of-the-art NLP model were to surpass transformers in the coming years, T-OPU is most likely flexible enough to adapt to it without requiring reconfiguring the FPGA accelerator or adding specialized processing modules. T-OPU can also meet the inference latency requirements for conversational AI for the BERT language model. Relative to CPU and GPU, T-OPU has 4× and 6× lower power consumption respectively. Our accelerator shows comparable performance to a transformer model specialized FPGA accelerator, but T-OPU uses 3× lower FPGA resources. Overall, we find that T-OPU is a promising solution for low-cost and low-power NLP network inference at the edge.

CHAPTER 5

Conclusion

In this paper, we propose T-OPU, an FPGA-based overlay processor that is domain-specialized for Natural Language Processing. T-OPU offers software-like programmability and provides a unified framework to process arbitrarily complex nonlinear functions. If a new state-of-the-art NLP model were to surpass transformers in the coming years, T-OPU is most likely flexible enough to adapt to it without requiring reconfiguring the FPGA accelerator or adding specialized processing modules. T-OPU can also meet the inference latency requirements for conversational AI for the BERT language model. Relative to CPU and GPU, T-OPU has 4× and 6× lower power consumption respectively. Our accelerator shows comparable performance to a transformer model specialized FPGA accelerator, but T-OPU uses 3× lower FPGA resources. Overall, we find that T-OPU is a promising solution for low-cost and low-power NLP network inference at the edge.

REFERENCES

[CMS] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. “End-to-End Object Detection with Transformers.”.

[DBK] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale.”.

[DCL] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.”.

[GZY] Kaiyuan Guo, Shulin Zeng, Jincheng Yu, Yu Wang, and Huazhong Yang. “A Survey of FPGA-Based Neural Network Accelerator.”.

[KTR] Ian Kuon, Russell Tessier, and Jonathan Rose. *FPGA Architecture: Survey and Challenges*. Now Publishers Inc. Google-Books-ID: AdK2OWDP7L0C.

[VSP] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. “Attention Is All You Need.”.

[YWZ] Yunxuan Yu, Chen Wu, Tiandong Zhao, Kun Wang, and Lei He. “OPU: An FPGA-Based Overlay Processor for Convolutional Neural Networks.” **28**(1):35–47. Conference Name: IEEE Transactions on Very Large Scale Integration (VLSI) Systems.