Accuracy-preserving Layer Normalization Approximations for Efficient Transformer Hardware Accelerators

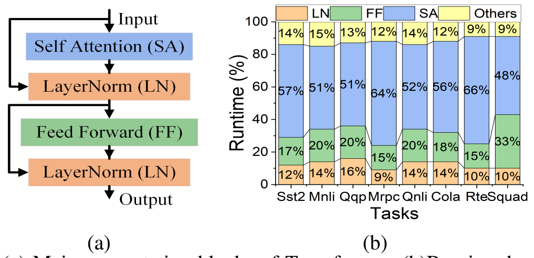
Nazim Altar Koca[*†], Anh Tuan Do[*] and Chip Hong Chang[†]
[*] Institute of Microelectronics, A*STAR (Agency for Science, Technology and Research), Singapore
[†] School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore

***Abstract*—Transformers have emerged as leading models for solving natural language processing (NLP) problems. Layer normalization (LN) has been found to be a throughput and latency bottleneck for Transformer networks. The LN datapath involves sequential processing that is dependent on the input data, and GPU and CPU unfriendly nonlinear square root and reciprocal operations. These make pipelining and hardware simplification challenging without compromising the accuracy of pretrained models. In this paper, we propose a hardware-efficient and accuracy-preserving design for LN approximation. The LN core can be directly plugged into pretrained Transformer networks to accelerate NLP tasks without fine tuning. The FPGA implementation of our design outperforms the state-of-the-art FPGA implementation of LN, with 3.5× higher throughput per LUT and 1.3× higher throughput per DSP. Our approximation introduces negligible average and worst-case accuracy drops of only 0.28% and 1.09%, respectively on the main language benchmark tasks. Additionally, a better pipelined design with pairwise variance calculation is proposed to reduce the number of iterations for LN, resulting in a further 27% reduction in latency with a slight increase in hardware resource requirement.**

I. Introduction

Natural Language Processing (NLP) is among the hottest pursuits in the field of data science nowadays. It finds applications in speech recognition (e.g., Apple Siri), machine translation (e.g., Google Translation), chatbots (e.g., chatGPT), etc. Transformers [1] have emerged as the choice learning model for NLP. Pre-trained Transformers (such as BERT [2] and Distilbert [3]) achieved state-of-the-art accuracy in NLP. They can operate without being constrained by the relative positioning among tokens within an input-output sequence. The elimination of sequence-aligned processing provides more parallelization and reduces the training time. Nevertheless, Transformer has its own implementation challenges due to the large model size, nonlinear layers, and computational complexity. Fig. 1(a) shows the main computation blocks of a basic Transformer. Self-attention and feed forward layers consist of large matrix multiplications as well as nonlinear operations like Softmax and Gaussian error linear unit (GeLU). The layer normalization (LN) block comes after (or before in recent models) every self attention and feed forward blocks to normalize the activities of the inputs, which contributes to the overall latency of the system. Self attention and feed forward layers in Transformer has been substantially sped up by recent advancement [4] [5], but efficient hardware implementation of LN towards accuracy-preserving inference at the edge is still lacking.

For the LN of Transformer, the mean and variance values need to be recomputed based on the inputs at runtime as hidden dimension statistics change continuously. It is reported in [6] that although the number of raw mathematical operations that accounts for LN is around 0.1%, the GPU execution time of LN is nearly 10% for GPT2 network inference. This implies that GPUs are not optimized for these sequential and nonlinear operations. We conduct experiments on GPU specifically to show the runtime latency of different computation blocks of Distilbert Transformer model for GLUE [7] and SQuAD [8] benchmark tasks during inference in Fig. 1(b). It can be seen that the LN takes up between 9-16% of the runtime. With the recent advancement in efficient matrix multiplication hardware for linear operations, the performance bottleneck of nonlinear layers becomes even more dominant. LN is hardware inefficient due to the square root and division operations. It is also time consuming since it needs three iterations on the input data over the hidden dimension (usually consists of at least 512 elements). The first iteration is needed for calculating the mean value, the second iteration for the variance and



(a) (b)

Fig. 1. (a) Main computation blocks of Transformer. (b)Runtime breakdown of Transformer's components for 7 GLUE tasks and the SQuAD task.

the last iteration for the normalization. These statistical values need to be calculated online during both the training and inference phases.

This paper presents two efficient hardware architectures that approximate LN with minimal accuracy loss for NLP tasks without requiring fine tuning. Our main contributions are: 1) We identify the runtime issue of LN on inference machine for popular GLUE and SQuAD tasks; 2) We explore the design space of data quantization and parallelism for optimal area-time trade off; 3) We significantly reduce the hardware complexity of the nonlinear functions of LN by piecewise linear approximation while preserving the accuracy of pretrained model; 4) We propose a fully pipelined architecture based on pairwise variance calculation to further reduce the latency of LN incurred by the three data-dependent iterations on the input data. To the best of our knowledge, this is the first efficient custom hardware implementation of the pairwise variance algorithm; 5) We benchmark our proposed designs against the state-of-the-art FPGA implementations of LN to demonstrate their performance and hardware efficiency.

II. Preliminaries and Related Works

A. Layer Normalization

Normalization improves convergence and stability of the optimization process by reducing the sensitivity of the neural network model to variations in the input data. As the input data distribution of each layer changes during training, the layers need to continuously adapt to the new distribution, which slows down the training process. For a fast and stable training, the activations should preferably be Gaussian distributed with zero mean and unit variance. LN was introduced in 2016 [9]. The LN function is defined as follows:

$$LN(x_{i,j}) = \widehat{x_{i,j}} \cdot \gamma_j + \beta_j \quad \text{for } j = 1, 2, \ldots, d_{model} \tag{1}$$

where $\widehat{x_{i,j}}$ is calculated by (2). γ and β are the scaling and shifting factors, respectively, to allow the network to modify the distribution when necessary. They are different for each hidden dimension.

$$\widehat{x_{i,j}} = \frac{x_{i,j} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} \quad \text{for } i = 1, 2, \ldots, d_{token} \tag{2}$$

In (2), ϵ is a small fixed value added to avoid division by 0. The mean (μ) and variance (σ^2) are calculated over the hidden dimension for every token by (3).

$$\mu_i = \frac{1}{d_{model}} \cdot \sum_{j=1}^{d_{model}} x_{i,j}, \quad \sigma_i^2 = \frac{1}{d_{model}} \cdot \sum_{j=1}^{d_{model}} (x_{i,j} - \mu_i)^2 \tag{3}$$

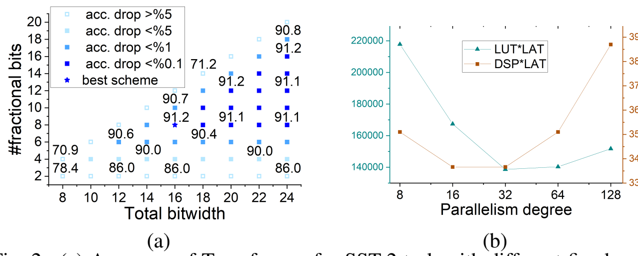


Fig. 2. (a) Accuracy of Transformer for SST-2 task with different fixed-point representations; (b) Area-time products vs degree of parallelism.

B. *Hardware Acceleration of Normalization Layers*

One main problem in the efficient hardware implementation of LN is the precision of nonlinear operations. Recently, the square root function of LN was replaced with integer only approximation [10]. For a given non-negative integer n, the algorithm finds $\sqrt{n}$ based on Newton's method that requires only integer operations. As the algorithm is iterative, this method does not prevent the delay from accumulating onto the critical path before the final result is calculated. In [11], nonlinear operations in Transformer were replaced with LUT based computations for ASIC implementation. This is achieved by using a one-hidden-layer ReLU neural network as a universal approximator to nonlinear operations of Transformer and the network parameters are stored in a LUT. As a result, only LUTs, a multiplier and an adder are required for its implementation. The authors showed that their method does not compromise the accuracy by evaluating it on RoBERTa network. However, this method requires domain expertise to design the universal neural network approximator to nonlinear operations for a new domain. In [12], the long latency of LN block on FPGA implementation was reduced by a one-pass variance algorithm for the variance calculation as follows.

$$\sigma_i^2 = \frac{1}{d_{model}} \cdot \sum_{j=1}^{d_{model}} x_{i,j}^2 - \mu_i^2 \tag{4}$$

According to this algorithm, x^2 can be calculated concurrently with the mean value. Consequently, the number of iterations for variance calculation is reduced from two to one. However, several papers reported that this algorithm is numerically unstable due to the precision errors [13] [14]. Inspired by the reduced precision calculation of [15], a new LN approximation was introduced in [16]. It uses a division-free technique by approximating the reciprocal square root function and fixed point operations. However, this method needs fine tuning of the network after making the approximation.

Lastly, LN acceleration aims to accelerate the mean and standard deviation calculation. Therefore, design methodologies that enable efficient LN hardware implementation will also benefit applications that require online statistical analysis. Unfortunately, hardware optimization of these nonlinear computer arithmetic functions is limited in existing literature.

III. Proposed LN Hardware Accelerators

A. *Design Space Exploration of Quantization and Parallelism*

To approximate the LN operation, design space for quantization is explored. An experiment was performed on Distilbert Transformer model for SST-2 tasks to find an optimal fixed-point representation with minimal accuracy drop for quantizing the input vector and intermediate operations of the LN block from 32-bit floating-point representation. Different combinations of 8-bit to 24-bit fixed-point representations with different bitwidths of integer and fractional parts were evaluated iteratively. The final average accuracy for each fixed-point representation is plotted in Fig. 2(a). It shows that the 16-bit fixed-point representation with 8 integer bits and 8 fractional bits gives the best solution. This quantization scheme was further validated on other GLUE and SQuAD tasks to also produce the optimal results.

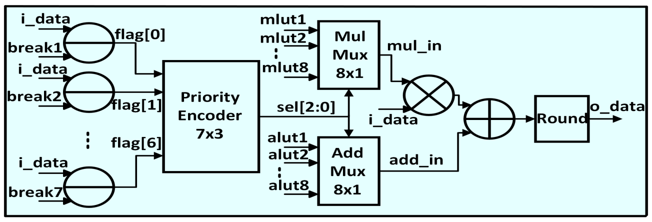


Fig. 3. Common architecture for PWL approximation of square-root and reciprocal functions.

Another experiment was carried out to explore the design space for parallelism. The standard LN module was instantiated for the realization of (2) with 8, 16, 32, 64 or 128 inputs. For the ease of exposition, we refer to the number of inputs that can be processed by a LN module at any one time as the degree of parallelism. A more complex design with a higher degree of parallelism will require less latency than a less complex design with a lower degree of parallelism to complete the calculation for the same total number of inputs. The latency required to process 512 inputs of the original Transformer network [1] is evaluated for each design option. LUTs and DSPs are two costly computation blocks of FPGAs in terms of area cost. Both LUT-latency product and DSP-latency product on Xilinx ZCU102 are evaluated against the degree of parallelism in Fig. 2(b). The best area-time trade-off is found to be the 32-input design.

B. *Piecewise Linear Approximation of Nonlinear Operations*

Square Root and Reciprocal functions are the two nonlinear functions that need to be minimized for efficient computation of LN. Integer-only approach to square root calculation was reviewed previously. However, it introduces additional delays into the critical path due to the iterative algorithm. Here the inherent error tolerance of neural networks is exploited for more efficient approximation, particularly to reduce the latency. Since the input of the square root function is a positive integer after the variance calculation, its value ranges between 0 and 128 for the signed 16-bit fixed-point representation with 8 integer bits. For the reciprocal operation that comes after the square root, the input range is between 0 and $\sqrt{128}$. By restricting the input and output ranges to these intervals, the nonlinear functions can be more aggressively and accurately approximated with piecewise linear (PWL) functions.

The hardware complexity for PWL approximation increases with the number of breakpoints. Although it is easier to evenly divide the input range of the function into equal subranges for linear interpolation, it requires more breakpoints than the non-uniform interpolation to preserve the network accuracy. To reduce the storage and computational costs, we use the differential evolution algorithm PWLF [17] of Python library to search for the minimum number of non-uniform breakpoints for accuracy-preserving linear interpolation with the given precision of the input/output ranges. Several tests have been performed to further confirm the effect of the PWL with the chosen optimal number of breakpoints on the accuracy of the network. It turns out that both the square root and reciprocal functions can be approximated by a PWL with only 7 non-uniform breakpoints (8 line segments) without compromising the network accuracy.

The proposed hardware architecture for approximating the nonlinear function using PWL with these seven optimal breakpoints is shown in Fig. 3. Unlike the uniform PWL, the domain of each line segment cannot be directly identified by simply checking the most significant bits of the input. To determine the correct line segment for an input value, the seven precalculated breakpoints are subtracted from the input (I_{data}) by seven parallel subtractors. If I_{data} falls

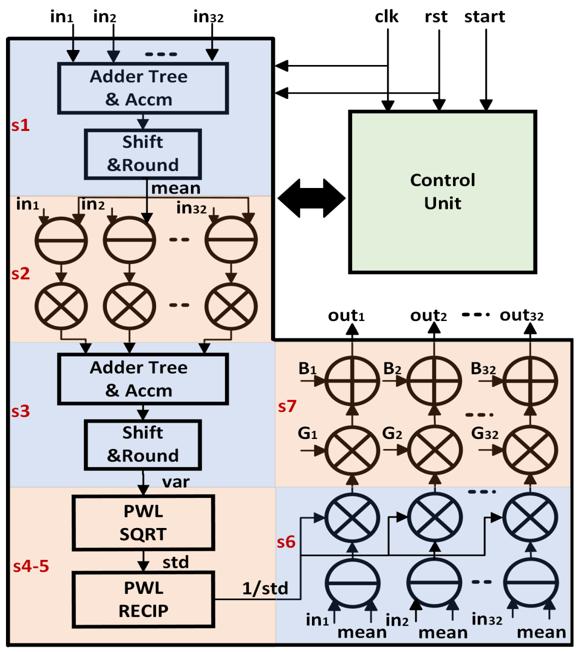


Fig. 4. Approximate standard LN architecture

above any of the breakpoints, the sign bit of its corresponding subtractor will be asserted. These sign-bit flags are fed into a priority encoder to locate the line segment where I_{data} resides. The encoder output sel[2:0] is used as a select signal to the Mul multiplexer and Add multiplexer to retrieve the corresponding precalculated gradient and intercept, respectively of the line segment. Finally, a multiplier and an adder are used to calculate the linearly interpolated value of I_{data}. The overflow due to the multiply-and-add operation is handled by a half round up operation to obtain a 16-bit result from a truncated fixed-width adder. This non-uniform PWL design can calculate the result in only one cycle and the detailed hardware comparison is provided in Section IV-B. This design allows the same architecture to be used for both square-root and reciprocal function approximations with different precalculated gradients and intercepts applied to mlut1-mlut8 and alut1-alut8 inputs of the multiplexers, respectively in Fig. 3.

C. Standard Layer Normalization

The LN hardware architecture is depicted in Fig. 4. It consists of seven pipeline stages indicated by s1 to s7. An FSM is used to control the data flow in the data-dependent iterations. The total number of 512 inputs are presented to the hardware in batches. Each batch consists of 32 elements (see Fig. 2(b)) and each element is represented in 16-bit fixed-point representation (see Fig. 2(a)). An adder tree/accumulator and shifter are used to calculate the mean value in s1. DSP multipliers are used for fast squaring operations in s2. The variance is calculated in s3. The square root and reciprocal functions are implemented with the proposed non-uniform PWL circuit in s4-s5. The final normalization is implemented in s6 and the scaling/shifting operations are performed in s7. To avoid precision loss in the variance calculation, higher bitwidth adders and accumulator are used in s3 than those in s1. Rounders are used to obtain the final mean and variance, and after the multipliers in s6 and s7. As the LN is parallelized at token-level, this hardware module can be easily scaled up by replication to process the token elements in parallel. To handle more input data (elements in hidden dimension), as seen in recent Transformers, changing a few parameters in the control unit is sufficient. However, in this design, s2 must wait for the result of s1 until all inputs are presented, which prevents us from implementing a fully pipelined architecture (see

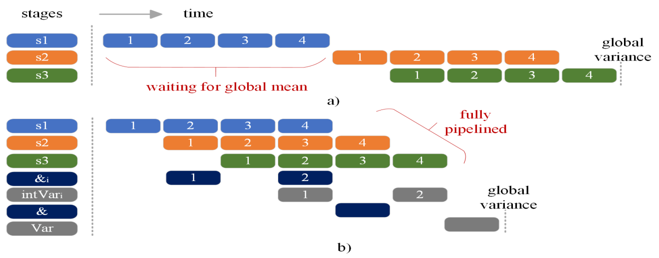


Fig. 5. Exemplified timing diagrams for a) standard variance b) pairwise variance calculation.

Fig. 5(a)). We present a solution to overcome this problem in the next section.

D. Pairwise Variance Algorithm

We revisited the unstable one-pass variance algorithm described in Section II-B. A numerically equivalent and safer Pairwise Variance algorithm was found in an old paper [14]. According to this algorithm, the input data can be split into groups and the variance of each group can be calculated separately. These intermediate values are used to calculate the global variance. This algorithm requires $O(\log(n))$ memory complexity due to the intermediate values [14], whereas the standard implementation has a constant complexity $O(1)$. However, due to its parallelizability, it requires only a single pass on the input data. Modern database systems like Spark and Impala leverage this algorithm for a fast and accurate variance calculation. As an example, the formulae for pairwise variance calculation with four groups of data are given as follows:

$$\delta_{1(2)} = \mu_{1(3)} - \mu_{2(4)}, \quad \delta = \frac{\mu_1 + \mu_2}{2} - \frac{\mu_3 + \mu_4}{2} \tag{5}$$

$$intVar_{1(2)} = \hat{\sigma}^2_{1(3)} + \hat{\sigma}^2_{2(4)} + \delta^2_{1(2)} \times \frac{n_{1(3)} \times n_{2(4)}}{n_{1(3)} + n_{2(4)}} \tag{6}$$

$$\hat{\sigma}^2 = intVar_1 + intVar_2 + \delta^2 \times \frac{(n_1 + n_2) \times (n_3 + n_4)}{n_1 + n_2 + n_3 + n_4} \tag{7}$$

where μ_i is the mean and $\hat{\sigma}^2_i$ is the variance of the i-th group of elements, and n_i is the number of elements in the i-th group. $\hat{\sigma}^2$ is the global variance. To avoid rewriting the same equation involving different groups twice, two equations are written as one equation in (5) and (6). Each equation can be obtained by taking only the unbracketed or bracketed group numbers separately as the subscripts of the variables.

This algorithm allows the number of iterations of LN on input data to be reduced to two with a dedicated hardware implementation. An efficient scheduler is shown in Fig. 5(b), where s1, s2 and s3 represent 3 pipeline stages. With the pairwise algorithm, there is no need to wait for the global mean value to be completed. Therefore, the accumulators after the adder tree can be removed. To save the registers required to latch the current batch of inputs, unlike the standard LN architecture, we move the subtractors from Stage s2 to s1. As a result, we are able to fully pipeline s1-s3 in Fig. 5(b). Also, the values of δ and *intVar* can be calculated in parallel. This way, the latency of one input iteration (4 cycles in this example) can be saved by introducing only one more clock cycle to calculate the global variance (*Var*). This illustrative design example is expanded to handle 512 inputs by processing 32 inputs at one time in 16 groups. As the number of groups increases, more latency can be saved with increased hardware parallelism and scheduling complexity. There is no change to the remaining hardware components of s4-7. The hardware results of this algorithm are presented and compared with the standard LN accelerator in Section IV-C.

TABLE I
ACCURACY RESULTS FOR NLP TASKS WITH OUR PROPOSED LN APPROXIMATION METHODS

		SQuAD	SST2	QNLI	MNLI	MRPC	QQP	RTE	CoLA
Bert	Baseline	88.22	92.43	91.54	84.45	84.55	90.90	72.56	53.38
Base	Proposed	88.17	92.08	91.52	84.16	84.31	90.69	72.20	54.43
Distil-	Baseline	84.41	91.05	88.48	82.12	85.78	89.92	64.98	56.85
Bert	Proposed	84.05	91.16	88.01	81.90	86.51	89.96	63.89	55.89

TABLE II
COMPARISON OF THE FIXED-POINT SQRT AND RECIP IMPLEMENTATIONS

Design	LUT	REG	DSP	Latency
Sqrt IP	286	183	0	9
Div IP	92	154	0	6
PWL Ours	46	20	1	**1**

IV. EXPERIMENTAL RESULTS

A. Impact of LN Approximation on Accuracy

We test the accuracy of the two widely used Transformer models BERT [2] and DistilBERT [3] from Huggingface libraries for GLUE [7] and SQuAD [8] benchmark problems by substituting the LN block of the pretrained model with our proposed LN approximation. DistilBERT is a lightweight, knowledge distilled version of the original BERT. It reduces the parameter size of the network by 40% while keeping the accuracy loss low on the benchmark tasks. Therefore, it is suitable for evaluating the performance of our approximation on mobile networks. GLUE and SQuAD are the main benchmark tasks for discriminative language models. GLUE consists of several subtasks (SST2, QNLI, etc.), each of which tests a different language understanding capability of the network. Table I shows the results of the original 32-bit floating-point LN (Baseline) and the proposed LN (Proposed) for different tasks. The results show that the proposed approximation introduces an average accuracy drop of only 0.06% for BERT and 0.28% for DistilBERT and 1.09% at most for all GLUE tasks and SQuAD tasks without fine tuning. As BERT is over-parameterized compared to DistilBERT, it is more resilient to the approximation errors as expected. Since both our proposed approximation methods (standard LN and pairwise LN) are numerically equal, only one result is presented for both proposed methods in this table.

B. Square Root and Reciprocal Functions

The proposed hardware modules are described by Verilog code in the Vivado design environment and implemented on the FPGA chip of the Xilinx ZCU102 development board. The golden test vectors are generated in Python for functional verification. The PWL design detailed in Section III-B can run at 200 MHz clock frequency. The same module can perform either square root (sqrt) or reciprocal (recip) functions as described in Section III-B. Its resource utilization and latency are evaluated and compared with the implementations using Xilinx Repository IPs. A 16-bit fixed point square root IP core that utilizes the CORDIC algorithm is used for the evaluation. Since there is no IP core for the reciprocal function in Xilinx IP repository, it is implemented by instantiating a 16-bit radix-2 fixed-point divider IP core with the dividend value fixed to a constant one. The results are shown in Table II, the proposed implementation uses only one DSP and a significantly reduced number of LUTs and registers. It calculates the result in only one cycle as opposed to 6 for Div and 9 for Sqrt using the Xilinx IP cores.

C. Complete LN Hardware Accelerators

Table III compares the hardware resources and latencies of three different LN accelerator designs implemented on the same FPGA chip running at 200 MHz clock frequency. The $Baseline$ design in Table III refers to the standard LN design implemented by the 32-bit Xilinx Floating Point IPs with its accuracy given in Table I. The resources and latencies consumed by the two proposed approximate LN accelerator implementations based on the standard and pairwise variance calculation algorithms, respectively are compared with this baseline design. The latency refers to the number of clock cycles needed to process all the 512 elements of the input vector. From Table III, the proposed LN accelerator with the standard variance calculation has reduced the latency of the baseline design by almost 3×. The proposed design with the pairwise variance calculation has reduced the latency of the baseline design by 4×, and the latency of the proposed standard LN design by 27% at the expense of using more hardware resources.

TABLE III
RESOURCE UTILIZATION AND LATENCY OF THE BASELINE AND PROPOSED LN HARDWARE ACCELERATORS

Design	LUT	REG	DSP	Latency
Baseline	91651	40645	205	153
Proposed Standard LN	2945	645	98	**52**
Proposed Pairwise LN	3582	2160	102	**38**

TABLE IV
PERFORMANCE COMPARISON ON DIFFERENT LN HARDWARE IMPLEMENTATIONS

Design/ Platform	Method	P/W	LUT/ REG	DSP/ BRAM	Freq.(MHz) Power(W)	TPL/ TPD
SOCC'20 [12]	One-Pass	64/8	10551	129	200	9.705
XCVU13P	LN		5325	24.5	NA	793.79
ISLPED '24 [16]	Peano	16/8	8157	52	250	3.923
VU9P	LN		8621	0	NA	615.38
Ours I	Standard	32/16	2945	98	200	**34.770**
ZCU102	LN		645	0	0.890	**1044.89**
Ours II	Pairwise	32/16	3582	102	200	**28.587**
ZCU102	LN		2160	0	0.922	**1003.92**

D. Comparison With Related Works

We compare our proposed LN accelerators against the reported FPGA implementations (Section II-B) of one-pass variance [12] and Peano [16] for LN in Table IV. To account for the different degrees of parallelism P, the operator bitwidth W, and the working frequency f_{clk} (MHz) across designs, a better comparison is provided by the following figures of merit *TPL* (Throughput per LUT) and *TPD* (Throughput per DSP) listed in the last column of Table IV for the average number of bits per second processed by each unit of used resource :

$$\mathit{TPL(TPD)} = \frac{f_{clk} \times P \times W}{\mathit{LUT(DSP)}} \tag{8}$$

Our design has the highest *TPL* and *TPD* among all designs in comparison.

V. CONCLUSION

In this paper, two new approximate LN hardware accelerators for Transformer network are proposed with negligible accuracy degradation. By exploring the design space for quantization and parallelism, our first design replaces complex nonlinear and floating-point operations by simpler arithmetic operators and small lookup tables. Since it does not require retraining to restore the model accuracy, this LN core can be used as a direct drop-in replacement for different pretrained Transformer models. The proposed design is evaluated to be the most hardware-efficient among existing FPGA implementations of normalization algorithms. We also proposed a faster pipelined architecture using pairwise variance calculation with slightly more hardware resources to further reduce the latency of the first design by another 27%.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. NIPS*, 2017, p. 6000–6010.

[2] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805

[3] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," 2020.

[4] T. Yang, D. Li, Z. Song, Y. Zhao, F. Liu, Z. Wang, Z. He, and L. Jiang, "Dtqatten: Leveraging dynamic token-based quantization for efficient attention architecture," in *Proc. DATE*, 2022, pp. 700–705.

[5] H. Peng, S. Huang, S. Chen, B. Li, T. Geng, A. Li, W. Jiang, W. Wen, J. Bi, H. Liu, and C. Ding, "A length adaptive algorithm-hardware co-design of transformer on fpga through sparse attention and dynamic pipelining," in *Proc. DAC*, 2022, p. 1135–1140.

[6] S. Hong, S. Moon, J. Kim, S. Lee, M. Kim, D. Lee, and J.-Y. Kim, "Dfx: A low-latency multi-fpga appliance for accelerating transformer-based text generation," in *IEEE/ACM MICRO*, 2022, pp. 616–630.

[7] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," *CoRR*, vol. abs/1804.07461, 2018.

[8] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100, 000+ questions for machine comprehension of text," *CoRR*, vol. abs/1606.05250, 2016.

[9] J. Lei Ba, J. R. Kiros, and G. E. Hinton, "Layer Normalization," *arXiv e-prints*, p. arXiv:1607.06450, Jul. 2016.

[10] S. Kim, A. Gholami, Z. Yao, M. W. Mahoney, and K. Keutzer, "I-bert: Integer-only bert quantization," *ICML*, 2021.

[11] J. Yu, J. Park, S. Park, M. Kim, S. Lee, D. H. Lee, and J. Choi, "Nn-lut: Neural approximation of non-linear operations for efficient transformer inference," in *Proc. DAC*, 2022, p. 577–582.

[12] S. Lu, M. Wang, S. Liang, J. Lin, and Z. Wang, "Hardware accelerator for multi-head attention and position-wise feed-forward in the transformer," in *IEEE SOCC*, 2020, pp. 84–89.

[13] E. Schubert and M. Gertz, "Numerically stable parallel computation of (co-)variance," in *Proc. SSDBM*. ACM, 2018.

[14] T. F. Chan, G. H. Golub, and R. J. LeVeque, "Updating formulae and a pairwise algorithm for computing sample variances," in *COMPSTAT 1982 5th Symposium*, 1982, pp. 30–41.

[15] W. Wang, S. Zhou, W. Sun, P. Sun, and Y. Liu, "Sole: Hardware-software co-design of softmax and layernorm for efficient transformer inference," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2023, pp. 1–9.

[16] M. E. Sadeghi, A. Fayyazi, S. Azizi, and M. Pedram, "Peano-vit: Power-efficient approximations of non-linearities in vision transformers," in *Proceedings of the 29th ACM/IEEE International Symposium on Low Power Electronics and Design*, ser. ISLPED '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 1–6. [Online]. Available: https://doi.org/10.1145/3665314.3670843

[17] C. F. Jekel and G. Venter, *pwlf: A Python Library for Fitting 1D Continuous Piecewise Linear Functions*, 2019.