Ph.D. Project: Achieving Low-Latency Acceleration on Multi-FPGA for GPT Application

ZhenDong Zheng, Teng Wang, Chao Wang

High Energy-Efficient Intelligent Computing Lab, Suzhou Institute for Advanced Research, University of Science and Technology of China

ABSTRACT

Large Language Models (LLMs) have been widely deployed in data centers to provide various services, among which Generative Pre-trained Transformer (GPT) is the most representative. GPT has heavy memory and computing overhead, and its inference process has two phases with distinct computing characteristics: Initialization and Generation. Utilizing existing GPUs and FPGA accelerators to construct a platform for deploying GPT in data centers faces the challenges of needing more effective synchronization strategies or high computational intensity structure. This paper proposes a latency-oriented recurrent architecture for GPT on multi-FPGA with communication optimization. We devise an efficient communication scheme that overlaps part of the computation and communication delay to improve the latency and scalability of our platform. Then, we deploy recurrent structures on each FPGA to accelerate the different phases of GPT. A preliminary experiment shows that our method can reduce the synchronization overhead and increase the computing intensity, resulting in an average 11.8× speedup over NVIDIA V100 GPU and 3.0× speedup over existing multi-FPGA accelerator appliance on the GPT-2 model.

PROBLEM AND MOTIVATION

In recent years, Transformer-based Large Language Models (LLMs) have achieved a significant influence on both academic research and daily life[1]. Among them, the GPT series models[2] developed by OpenAI have become a research hotspot in Artificial Intelligence. GPT can achieve excellent results in complex natural language processing (NLP) applications such as scientific writing, machine translation, and text classification. Part of these benefits come from the architectural innovation brought by self-attention, and on the other hand, comes from a large number of parameters and training data, which also bring massive computing and memory overhead to the platform.

Data centers mainly deploy GPT models on GPU clusters to provide NLP services. However, the following two issues can lead to low GPU utilization. First, the Generation phase of GPT requires frequent sequential computing of a single token. Second, to improve the Quality of Service (QoS) and meet the real-time requirements, the acceleration cluster will not process small batches of user input by waiting for more data, so the batch size is usually set to one[3]. The architecture of GPU is designed to process batches of data using data parallelism. So, in this case, the GPU will face insufficient computing intensity. We can use FPGA to customize the acceleration framework to alleviate this problem.

FPGA has the advantages of a short development cycle, low latency, and high energy efficiency and is widely used to deploy various types of neural network accelerators[4]. In the early days, researchers focused only on accelerating a specific computing module in Transformer[5]. This creates a large amount of data that needs to be transferred frequently between the host and the accelerator. As a result, more and more work has begun to develop end-to-end accelerators that support the entire Transformer operation without missing any intermediate modules[6].

However, most FPGA accelerators deploy their Transformer accelerators only on a single card. Using these accelerators to accelerate the entire GPT model will face severe storage limitations. Taking GPT-3 as an example, it has 175 billion parameters. If we plan to mitigate this issue through replication, we need model splitting and synchronization algorithms. Moreover, their architectures are designed for data-parallel scenarios; when operating single batch input, they face problems similar to GPU.

This paper proposes a low-latency end-to-end GPT multi-FPGA acceleration platform. In response to the storage limitation issues faced by single-card deployment, we use the model parallel algorithms, the same as the existing work[7], to split the GPT model and deploy it to multiple FPGA boards. Secondly, to reduce transmission delays, we design efficient synchronization algorithms and communication strategies to overlap computation and communication as much as possible. Finally, to address the low utilization problem faced by GPUs and traditional FPGA accelerators in data centers, we design an accelerator with an input batch size of one by using recurrent structures and deploy it on each FPGA board of the platform.

BACKGROUND AND RELATED WORK

Among the existing work on the GPT acceleration platform based on multi-FPGA, DFX has proposed a relatively complete framework, including model splitting methods, synchronization strategies between boards, and computing core designs.

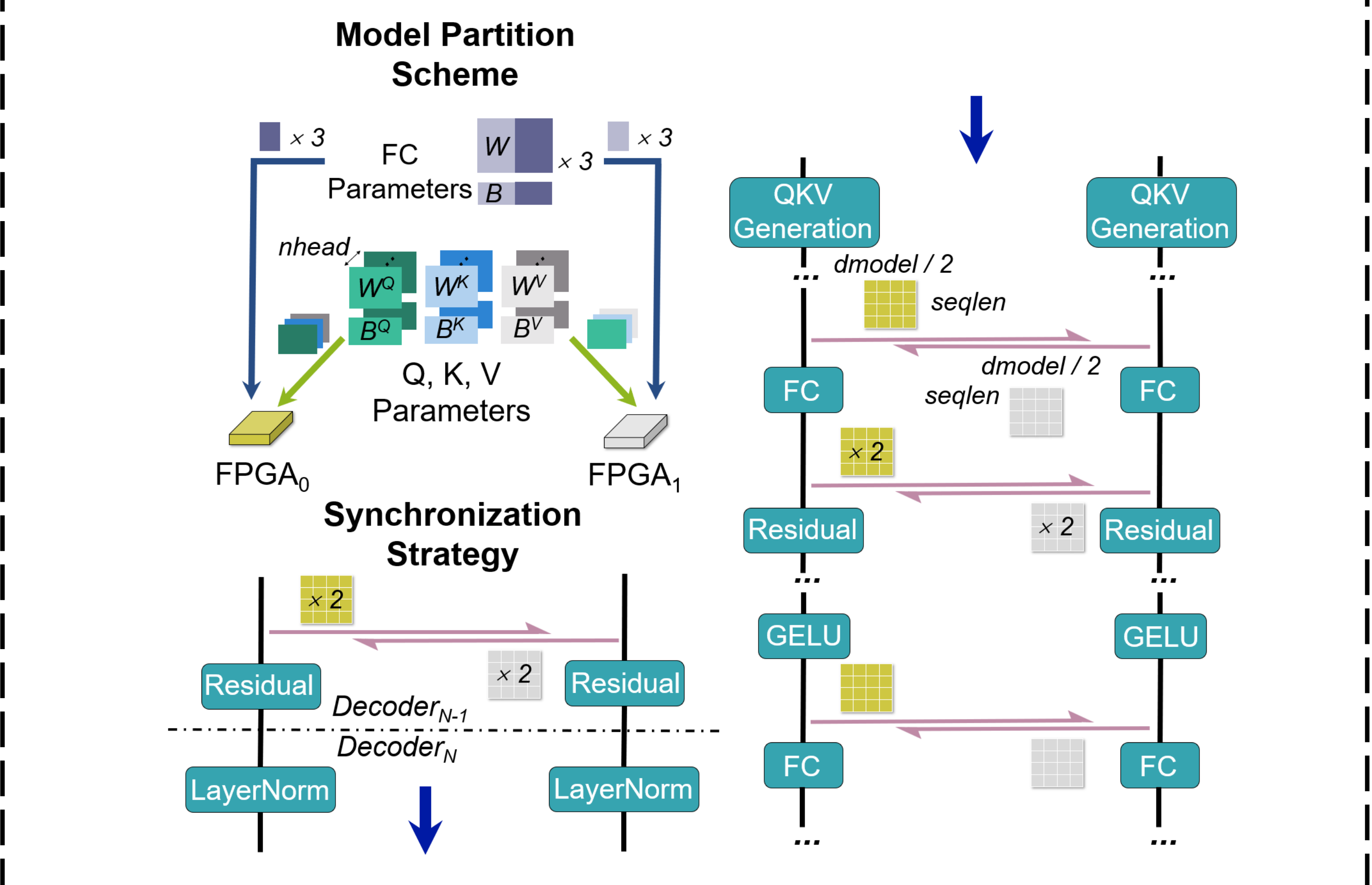

Fig. 1: DFX's Model Partition and Synchronization Strategy on 2 FPGAs.

As shown in Fig. 1. The synchronization strategies DFX uses lead to two problems. First, the synchronization timing is far from the modules that use these data. So, the calculation load of the intermediate layers will be increased. Second, the transmission method will bring serious communication latency growth when the cluster expands. DFX uses a multiply-add tree and a fixed tiling scheme optimized for vector-matrix multiplication to accelerate the Initialization and Generation Phase of GPT models, which results in a poor weight data reuse rate when facing inter-matrix multiplication. Different tiling schemes and a broadcast array can accelerate these multiplications.

APPROACH AND UNIQUENESS

We adjust the synchronization strategy to reduce the model inference caused by communication. Then, we optimize the tiling scheme and computing core design to improve the computing intensity of our accelerator.

A. Synchronization Strategy Design

When extending a single-card FPGA accelerator to multi-cards using model parallelism, we need to consider how to partition the GPT model. We divide the parameters of all QKV matrices and FC layers into different boards according to the head and columns. After reshaping the results of the Attention and FC layers, each board will only have data corres--ponding to specific columns of the entire model. Based on the board's local data, subsequent LayerNorm, QKV Generation, or FC layers cannot produce complete output.

To solve this issue, we set that when the calculation proceeds to LayerNorm, QKV Generation, and FC layers, all boards must communicate to obtain global data, as shown in Fig. 2. These synchronization timings will not increase the calculation load of other layers. The synchronization strategies used also need to minimize system congestion caused by communication. So, we overlap the operations of computation and communication. To do this, we adjust the order of operations of specific calculations and transmit the data sooner after it is generated.

When computing the mean and variance in LayerNorm of each row, we first calculate the partial sum according to the local data of each board. After that, we transfer these partial sums between boards immediately. As for the remaining operations, we can use the board's local data to complete them. This strategy compresses the data volume required for LayerNorm synchronization from the whole input into several partial sums.

When calculating FC layers, We first carry out all the operations between the existing local input data and the local weight and then complete the remaining operations after obtaining the input data of other boards. We can advance the broadcast of local input data and the reception of input data from other boards to use the longer computing delay to overlap the transmissions. The difference among these FC layers lies in the timing of the start of the broadcast and reception.

Before the operation of the first FC layer, each board needs to multiply *head* matrices of size *seqlen*×*seqlen* and *seqlen*×*dhead* in sequence, and we call this step SmulV. After calculating the result corresponding to a head in SmulV, each board can mark, pack, and broadcast the results to other boards. From then on, each FPGA will also be ready to receive data from other boards.

The operations in front of the remaining two FC layers are Scaling in LayerNorm and GELU. Each data calculation in these two processes runs independently and has similar operation steps. We can arrange the start timing of broadcast and receiving behaviors after completing the calculation of a row of data. As for the QKV Generation layer's synchronization strategy is almost identical to that of the FC layers.

Next, we can package *d* rows of data into a group. The rows in the same group are processed in parallel, and the pipelined algorithm is used between groups. Considering that the bit width of the stream port in the communication IP is usually much larger than the data width of the GPT model, we can reduce bandwidth waste by packing *d* data and transmitting them together. The more important point is that when resources are limited, packing and inter-row parallelism will better ensure that our computing operations can completely cover up communication delays, and this operation will not increase the total delay.

B. Computing Unit Design

The main innovation of the Computing Unit is to use more reasonable tiling schemes and a more intensive calculation method to accelerate matrix multiplication. By multiplexing the Matrix Processing Unit (MPU), we can complete the intra-matrix and vector-matrix multiplication. As for Vector Function Unit (VFU), it is responsible for vector addition, division, and other vector operations.

Inter-Matrix Multiplication: After tiling the large matrix into small matrices of *d*×*d*, we use the small matrix as the basic unit and traverse the input and weight in the order of first column, then row and first row, then column, respectively. After that, we use broadca--sting to further accelerate the matrix multiplication after tilling. We arrange *d*×*d* Processing Elements (PEs) according to the array shown in Fig. 3. Each PE contains a multiplier, an adder, and an accumulation register. When we multiply two *d*×*d* matrices, we need to broadcast the *i-th* tow of input to the *i-th* row of PEs and broadcast the *j-th* column of weight to the *j-th* column of PEs. Because all PEs run in parallel and intra-PE is accelerated through pipelines, the matrix multiplication theoretically requires at least *d* cycles.

Vector-Matrix Multiplication: We then multiplex the MPU to accelerate vector-matrix multiplication. In one processing cycle, we complete the multiplication between a *d* dimensional vector and *l* different *d*×*d* matrices. We broadcast the *d* dimensional vector to all PEs and send these *l* different *d*×*d* matrices to *l* rows of PE units in turn. For each row of PEs, send the *j-th* column data of the *d*×*d* matrix corresponding to this row to the *j-th* PE processing unit. After *d* times MAC operations, all vector-matrix multiplication in one processing cycle can be completed.

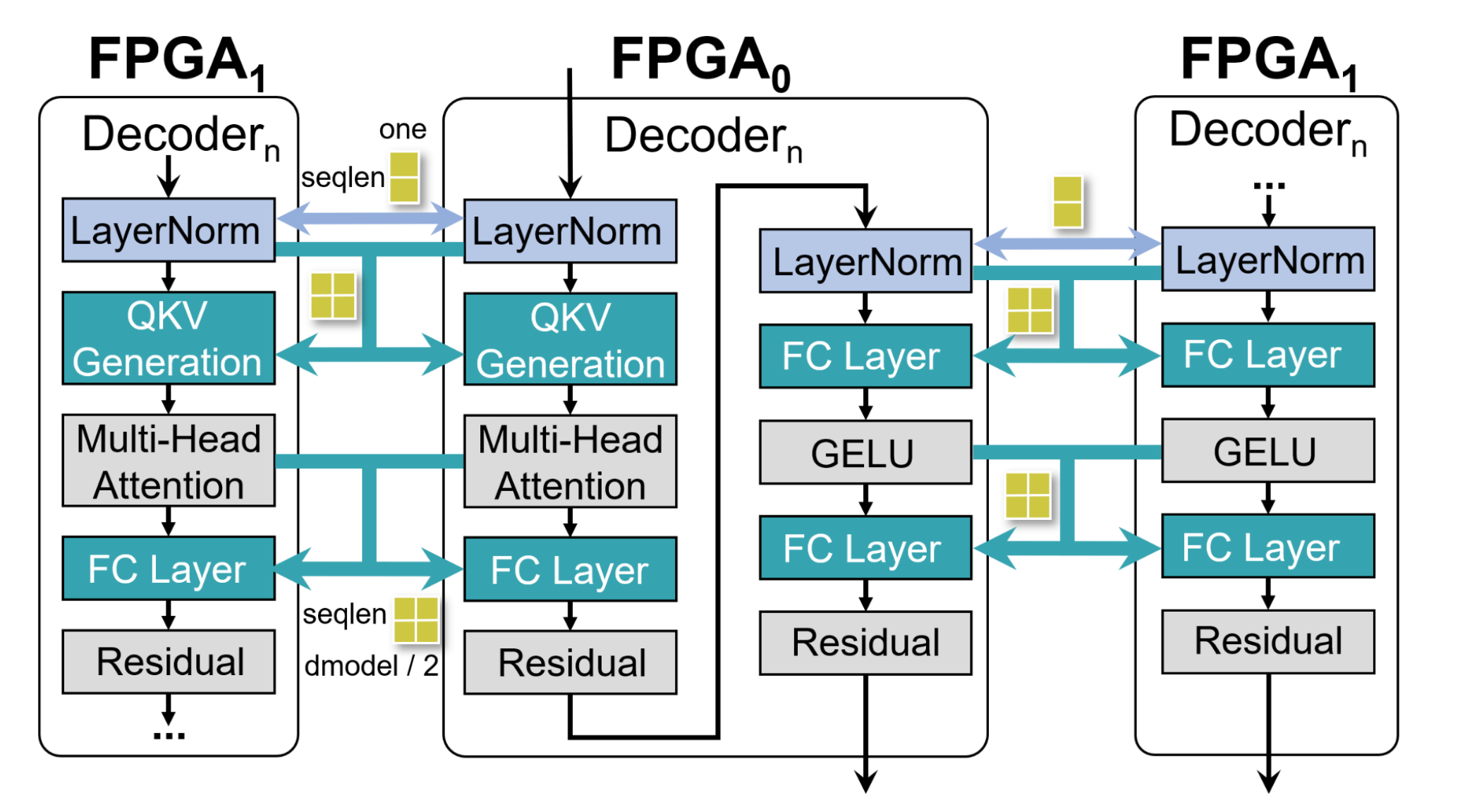

Fig. 2: Synchronization Strategy for Decoder on 2 FPGAs.

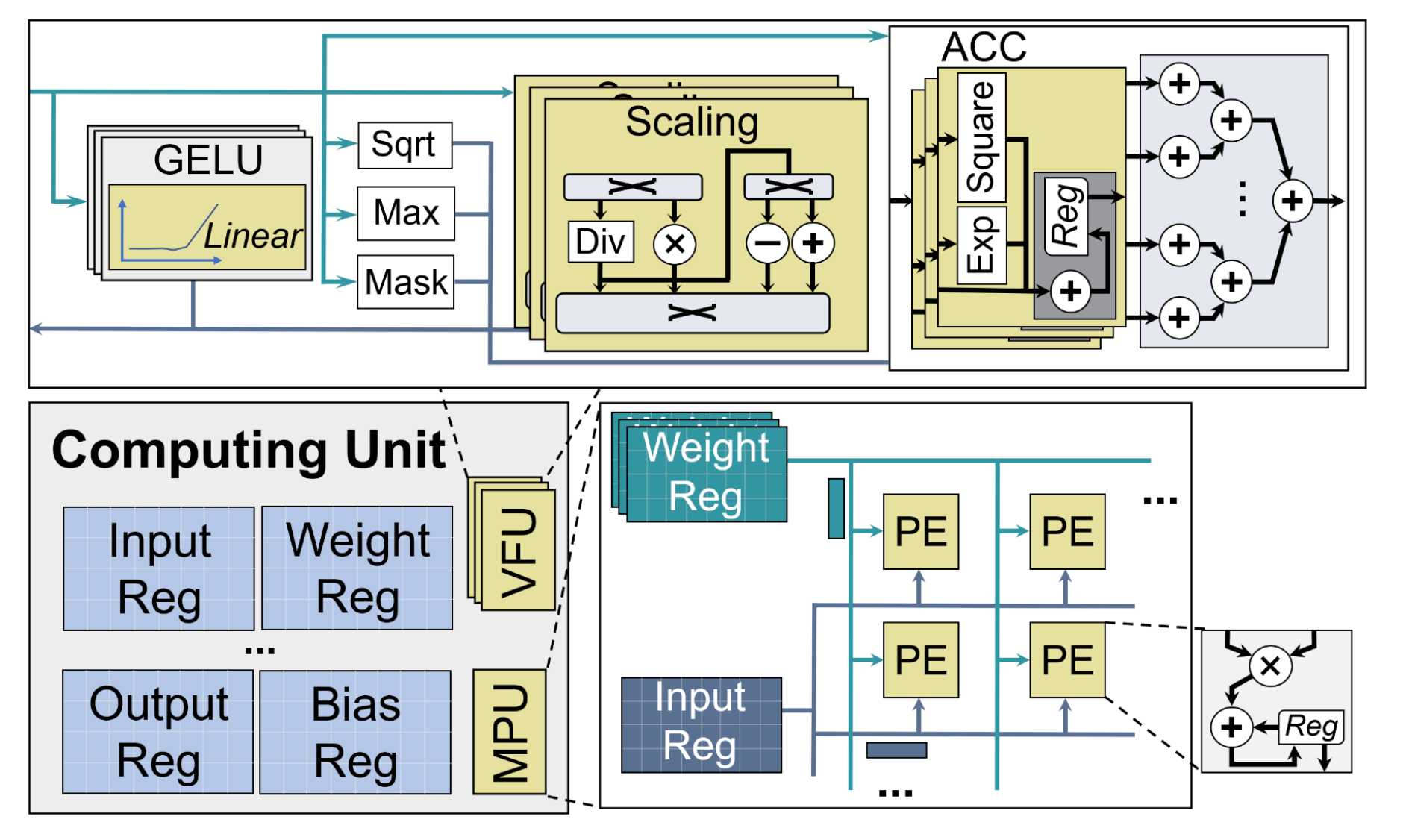

Fig. 3: The Overall Architecture of Computing Unit.

EXPECTED RESULTS AND CONTRIBUTIONS

We build an appliance prototype using an AUS ESC8000A-E11 rack server and two Alveo U280 data center accelerator cards for evaluation. These FPGA boards are connected by Aurora 64/66B IP and QSFP cable. We configure the DFX platform and V100 GPU in the same way as[7]. The experiment results in Table 1 show that the average acceleration ratio of our design compared to V100 GPU is *11.8*×, and the average ratio compared to DFX is *3.0*×. After evaluation, the synchronization strategy we introduced saves at least *10ms* of model inference latency in the Initialization Phase. Though the communication latency saved here is relatively small, we have covered almost all communication delays using computing. When the cluster expands, the effect of our proposed fine-grained algorithm will be more significant. In future work, we will conduct experiments on more boards, models and the size of input and output tokens.

Latency	[Input Size: Output Size]			
(ms)	[128:1]	[128:4]	[128:64]	[128:256]
V100	67.7	251.2	4150.8	17692.3
DFX	876.5	869.9	1304.5	2609.1
Ours	213.0	222.6	413.4	1024.0

Table 1. Inference Latency of V100 GPU, DFX and our design when Deploying 774M GPT-2 on two cards

REFERENCES

[1] Wei J, Tay Y, Bommasani R, et al. Emergent abilities of large language models[J]. arXiv preprint arXiv:2206.07682, 2022.

[2] Achiam J, Adler S, Agarwal S, et al. Gpt-4 technical report[J]. arXiv preprint arXiv:2303.08774, 2023.

[3] Zeng S, Liu J, Dai G, et al. FlightLLM: Efficient Large Language Model Inference with a Complete Mapping Flow on FPGA[J]. arXiv preprint arXiv:2401.03868, 2024.

[4] Wang T, Gong L, Wang C, et al. Via: A novel vision-transformer accelerator based on fpga[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2022, 41(11): 4088-4099.

[5] Qu Z, Liu L, Tu F, et al. Dota: detect and omit weak attentions for scalable transformer acceleration[C]//Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. 2022: 14-26.

[6] Qin Y, Wang Y, Deng D, et al. FACT: FFN-Attention Cooptimized Transformer Architecture with EagerCorrelation Prediction[C]//Proceedings of the 50th Annual International Symposium on Computer Architecture.2023: 1-14.

[7] Hong S, Moon S, Kim J, et al. DFX: A low-latency multi-FPGA appliance for accelerating transformer-based text generation[C]//2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE, 2022: 616-630.

CONTACT

ZhenDong Zheng
High Energy-Efficient Intelligent Computing Lab,
Suzhou Institute for Advanced Research,
University of Science and Technology of China.
Email: zzd1411@mail.ustc.edu.cn