Calabash: Accelerating Attention using a Systolic Array Chain on FPGAs

Zizhang Luo* — *School of Integrated Circuits, Peking University* — semiwaker@pku.edu.cn

Liqiang Lu* — *College of Computer Science and Technology, Zhejiang University* — liqianglu@zju.edu.cn

Yicheng Jin — *Trinity College of Arts and Sciences, Duke University* — yicheng.jin@duke.edu

Liancheng Jia — *School of Computer Science, Peking University* — jlc@pku.edu.cn

Yun Liang† — *School of Integrated Circuits, Peking University* — ericlyun@pku.edu.cn

***Abstract*—In recent years, attention mechanism has achieved remarkable performance in natural language processing and computer vision applications, at the expense of high computation cost. FPGAs have been demonstrated to be an effective hardware platform for various AI applications. However, the attention mechanism involves complex data dependency, which makes FPGA acceleration difficult. In this paper, we propose Calabash, an FPGA accelerator for attention-based applications. We design a chain of two systolic arrays, applying the same dataflow. Then, we design two scheduling techniques for different matrices to ensure the intermediate matrix can be cached in the on-chip memory. Finally, we develop analytical models for resource utilization estimation, workload balancing, and latency prediction to guide design space exploration. Experiments show that Calabash achieves 1.76 TOP/s, 1.06 TOP/s on Xilinx VU9P and ZCU102 platforms, yielding an average 50.1X and 3.94X energy-efficiency improvement compared with CPU and GPU, respectively.**

I. Introduction

Attention mechanism is an emerging neural network primitive that allows the model to focus on relevant parts of the input (either text or image) when producing an output element. Recent studies have demonstrated that attention mechanism can be applied to various AI applications, such as computer vision [19], natural language processing [25], [4], [22], [11]. For example, Transformer is a well-known network architecture proposed by Google that leverages the attention mechanism, and has achieved remarkable improvement in the translation task [25].

A myriad of novel architectures have been proposed under the umbrella of the transformer family since its sensational debut. The core of transformers is a variant of attention mechanism, namely scaled dot-product attention. As shown in Figure 1, its computation can be divided into two stages. In the first stage, the input sequence is multiplied with a set of weight matrices to generate three matrices (query, key, and value). In the second stage, we calculate the score matrix by multiplying the query matrix and the key matrix, divide it by a constant and then apply row-wise softmax function. Finally, we multiply the resultant matrix of normalized probability with the value matrix to generate the final output. In addition, the Transformer paper further proposes multi-head attention where several heads run through attention mechanism in parallel [25], which allows the model to jointly attend to information from representation subspaces at different positions. We depict the multi-headed case at the bottom of Figure 1.

Recently, FPGAs have emerged as an efficient platform for various AI applications thanks to their high performance, energy efficiency, and flexibility [21], [5], [26], [24], [14], [13], [28], [18], [29], [16], [17]. Prior FPGA designs for AI applications primarily focus on accelerating a single kernel such as matrix multiplication and convolution [3], [9], [10]. However, attention mechanism invokes multiple kernels with complex data dependency, which poses new challenges for FPGA acceleration. First, attention mechanism involves a general-purpose matrix multiplication chain (GEMMc), where three input matrices (Q, K, V) are multiplied together to generate an output matrix $(Z = softmax(Q \times K^T) \times V)$. Moreover, the intermediate result needs to conduct a softmax function row by row before the next matrix multiplication. It is very challenging to coordinate the execution of all these kernels and ensure the data transfer between different kernels goes smoothly. The second challenge comes from the large matrices involved in the inference of attention-based models. For example, BERT-base model [4] has a typical matrix size of 512×768, which requires more than 20MB to store all the necessary matrices of a single layer. As the block RAM size of FPGAs is usually less than 10MB, input matrices need to be partitioned into tiles. However, it is difficult to determine a partition scheme that is the most amenable for the architecture.

In this paper, we propose Calabash, an efficient architecture for accelerating attention-based applications on FPGAs. We first propose a systolic array chain to conduct the GEMMc operation. To avoid extra data movement and rearrangement, we carefully design the two systolic arrays to share the same dataflow. More clearly, they both use output stationary (OS) dataflow, where data are accessed in a row-major pattern for both input and output. Besides, Calabash supports the softmax function by inserting multiple EXP modules that approximate the exponential operator based on look-up tables. We also propose a partition scheme that partitions the key and query matrix in row dimension, and partitions the value matrix in column dimension. We use different scheduling

* These authors contributed equally.

† Corresponding Author

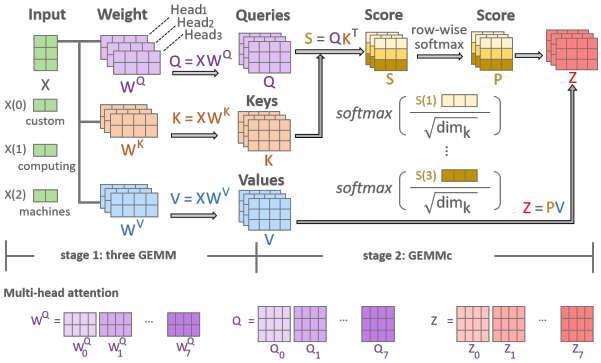

Fig. 1. Computation stages in the attention mechanism.

techniques for different matrices. This helps to ensure that data layout is consistent during the on-chip and off-chip data communication. Finally, we build analytical models to estimate various performance metrics and rely on them to determine the optimal parameters (e.g., partition factors, systolic array size) to minimize the total latency.

Our contributions are summarized as follows,

- We propose Calabash, an efficient architecture to accelerate attention on FPGAs. Calabash employs a systolic array chain to accelerate the GEMMc.
- We propose a partition method to partition the matrices into tiles to reduce the memory requirement and also propose a scheduling scheme for the partitioned tiles to keep the data layout consistent during data transfer.
- We develop an analytical model to predict the resource utilization and latency. And we use this model to find the optimal design with minimum latency.

We perform rigorous validation of our techniques using the state-of-the-art transformer-based models, including BERT [4], GPT-2 [22], BART [11]. Experiments demonstrate that Calabash achieves 1.76 TOP/s, 1.06 TOP/s on Xilinx VU9P and ZCU102 platforms. Calabash outperforms Intel E5-2698 v4 with 2.5X - 6.7X speedup, and 27.0X - 77.3X energy-efficiency improvement. Compared with NVIDIA 2080Ti GPU and V100 GPU, Calabash shows 4.9X - 5.1X and 5.4X - 5.6X energy-efficiency improvement.

II. Background

Attention mechanism can be described as mapping a query vector and a set of key-value vector pairs to an output vector. Multiple vectors of queries, keys and values form the query matrix, key matrix, and value matrix, respectively. We can divide the computation of attention mechanism into two stages, as shown in Figure 1. In the first stage, the query (Q), key (K), value (V) matrices are obtained by multiplying the input matrix and corresponding weight matrices (W_Q, W_K, W_V). In the second stage, the score matrix is calculated by multiplying the query matrix and key matrix. We then divide the score matrix by the square root of dimension of the key vectors, and pass it to a row-wise softmax function where the exponential operation e^x is required. Finally, the output matrix of attention mechanism is generated by multiplying the normalized score matrix and value matrix. In summary, the attention mechanism can be formulated as follows.

$$S(i,j) = \sum_k Q(i,k)K(j,k) \tag{1}$$

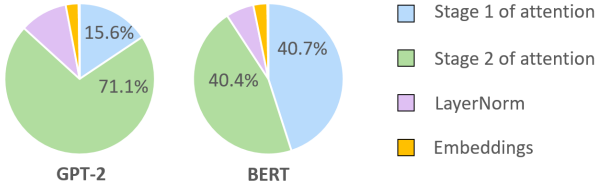

Fig. 2. Latency profiling on 2080Ti GPU.

$$P(i,j) = \frac{e^{(S(i,j)/\sqrt{dim_k})}}{\sum_j e^{(S(i,j)/\sqrt{dim_k})}} \tag{2}$$

$$Z(i,j) = \sum_k P(i,k)V(k,j) \tag{3}$$

According to Equation 1 and 3, each output element depends on one row of query matrix, the entire key matrix and one column of value matrix. Besides, according to Equation 2, to calculate $P(i,j)$, the entire row i of score matrix is needed. The complex data dependency of attention mechanism makes it difficult for FPGA acceleration. Moreover, the attention mechanism can involve multiple heads, namely multi-head attention. Each head helps the model to project the input sequence into a certain representation subspace for better interpretation. For multi-head attention, each head corresponds to one weight matrix and we can batch these matrices into a larger matrix, as shown in Figure 1.

We characterize the execution of attention mechanism by performing a detailed breakdown analysis on 2080Ti GPU. We use two state-of-the-art benchmarks, BERT [4] and GPT-2 [22]. We observe that execution time is mostly spent in the two stages of attention mechanism. For GPT-2 benchmark, attention accounts for 86.7% computation and it is 81.1% for BERT benchmark, as shown in Figure 2. The rest of the time is spent on the layer normalization and concatenation. Thus, hardware acceleration of attention mechanism is the key to overall performance.

III. Architecture Design

A. Overview

Figure 4 presents the overall architecture of Calabash. We design a systolic array chain as the computation engine. For the first stage of computation in Figure 1, we calculate the query, key, and value matrices individually using the two systolic arrays. The output of the two systolic arrays is directly stored to the off-chip memory. For each GEMM operation, we partition the input matrix into two sub-matrices equally and assign them to the two systolic arrays. For the second stage of computation, the two systolic arrays work together in a pipeline manner, which takes the query, key, value matrices as inputs, and calculates the output matrix. The softmax function is performed between two systolic arrays, which receives the output of the first systolic array and sends the results to the second systolic array.

To minimize the data transfer, Calabash optimizes the data layout from two aspects. First, the two systolic arrays apply the same output stationary (OS) dataflow for GEMM implementation [2]. Using this dataflow, the partial sums of

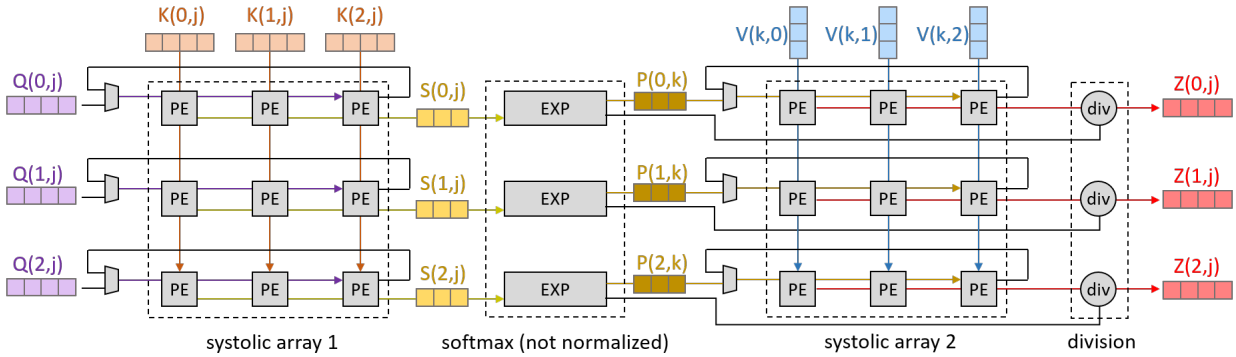

Fig. 3. Systolic array chain for accelerating attention mechanism.

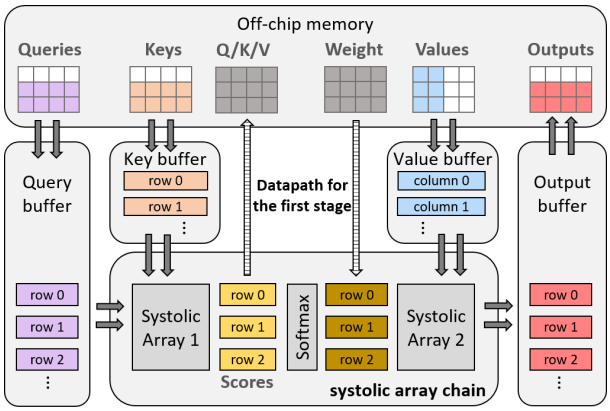

Fig. 4. Architecture overview.

one output element is always generated by the same PE. Therefore, each output element stays stationary in a PE, and each PE sends the input data of two input matrices horizontally and vertically to the neighboring PEs at each cycle. We choose OS dataflow because both input and output data are accessed in row-major sequence, which ensures data moves smoothly between two systolic arrays. Second, we propose a partition scheme that partitions the query matrix and key matrix in the row dimension, and partitions value matrix in the column dimension, as shown in Figure 4. By doing this, the intermediate score matrix generated by the first systolic array can be directly used by the second systolic array without data rearrangement.

B. Systolic Array Chain

The key component of Calabash is a systolic array chain. Figure 3 depicts the detailed dataflow of the second stage. Data passes through the first systolic array, then the EXP unit, and the second systolic array. The first systolic array receives the query matrix horizontally with each row of query matrix mapped to one PE row. The row of key matrix is vertically sent to the PEs of the systolic array to conduct the operation ($S = Q \times K^T$). In this dataflow, each single score element is kept stationary in a PE until the calculation is finished. The entire PE array performs an outer-product where each element in one column of matrix Q is multiplied with all the elements in one row of matrix K^T. The score matrix S, which is generated in row-major sequence by the first systolic array is passed to the EXP unit to calculate the softmax matrix using Equation 2. The EXP module calculates the non-normalized value $e^{(S(i,j)/\sqrt{dim_k})}$ and pass it to the next systolic array. The normalization is delayed to the end of computation when the entire row of the score matrix is generated.

The second systolic array uses the same dataflow as the first one, which means each row of the score matrix can be directly accessed without rearrangement. The second systolic array multiplies the non-normalized matrix P with the value matrix V. Each column of matrix V is mapped to one column of the PE array and each row of matrix P is mapped to one row of the PE array. Finally, the result matrix is divided with the accumulated value from EXP unit for normalization.

C. PE Details

For the first systolic array, the PE in the position (x,y) multiplies the query Q(x,j) with key K(y,j), and accumulates the multiplication result to the score S(x,y). In the next cycle, the query Q(x,j) is sent forward to the adjacent PE(x,y+1) while the key K(y,j) is sent to PE(x+1,y). The output is kept stationary in a PE. When one PE finished its computation, it will move its score and the gathered score from the left to the right-neighboring PE. In the end, the right-most PE will collect the entire row of the score matrix. The second systolic array follows the same dataflow but operating on different matrices.

D. EXP Module Design

We adopt similar design in [27] to conduct the exponential operator. The x^{th} EXP module outputs one $e^{S(x,j)}$ in each cycle. $e^{S(x,j)}$ is firstly converted into $2^{S(x,j)/\ln 2}$. $S(x,j)/\ln 2$ is then split into the integer part u and the fractional part v. In this manner, the result equals to $2^{u+v} = 2^v << u$. Different from the approach in [27], we use a look-up table to approximate 2^v. Considering that the first-order approximation of 2^v is $(1 + v \ln 2)$, we use $(d + v)$ for approximation as it requires less logic resources. d is determined by a look-up table, which minimizes the error for a specific range of v.

E. Workload Partition and Scheduling

The matrices in Figure 1 are usually with large size in modern transformer models. Considering the limited memory resource on FPGAs, we have to partition the matrices into tiles, as shown in Figure 5 (a). Clearly, the query tile contains T_Q rows of query matrix, the key tile contains T_K rows, and

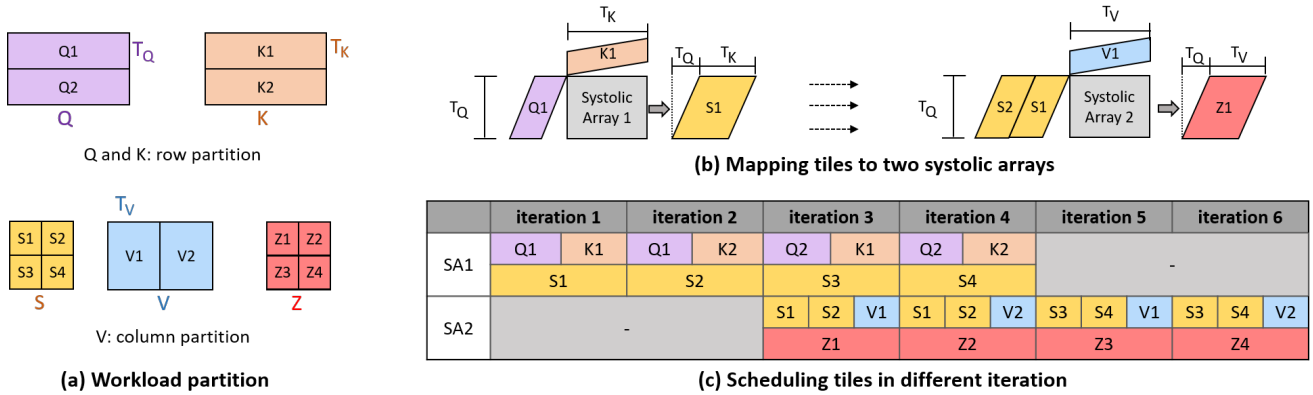

Fig. 5. The partition of the query, key, value matrix to fit the systolic array size and the scheduling for the partitioned tiles.

the value tile contains T_V columns of value matrix. Such partition strategy determines the systolic array size, as shown in Figure 5 (b). The size of the first systolic array is $T_Q \times T_K$, which generates a score tile with the same size. For the second systolic array, the size is $T_Q \times T_V$.

As the matrices are partitioned into multiple tiles, there are different access sequence of these tiles. In Calabash, we schedule the input tiles to ensure the score matrix is calculated in a row-by-row manner. In other words, we can cache the query tile on chip, and multiply it with different key tiles. In this manner, T_Q rows of the score matrix can be stored using on-chip memory and used by the second systolic array without off-chip memory transfer. Similarly, this score tile is reused by multiple value tiles to generate a $T_Q \times T_V$ result tile. Figure 5 (c) shows a tile scheduling example. We partition the input matrices (Q, K, V) into two tiles. The Q1 tile is kept on chip and multiplied with K1 and K2 tiles in two iterations. The generated S1 and S2 tiles compose T_Q rows of score matrix, which are sent to the second systolic array. The S1 and S2 tiles are reused twice with value tiles V1 and V2 to calculate the result tile Z1 and Z2.

IV. Design Space Exploration

A. FPGA Resource Utilization

Our implementation of attention involves a few design parameters (T_Q, T_K, T_V), which are the partition factors for query, key, value matrix. These design parameters determine the on-chip memory requirement and the systolic array size, which further affect the performance and latency. We first model the computation and memory resources. The computation resource is modeled as the number of multipliers in the systolic array chain and the EXP module. We use α_{EXP} to denote the number of multipliers for one EXP module. For the memory resource, we calculate the number of required memory banks where we assume that one bank only has one port for data transfer.

$$\begin{aligned} Res_{multiplier} &= T_Q \times T_K + T_Q \times T_V + T_Q \times \alpha_{EXP} \\ Res_{bank} &= 6T_Q + 2T_K + 2T_V \end{aligned} \tag{4}$$

As the query tile and score tile are cached using the on-chip memory and iteratively reused, the bandwidth requirement mainly depends on the access of key tiles and value tiles. Assuming that the partition factor is much smaller than the overall matrix size, and the data is stored in 16 bits. The bandwidth requirement can be estimated as follows,

$$BW = (T_K + T_V) \times 16bits \times freq \tag{5}$$

where $freq$ is the FPGA frequency.

B. Latency Modeling and Workload Balancing

The latency of the first stage can be easily derived as each systolic array works individually. The second stage is more complex as the two systolic arrays are pipelined. In the following, we model the latency of the second stage. For simplicity, the GEMMc operation is notated as,

$$Z[L_Q][L_V] = Q[L_Q][L_N] \times K^T[L_N][L_K] \times V[L_K][L_V] \tag{6}$$

The latency for calculating one output tile in each systolic array can be written as

$$\begin{aligned} LAT_{sa1} &= L_N + 2T_K + T_Q, \\ LAT_{sa2} &= L_K + 2T_V + T_Q \end{aligned}$$

where L_N is the latency to perform dot-product operation for two vectors with length L_N and $2T_K + T_Q$ is the skewing factor for systolic array, as shown in Figure 5 (b). The latency for computing T_Q rows of the score matrix and the result matrix is

$$\begin{aligned} LAT_s &= \lceil \frac{L_K}{T_K} \rceil \times LAT_{sa1}, \\ LAT_z &= \lceil \frac{L_V}{T_V} \rceil \times LAT_{sa2} \end{aligned}$$

The total latency is

$$LAT_{total} = \lceil \frac{L_Q}{T_Q} \rceil \times max(LAT_s, \ \ LAT_z) \tag{7}$$

In general, the input matrix size is much more larger than the design parameters,

$$(L_Q, L_N, L_K, L_V) >> (T_Q, T_K, T_V)$$

When $LAT_s = LAT_z$, the workloads of the two systolic arrays are balanced. Our goal is to minimize the total latency when the workload balance constraint is satisfied,

$$LAT_{total} \approx \frac{L_Q}{T_Q} \times max(\frac{L_K \times L_N}{T_K}, \frac{L_V \times L_K}{T_V}) \tag{8}$$

TABLE I
PERFORMANCE COMPARISON WITH 2080TI GPU AND XEON E5-2698 V4 CPU ON THREE BENCHMARKS.

Applications	BERT[4]					GPT-2[22]					BART[11]				
Platform	Intel E5 -2698 v4	Nvidia 2080Ti	Nvidia V100	Xilinx VU9P	Xilinx ZCU102	Intel E5 -2698 v4	Nvidia 2080Ti	Nvidia V100	Xilinx VU9P	Xilinx ZCU102	Intel E5 -2698 v4	Nvidia 2080Ti	Nvidia V100	Xilinx VU9P	Xilinx ZCU102
Technology	14nm	12nm	12nm	16nm	16nm	14nm	12nm	12nm	16nm	16nm	14nm	12nm	12nm	16nm	16nm
Frequency (MHz)	2200	1950	1530	243	300	2200	1950	1530	243	300	2200	1950	1530	243	300
Performance (TOP/s)	0.28	8.54	8.19	1.76	1.06	0.32	9.32	10.2	1.76	1.06	0.73	7.48	7.19	1.61	0.96
Power (W)	135	260	275	12.9	6.6	135	260	275	12.9	6.6	135	260	275	12.9	6.6
Energy-eff (GOP/s/W)	2.07	32.8	29.8	136.4	160.7	2.37	35.8	36.4	136.4	160.7	5.41	28.8	26.1	124.8	146.2

V. EXPERIMENT

A. Experiments Setup

We evaluate Calabash using three well-known transformer-based models in the Natural Language Processing (NLP) domain, including BERT [4], GPT-2 [22], BART [11]. The data is quantified into 16-bit integer arithmetic.

We target Xilinx VU9P and ZCU102. Calabash is written in the Chisel[1]. Xilinx Vivado is used to obtain the FPGA bitstream. We use 2080Ti GPU platform operated at 1.95 GHz, and V100-DGXS GPU platform operated at 1.53 GHz. We measure the performance of GPUs using PyTorch with cuBLAS 11.2. For CPU, we use Intel Xeon E5-2698 v4 platform operated at 2.2 GHz and PyTorch MKL 2020.0.2.

B. Performance Evaluation

For Calabash on Xilinx V9PU and ZCU102, we set the design parameters $\{T_Q = 64,\ T_K = T_V = 32\}$, $\{T_Q = 64,\ T_K = T_V = 16\}$, respectively. Table I gives the comparison results at the batch size of 1.

The theoretical performance of Calabash is 1.99 TOP/s ($4096 \times 0.243 \times 2$) and 1.23 TOP/s ($2048 \times 0.3 \times 2$) on VU9P and ZCU102. As shown in Table I, our design achieves 1.76 TOP/s and 1.06 TOP/s on VU9P and ZCU102, which results in 88.4% and 87.0% PE efficiency. The gap mainly comes from the initial and the last iterations in the pipeline where only one systolic array is activated, as shown in Figure 5 (c). In the three benchmarks, the most common size is 512×768 for query, key, value matrices, which requires $\frac{512}{16} = 32$ iterations to get the first 64 rows of score matrix. And the iteration number for each systolic array is $\frac{512}{16} \times \frac{768}{64} = 384$. Therefore, the theoretical efficiency of the systolic array chain is $\frac{384}{384+32} = 92.3\%$, which is similar to the real efficiency.

Table II lists the predicted and real resource utilization and performance. It shows that our resource model can accurately estimate the BRAM and DSP utilization. As the maximum of matrix width is 768, each row is stored in one BRAM. According to Equation 4, the predicted BRAM utilization is 512 (384+64+64) and 448 (384+32+32) for VU9P and ZCU102, respectively. It is reasonable that the real BRAM utilization is slightly higher than the predicted. The extra BRAMs are used for FIFO logic in the off-chip data transfer. The predicted DSP number is 4224 (2048+2048+128) and 2176 (1024+1024+128) for VU9P and ZCU102, which is very close to the real DSP utilization. We also give the predicted performance calculated by dividing the total operations with the estimated latency. The results show that the error rate of performance prediction is within 7%.

TABLE II
RESOURCE UTILIZATION BREAKDOWN

		BRAM18Kb	DSP	FF	LUT
VU9P 4096×PEs 64×EXPs	Systolic Array	0	4096	210691	204380
	EXP	0	128	1024	2158
	Mem	512	0	255	4534
	Others	128	3	1134	37
	Total	640	4227	213004	211109
	Available	4320	6840	2364480	1182240
	Utilization	14.8%	61.8%	9.0%	17.8%
	Predicted Utilization	11.9%	61.7%	-	-
	Predicted Performance	1.84 TOP/s (real: 1.76 TOP/s, 5% error)			
ZCU102 2048×PEs 64×EXPs	Systolic Array	0	2048	106273	71116
	EXP	0	128	1024	1903
	Mem	448	0	446	3777
	Others	64	3	816	31
	Total	512	2179	108559	76827
	Available	1824	2520	548160	274080
	Utilization	28.1%	86.4%	19.8%	28.0%
	Predicted Utilization	24.6%	86.3%	-	-
	Predicted Performance	1.13 TOP/s (real: 1.06 TOP/s, 7% error)			

VI. RELATED WORK

A few designs have been proposed for attention acceleration [7], [12], [20], [23], [6], [8], [15]. These accelerators are limited in either efficiency or generality, and most of them target ASICs. Sanger[15] proposed another co-design framework for sparse attention. However its RePE design may cause severe place and route problem on FPGAs. TRAC[20] proposed a compiler, a library of operators and modules for implementing transformer accelerators on FPGAs, which is vertical to this work.

VII. CONCLUSION

We propose Calabash for attention-based applications acceleration. We link two heterogeneous systolic arrays to reduce the off-chip memory bandwidth requirement, and propose a scheduling scheme for the partitioned tiles to maintain the data layout and balance the workload. Experiments demonstrate that Calabash achieves 1.76 TOP/s, 1.06 TOP/s on Xilinx VU9P and ZCU102 platforms, which shows an average 50.1X and 3.94X energy-efficiency improvement compared with CPU and GPU, respectively.

ACKNOWLEDGEMENTS

This work is supported in part by the National Natural Science Foundation of China (NSFC) under grant No T2293700 and T2293701

REFERENCES

[1] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: constructing hardware in a scala embedded language," in Proceedings of DAC, 2012.

[2] Y.-K. Chen and S.-Y. Kung, "A systolic design methodology with application to full-search block-matching architectures," VLSI, 1998.

[3] J. Cong and J. Wang, "PolySA: Polyhedral-Based Systolic Array Auto-Compilation," in Proceedings of ICCAD, 2018.

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in Proceedings of NAACL-HLT, 2019.

[5] F. Fahim, B. Hawks, C. Herwig, J. Hirschauer, S. Jindariani, N. Tran, L. P. Carloni, G. D. Guglielmo, P. C. Harris, J. D. Krupa, D. S. Rankin, M. B. Valentin, J. D. Hester, Y. Luo, J. Mamish, S. Orgrenci-Memik, T. Aarrestad, H. Javed, V. Loncar, M. Pierini, A. A. Pol, S. Summers, J. M. Duarte, S. Hauck, S. Hsu, J. Ngadiuba, M. Liu, D. Hoang, E. Kreinar, and Z. Wu, "hls4ml: An open-source codesign workflow to empower scientific low-power machine learning devices," Arxiv preprint 2103.05579, 2021. [Online]. Available: https://arxiv.org/abs/2103.05579

[6] H. Fan, T. Chau, S. I. Venieris, R. Lee, A. Kouris, W. Luk, N. D. Lane, and M. S. Abdelfattah, "Adaptable butterfly accelerator for attention-based nns via hardware and algorithm co-design," in Proceedings of MICRO, 2022.

[7] T. J. Ham, S. J. Jung, S. Kim, Y. H. Oh, Y. Park, Y. Song, J.-H. Park, S. Lee, K. Park, J. W. Lee et al., "A3: Accelerating attention mechanisms in neural networks with approximation," in Proceedings of HPCA, 2020.

[8] T. J. Ham, Y. Lee, S. H. Seo, S. Kim, H. Choi, S. J. Jung, and J. W. Lee, "ELSA: hardware-software co-design for efficient, lightweight self-attention mechanism in neural networks," in Proceedings of ISCA, 2021.

[9] L. Jia, Z. Luo, L. Lu, and Y. Liang, "Tensorlib: A spatial accelerator generation framework for tensor algebra," in Proceedings of DAC, 2021.

[10] L. Jia, Y. Wang, J. Leng, and Y. Liang, "EMS: efficient memory subsystem synthesis for spatial accelerators," in Proceedings of DAC, 2022.

[11] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," ACL, 2020.

[12] B. Li, S. Pandey, H. Fang, Y. Lyv, J. Li, J. Chen, M. Xie, L. Wan, H. Liu, and C. Ding, "FTRANS: energy-efficient acceleration of transformers using FPGA," in Proceedings of ISLPED, 2020.

[13] Y. Liang, L. Lu, Q. Xiao, and S. Yan, "Evaluating fast algorithms for convolutional neural networks on fpgas," TCAD, 2020.

[14] Y. Liang, Q. Xiao, L. Lu, and J. Xie, "Fcnnlib: A flexible convolution algorithm library for deep learning on fpgas," TCAD, 2022.

[15] L. Lu, Y. Jin, H. Bi, Z. Luo, P. Li, T. Wang, and Y. Liang, "Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture," in Procceedings of MICRO, 2021.

[16] L. Lu and Y. Liang, "Spwa: an efficient sparse winograd convolutional neural networks accelerator on fpgas," in DAC, 2018.

[17] L. Lu, Y. Liang, R. Huang, W. Lin, X. Cui, and J. Zhang, "Speedy: An accelerator for sparse convolutional neural networks on fpgas," in FPGA, 2019.

[18] L. Lu, J. Xie, R. Huang, J. Zhang, W. Lin, and Y. Liang, "An efficient hardware accelerator for sparse convolutional neural networks on fpgas," in FCCM, 2019.

[19] S. Na, S. Lee, J. Kim, and G. Kim, "A read-write memory network for movie story understanding," in Proceedings of ICCV, 2017.

[20] P. Plagwitz, F. Hannig, and J. Teich, "TRAC: compilation-based design of transformer accelerators for fpgas," in Proceedings of FPL, 2022.

[21] A. Putnam, et al., "A reconfigurable fabric for accelerating large-scale datacenter services," in Proceedings of ISCA, 2014.

[22] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," OpenAI blog, 2019.

[23] G. Shen, J. Zhao, Q. Chen, J. Leng, C. Li, and M. Guo, "SALO: an efficient spatial accelerator enabling hybrid sparse attention mechanisms for long sequences," in Proceedings of DAC, 2022.

[24] Y. Umuroglu, Y. Akhauri, N. J. Fraser, and M. Blott, "Logicnets: Co-designed neural networks and circuits for extreme-throughput applications," in Proceedings of FPL, 2020.

[25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," NeurIPS, 2017.

[26] E. Wang, J. J. Davis, P. Y. K. Cheung, and G. A. Constantinides, "Lutnet: Rethinking inference in FPGA soft logic," in Proceedings of FCCM, 2019, pp. 26–34.

[27] M. Wang, S. Lu, D. Zhu, J. Lin, and Z. Wang, "A high-speed and low-complexity architecture for softmax function in deep learning," in Proceedings of APCCAS, 2018.

[28] Q. Xiao and Y. Liang, "Towards agile DNN accelerator design using incremental synthesis on fpgas," in FPGA, 2022.

[29] Q. Xiao, Y. Liang, L. Lu, S. Yan, and Y. Tai, "Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on fpgas," in DAC, 2017.