PipeFL: Hardware/Software Co-Design of an FPGA Accelerator for Federated Learning

ZIXIAO WANG,1 BIYAO CHE1, LIANG GUO,2 YANG DU,1 YING CHEN,1 JIZHUANG ZHAO1, AND WEI HE3

1China Telecom Research Institute, Beijing 102209, China
2Institute of Cloud Computing and Big Data of CAICT, Beijing 100191, China
3China Telecom BestPay Company Ltd., Beijing 100031, China

Corresponding author: Zixiao Wang (wangzx15@chinatelecom.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2021ZD0113003, and in part by the China Telecom Internet Finance Research and Development Program under Grant 22HQBYYF0089.

ABSTRACT Federated learning has solved the problems of data silos and data fragmentation on the premise of satisfying privacy. However, cryptographic algorithms in federated learning brought significant increase in computational complexity, which limited the speed of model training. In this paper, we propose a hardware/software (HW/SW) co-designed field programmable gate array (FPGA) accelerator for federated learning. Firstly, we analyzed the time consumption of each stage in federated learning and the involved cryptographic algorithms, and found the performance bottleneck. Secondly, a HW/SW co-designed architecture is introduced, which can speed up encryption, decryption and ciphertext-space computation at the same time without reconfiguring FPGA circuit. In the HW part, we proposed a Hardware-aware Montgomery Algorithm (HWMA) which utilized data parallelism and pipeline, and designed an FPGA architecture to decouple data access and computation. In the SW part, an Operator Scheduling Engine (OSE) is designed, which can flexibly resolve the target algorithm into multiple HWMA calls, and complete other non-computation-intensive calculations. Finally, evaluations for both specific algorithms and practical applications are implemented. Experimental results show that when deployed on Intel Stratix 10 FPGA, our accelerator can increase the throughput of 2048-bit modular multiplication, modular exponentiation and Paillier algorithm to more than 3x of the CPU. When integrated into a industrial grade federated learning open source framework, the end-to-end training time of linear regression and logistic regression can be shortened by 2.28x and 3.30x respectively, which is more than 2x faster than the reported best results of FPGA accelerator.

INDEX TERMS Federated learning, field programmable gate arrays, homomorphic encryption, privacy preserving.

I. INTRODUCTION

In recent years, with the continuous update and iteration of algorithms and specialized hardware, the development of machine learning and deep learning is in full swing, but these algorithms usually have huge demands for the amount of training datasets. Traditional distributed machine learning usually handles all the training datasets that can be obtained to be managed and used by one data center, which may leak and threaten the privacy contained in the data. With the implementation of the European Union’s General Data Protection Regulations [1], the model-related services provider must meet the prerequisite for ensuring the privacy of the user. In this background, the federated learning technology [2] proposed by Google has gained more and more attention.

Yang et al. [3] extended the concept of federated learning into secure federated learning which is currently widely used, and classified it as horizontal federated learning, vertical federated learning and federated transfer learning.
Secure federated learning is mainly based on cryptography, focusing on privacy protection and data security problems in the process of model training. During the training stage, only information related to the encrypted model (such as weight parameters, gradients, etc.) can be exchanged in all parties, and the datasets are always preserved in training participants locally. Thus, encryption-calculation-decryption operations are the most common operations in secure federated learning framework. Compared with traditional machine learning methods, they have brought new challenges. To ensure that the encrypted parameters can be calculated, the encryption algorithm is required to meet the characteristics of the homomorphic encryption (HE), that is, when calculate in ciphertext-space and then decrypt, the result should be consistent with the corresponding calculation directly in the plaintext-space.

Partial homomorphic encryption algorithm (PHE), represented by Paillier [4] algorithm and RSA [5] algorithm, can well meet the above requirements. The former is additive homomorphic algorithm, which is often used to encrypt the gradient of the model, and the latter is multiplicative homomorphic algorithm, which is often used to align the training samples of each participant.

However, PHE algorithms usually bring a great amount of extra calculations and large pressure of memory access. Firstly, the computational overhead mainly comes from the modular multiplication (ModMult) and modular exponentiation (ModExp) operations in the encryption algorithm. Secondly, since the bit width of the encryption key usually exceeds 1024-bit, the encrypted data of any plaintext parameter will also become a number with a large bit width, which makes the data access during gradient transmission and calculation expand sharply. The above challenges cause the training speed of federated learning to be much slower than that of distributed machine learning. Some researchers [6] have pointed out that, the training time after encryption in federated learning can reach more than three times than that of traditional distributed training, which makes the deployment of federated learning in the production environment extremely challenging.

Heterogeneous computing accelerators have been proved to be very effective in the field of machine learning and deep learning [7], [8], [9], [10], [11], [12]. Among them, field-programmable gate arrays (FPGAs) [10], [11], [12] have gain more and more attention in recent years because of their reconfiguration capability, rich logic implementation resource, and rapid development cycle supported by high-level synthesis (HLS). Therefore, how to use heterogeneous hardware to accelerate federated learning has been widely discussed in both academia and industry.

The main contributions of this paper are as follows:

- A hardware/software (HW/SW) co-design method is proposed targeting CPU+FPGA-based heterogeneous platforms, which can accelerate all time-consuming algorithms in federated learning that use ModMult or ModExp as the basic operation without programing FPGA repeatedly.
- The accelerator proposed in this paper has taken the real application in federated learning into consideration, used three OpenCL kernels to decouple data access and computation, and explored both data parallelism and pipeline parallelism to reach state-of-the-art throughput.
- The proposed accelerator is implemented on Intel Stratix 10 GX2800 FPGA and its performance is verified. At a bit width of 2048-bit, the throughput of ModMult operation can reach 1121.2 kOP/s, and the throughput of ModExp operation is 743.1 OP/s, which is 5.3x and 3.0x of CPU performance respectively. When integrated into an industrial grade federated learning open source framework, the end-to-end linear regression and logistic regression training time can be reduced by 2.28x and 3.30x respectively compared with CPU. Our design shows more than 3 times improvement on throughput over the best FPGA-based federated learning accelerator reported in literature.

This paper is organized as follows: Section II analyzes the existing accelerator design, summarizes its common characteristics, and puts forward the direction that can be further optimized for the federated learning scenario; Section III conducts an in-depth analysis of the encryption algorithm in federated learning, and points out bottlenecks in the current algorithm as theoretical guidance; Based on the previous analysis, Section IV describes the proposed HW/SW co-design in detail; In section V, two experiments are designed to evaluate the acceleration effect of the designed accelerator in both specific algorithms and end-to-end federated learning applications. The last Section VI gives conclusions and puts forward the future research direction.

II. RELATED WORK

In view of the challenges brought by encryption algorithms, researchers have proposed several accelerator schemes for encryption algorithms. Among them, due to its hardware friendliness, Montgomery modular multiplication algorithm [13] is the most widely used.

San et al. [14] implemented ModMult and ModExp accelerators using Montgomery modular multiplication algorithm and Karatsuba algorithm [15] on the embedded Xilinx Zynq 7000 equipped with Xilinx 7VX330T-3FPGA. For each ModMult and ModExp operation, the researchers put one compute pipeline on chip and designed the architecture respectively. The reported 2048-bit ModMult throughput is 366.3 kOP/s and the ModExp throughput is 176.1 OP/s. Then, in 2016, the same research team realized the acceleration of Paillier algorithm on the same platform [16], and the throughput of 512-bit and 1024-bit Paillier encryption was 680.3 OP/s and 88.7 OP/s respectively.
Dai et al. [17] designed an Fast Fourier Transform (FFT)-based McLaughlin’s Montgomery Exponentiation accelerator, involving no conditional selection against timing attacks and reported 396.8 OP/s and 132.6 OP/s for 1024 and 2048-bit ModExp respectively when implemented on Xilinx XC6VLX240T-3 FPGA.

Milad et al. [18] proposed a Paillier algorithm accelerator of HW/SW co-design, which also uses the Montgomery modular multiplication algorithm and deploys it on Xilinx Zynq 7020 SOC. They reported the throughput of 1024-bit ModMult and ModExp as 730.9 OP/s and 237 OP/s respectively, and the throughput of 2048-bit Paillier encryption is 65.8 OP/s.

A recent study by Yang et al. [19] designed an FPGA Paillier algorithm accelerator, implemented the Paillier algorithm on FPGA using Montgomery modular multiplication algorithm, and deployed on AWS F1 instance equipped with Xilinx XC6VU9P FPGA. With the help of more advanced FPGA hardware, under the condition of 1024-bit key bit width, their throughput of each Paillier encryption and decryption operation is about 5.2 kOP/s, which is 10.62x and 2.76x faster than CPU. Compared with CPU, the end-to-end training time is reduced by about 1.3x after being integrated into FATE [20].

For the application of federated learning, the existing research has the following disadvantages:

- The majority of the recent studies like [14], [16], [17] and [19] designed fixed circuits: Without reprogramming FPGA, they can only accelerate one specific algorithm. This strategy makes these accelerators unable to be applied to multiple PHE algorithms and other time-consuming tasks like ciphertext-space computation in federated learning at the same time, thus limiting the improvement of end-to-end training speed. For example, in the federated training process protected by Paillier algorithm, in addition to the Paillier encryption/decryption, the operation of aggregating the encrypted gradient will also bring a lot of workloads from ModMult operation. In addition, before the training begins, it usually needs to align the samples owned by each participant, which requires RSA algorithm, and involves ModExp operation.

- Many researchers [14], [16], [18] have made great efforts to optimize the Digital Signal Processor (DSP) efficiency of accelerators and verify them on embedded FPGA devices. However, secure federated learning is usually carried out among governments, banks, telecom operators, hospitals and other enterprises, and the data of all participants are stored in data center servers, which causes a big gap between these studies and practical applications. High power data center servers can be equipped with higher-end FPGA boards and provide sufficient DSP resources. Therefore, designing an accelerator suitable for high-end FPGA and achieving the highest throughput is a problem that needs to be stressed in the current scene.

![FIGURE 1. Breakdown of the execution time of each procedure of the federated learning flow.](image)

### III. REVIEW AND ANALYSIS OF ENCRYPTION ALGORITHMS IN FEDERATED LEARNING

To better illustrate the performance bottleneck of federated learning, we tracked and recorded the time taken to perform a two-party vertical federated learning task on the CPU, and quantitatively analyzed the time cost. There are 14 iterations from training start to convergence and the result is shown in Fig. 1.

It can be seen from Fig. 1 that encryption, decryption and ciphertext-space calculation account for 86.4% of the total time, whereas key generation and other operations (including some plaintext calculation, gradient update and local data access time) account for only 13.6%. This is due to the sharp increase in the overall amount of computation caused by cryptography related operations. In addition to the parts shown in the figure, the datasets of all parties need to be aligned through privacy intersection calculation before the training stage. The time-consuming of this process is closely related to the amount of data of all participants, and it will significantly consume additional time when the datasets are large.

Through the above analysis, accelerating cryptography related computation has become a key factor in accelerating federated learning. In the mature federated learning framework such as FATE [20], RSA [5] algorithm is used in the dataset alignment phase and Paillier [4] algorithm is used in the training phase. Therefore, the related operations involved in these two PHE algorithms are the main acceleration targets of this work. Next, we review these two algorithms with specific application scenarios, and summarize the characteristics of federated learning tasks, which leads to the motivation of this paper.

#### A. REVIEW OF THE RSA ALGORITHM

RSA algorithm [5] was proposed in 1977, and then widely used in the field of encryption and digital signatures.

1) **KEY GENERATION**

First, two different large prime numbers $p, q$ are randomly selected. Let

\[ n = p \cdot q \] (1)

and the Euler function $\varphi(n)$ can be computed as:

\[ \varphi(n) = (p - 1)(q - 1) \] (2)
Then, select an positive integer e that coprime with \( \phi(n) \) and smaller than \( \varphi(n) \), and the private key d can be obtained by calculating the modular multiplicative inverse of e for \( \varphi(n) \) by the following equation:

\[
d \equiv e^{-1} \pmod{\varphi(n)}
\]  

(3)

where “\( \equiv \)” means \( d \) is congruent to \( e^{-1} \) modulo \( \varphi(n) \). Thus, we can obtain key pair \((e, n)\) as public key, and \((d, n)\) as private key. In this phase, the purpose of selecting \( p \) and \( q \) is to make the Euler function \( \varphi(n) \) of \( n \) easy to calculate. After generating the key, \( p \) and \( q \) is discarded, which can ensure that no one can calculate \( \varphi(n) \) through \( n \) within a reasonable time, thus it is impossible to obtain the private key \( d \).

2) ENCRYPTION/DECRYPTION AND HOMOMORPHIC MULTIPLICATION

The encryption process of RSA is ModExp operation using the public key \((e, n)\) and plaintext \( m \) to obtain the ciphertext \( c \), in this paper we let the encryption of plaintext \( m \) be \( [m] \):

\[
c = [m] = m^e \mod n
\]  

(4)

Due to the irreversibility of modular operation, plaintext \( m \) cannot be deduced from ciphertext \( c \) and public key \((e, n)\).

The decryption process of RSA is very similar to the encryption. The only difference is that the exponentiation of ModExp is replaced by the private key \( d \):

\[
m = c^d \mod n
\]  

(5)

Given \( m_1 \) and \( m_2 \) are two plaintext and both are encrypted by the same RSA public key, then:

\[
[[m_1]] \otimes [[m_2]] = ([[m_1]] \cdot [[m_1]]) \mod n
 = [[m_1 \cdot m_2]]
\]  

(6)

where “\( \otimes \)” represents homomorphic multiplication. It can be seen from the above formula that each homomorphic multiplication can be expressed by once ModMult operation.

3) RSA ALGORITHM IN FEDERATED LEARNING

In federated learning, RSA algorithm is used in the stage of data alignment, aiming to find the intersection of datasets contained by multiple participants, while ensuring that this process does not disclose the data privacy of any party. To ensure sufficient security, the bit width of RSA key is usually 1024-bit or 2048-bit.

As shown in Fig.2, the dataset owned by Party A and Party B are denoted as \( X_A \) and \( X_B \) respectively. \( H(x) \) is the hash value of data \( x \). The data alignment process involves two participants using their own keys to perform blinding, signing and unblinding respectively. The first two steps mentioned involve ModExp operations and are the most time-consuming parts of the whole process, as indicated by the orange rounded rectangle in the figure. The time consumption of these steps will increase linearly with the increase of the dataset.

FIGURE 2. Flow diagram of sample alignment process with RSA algorithm in federated learning.

B. REVIEW OF THE PAILLIER ALGORITHM

Paillier algorithm [4] was first proposed by researchers in 1999. It is an additive homomorphic encryption algorithm which means the ModMult in the ciphertext-space is equivalent to addition in the plaintext-space.

1) KEY GENERATION

This process is used to generate public key \((n, g)\) and private key \((\lambda, \mu)\). The selection of \( p \) and \( q \) is the same as RSA algorithm. Then, an integer \( g \) is selected to satisfy \( g \in \mathbb{Z}_n^\times \) (multiplicative group of integers modulo \( n^2 \)), in FATE \( g \) is selected as \( g = n + 1 \) to simplify computation.

\( \lambda \) is the least common multiple (lcm) of \((p - 1) \) and \((q - 1) \):

\[
\lambda = \text{lcm}(p - 1, q - 1)
\]  

(7)

and \( \mu \) can be calculated by:

\[
\mu = L(g^{\frac{1}{\lambda}} \mod n^2)^{-1} \mod n
\]  

(8)

where,

\[
L(x) = (x - 1)/n
\]  

(9)

2) ENCRYPTION/DECRYPTION AND HOMOMORPHIC ADDITION

After the keys are generated, a random number \( r \) is selected to cooperate with the public key \((n, g)\) to perform encryption:

\[
c = [m] = g^mr^n \mod n^2
\]  

(10)

where the random number \( r \) should meet \( 0 < r < n \) and \( r \in \mathbb{Z}_{n^2}^\times \).
The process of decrypting ciphertext $c$ into plaintext $m$ requires private key pair $(\lambda, \mu)$:

$$m = \mu \cdot L(e^\lambda \mod n^2) \mod n \quad (11)$$

The homomorphic addition process, denoted by “⊕”, of two ciphertext $[m_1]$, $[m_2]$ is as follows:

$$[m_1] ⊕ [m_2] = ([m_1] \cdot [m_2]) \mod n^2 = [m_1 + m_2] \quad (12)$$

Thus, each homomorphic addition can be expressed by one ModMult operation.

3) PAILLIER ALGORITHM IN FEDERATED LEARNING

Taking vertical federated learning as an example, after the datasets are aligned, an ID set of the same samples owned by both participants $A$ and $B$ has been found. Both parties have different dimensional characteristics of the same ID, and only one party has label. Under the protection of Paillier algorithm, the features and labels of the two parties can be jointly modeled without disclosing data privacy. The specific process of training is shown in Fig.3.

In Fig.3, $D$ is the ID set of data shared by participants, $x_i^A, x_i^B$ is the $i$-th sample in $A$ and $B$ respectively, $\Theta_A, \Theta_B$ is the model parameter to be learned, and $y$ is the label whereas $\hat{y}$ is the current prediction. The steps shown in the figure will continue to iterate until the loss function converges to the target threshold, or the training iterations or time reaches the set upper limit. As noted by orange rounded rectangles, cryptographic computations take the most time in the training process, which lead to the performance bottleneck.

C. ANALYSIS OF THE CHARACTERISTICS OF FEDERATED LEARNING TASKS

By reviewing and analyzing the PHE algorithms and application examples in federated learning, we summarized the characteristics of federated learning tasks as follows:

1) In different stages of federated learning tasks, a variety of PHE algorithms are needed. We found that the basic operations of encryption and decryption and ciphertext-space calculation of various algorithms are ModMult and ModExp operations, which means these computationally sensitive algorithms can be implemented by one or more ModMult and ModExp calling.

2) The update frequency of keys in PHE algorithms is low. Generally, during one federated learning task, the keys of RSA and Paillier algorithms will not change, which means that the keys only need to be generated and transmitted once, and can be used in the whole training life cycle.

3) A relatively large batch size is commonly used for training. Large batch size is conducive to improving the training speed, but too large batch size is detrimental to model generalization ability [21]. Thus, in federated learning, the batch size of common training tasks like logistic regression is usually set to about 10K for large datasets.

One by one corresponds to the above findings, this paper proposes the following innovations:

1) We proposed a HW/SW co-design method. On the FPGA device (HW part), we accelerated the common components of ModMult and ModExp operations. On the CPU (SW part), we implemented hardware accelerated ModMult and ModExp operations, and provided federated learning acceleration by parsing the target algorithm into multiple ModMult and ModExp module calls.

2) We arranged some pre-calculations to be executed on the CPU. The required key and the modular multiplicative inverse are pre-calculated at the beginning of the federated learning task and then passed to FPGA, which can effectively save FPGA hardware resources.

3) We proposed a combination of data parallelism and pipeline to efficiently process large batch data in federated learning. The proposed architecture solved the challenge that large batch operations cannot be performed due to the bit width limitation of FPGA DDR controller and on-chip resource.

IV. HW/SW CO-DESIGN

A. THE TARGET HETEROGENEOUS PLATFORM

OpenCL [22] is a development framework based on C/C++, which provides a high-level synthesis (HLS) method for developing FPGA applications. Fig. 4 shows the development process on the FPGA-based heterogeneous computing platform defined by OpenCL framework.
As shown in Fig. 4, FPGA is referred to as device or HW part, which is usually used to implement computationally intensive algorithms, and is connected to CPU through PCIe interface. The algorithm on the device is first modeled into OpenCL kernel functions, and then be compiled to generate a bit stream file by the HLS tools provided by the FPGA vendor, and later be mapped on the FPGA fabric.

The CPU is referred to as the host or SW part. It is used to control the processes such as platform management, context management and task queue management. It is also responsible for data exchange between host memory and global memory. The host uses specific application programming interfaces (APIs) defined in the library to realize the above control process, and pyOpenCL [23] provides APIs defined in python.

Host memory is the DDR memory that only CPU can access, which is located on the motherboard of data center, whereas global memory is the DDR memory located on the FPGA board, which is visible to both CPU and FPGA, and is used for data exchange between HW and SW.

The target platform of the proposed federated learning accelerator is the heterogeneous computing platform of data center, including a general purpose CPU as the host and FPGA hardware as the device, which is the same of Fig. 4. As mentioned above, we use the OpenCL kernel function to implement the HW part of the accelerator, and pyOpenCL to realize the SW Part, so as to facilitate the docking with the federated learning framework based on python.

B. THE OVERALL ARCHITECTURE AND HW/SW PARTITION

Based on the summary of related work in Section II and the analysis presented in Section III-C, we proposed a HW/SW co-designed accelerator.

The timeline of HW/SW partition is shown in Fig. 5. Since the basic operation of most cryptography related algorithms in federated learning is ModMult and ModExp, and they can be realized by Montgomery algorithm, on the FPGA device (HW part), we only accelerate Montgomery algorithm. We proposed an optimized Montgomery algorithm, referred to as Hardware-aware Montgomery Algorithm (HWMA), and deployed it into a Montgomery Multiply Unit (MMU) on FPGA hardware. A pair of Data Load/Store Units (DLU/DSU) are designed and implemented on FPGA as the interfaces between CPU software and FPGA hardware. These three units will be illustrated in the next subsection.

On the CPU (SW part), we proposed an Operator Scheduling Engine (OSE), in which realized the initialization of OpenCL framework, pre-calculation (e.g., key generation and modular multiplicative inverse computation), and resolved the target algorithm into multiple HWMA calls. It should be noted that while the FPGA performs HWMA operations, the OSE performs non computation intensive operations (e.g., plaintext operations and loss computing) at the same time. Therefore, the time consumption of these calculations can overlap to further improve the acceleration effect. Both the OpenCL initialization and pre-calculation phase need to be implemented only once in the entire federated learning, and the other phases will be executed multiple times according to different scenarios.

Each HWMA calling in Fig. 5 is realized by the following steps: First, the OSE sends instructions to the DLU through OpenCL framework and ensures that the required data is in the global memory. Next, the DLU on FPGA reads the data and send them into MMU for accelerating computation. Then, the results of MMU are transmitted back into global memory through the DSU, and will be used for subsequent calculation or fetched by OSE.

C. HW ARCHITECTURE

The accelerator hardware architecture proposed in this paper is divided into three main parts: Data Load Unit (DLU), Data Store Unit (DSU) and Montgomery Multiply Unit (MMU). The overall architecture of the designed on-chip system is shown in Fig. 6. All parameters in the accelerator design are shown in Table 1.
In the federated learning task, a major challenge is that the bit width of the key and ciphertext in PHE algorithms are usually longer than 1024-bit, while the HLS tools of different device vendors have inconsistent support for large bit width data. For example, Xilinx can support numbers of 4096-bit natively [24], whereas Intel supports a maximum of 64-bit [25]. In order to make the design suitable for FPGA boards of different manufacturers, we designed a data structure with large bit width composed of multiple 32-bit integer data to ensure that the bit width of all data paths does not exceed 64-bit. At the same time, considering that federated learning will calculate a batch of data simultaneously, the proposed data structure flexibly supports the data parallelism in single instruction multiple data (SIMD) manner by storing multiple large numbers into a SIMD vector and then processing them on multiple parallel processing elements (PEs). Fig. 7 shows the proposed data structure for big number storage and processing.

As shown in Fig. 7, each line with the same color represents one large number with a bit width of \( L \), and it is represented by \( K + 2 \) 32-bit integers. The extra two integers reserved are used to process carries in the inner computation. Data parallelism is represented with vector size \( vs \), which can be flexibly configured at compile time according to the resources of different FPGA boards. To ensure maximum burst length of off-chip memory access, we store integers that form one large number continuously.

2) DATA LOAD/STORE UNIT

The task of Data Load/Store Unit (DLU/DSU) is to realize the data interaction between FPGA device and CPU, which is controlled and launched by SW side. DLU and DSU form the data transmission path of FPGA on-chip computing resources and global memory. We designed command queues for DLU and DSU respectively, when computing tasks are enqueued by pyOpenCL host, DLU and DSU are launched at the same time.

Once being launched, DLU begins to access global memory with the predefined data structure, fetching the necessary data for FPGA computing. The task scheduler in DLU has two functions: One is controlling whether it is necessary to update the key to DSU and MMU according to \( F_u \), the other is controlling the number of required memory access according to \( V_{num} \). These two parameters are concatenated into \( ctrl \) parameters by bits and then be passed to the task scheduler in DLU by SW program through global memory, after, \( ctrl \) will be passed to DSU and MMU through on-chip control channel, which can reduce the number of control flows. For example, if a calculation task has a operation of batch size 4096, \( vs \) is 2 and there is no need to update key \( m \), then the task scheduler will receive \( V_{num} = 2048 \) and \( F_u = False \). Then, it will control the DLU to perform 2048 times off-chip memory accesses to data \( X \) and \( Y \), skip memory accesses of \( m \), and in the mean time, forward the control signal.

Each time a set of vectors is obtained, DLU will immediately write them into the corresponding first-in-first-out queue (FIFO) in FPGA for computing in MMU. After startup, DSU will first receive control instructions from DLU, and then keep reading the result FIFO under the control of task scheduler, until all calculation tasks are completed. DSU is also responsible for the final subtraction of modulus in Montgomery algorithm.

3) MONTGOMERY MULTIPLY UNIT

The task of Montgomery Multiply Unit (MMU) is to get the required data from input FIFOs, calculate with the optimized
Algorithm 1 Original Montgomery Algorithm

Input: \( x = (x_{L-1} \ldots x_0)_2, y = (y_{L-1} \ldots y_0)_2, m = (m_{L-1} \ldots m_0)_2 \)

Output: \( xy^{-1} \mod m \) \( \Rightarrow x, y \) are scalars

\[ m_{\text{inv}} \leftarrow -m^{-1} \mod 2 \]

\[ s \leftarrow 0 \]

for \( i = 0 \) to \( L - 1 \) do

\[ q_i \leftarrow (s_0 + x_0 y_i) m_{\text{inv}} \mod 2 \]

\[ s \leftarrow (s + x y_i + q m)/2 \]

if \( s \geq m \) then

\[ s \leftarrow s - m \]

return(\( s \))

Algorithm 2 The Proposed Hardware-Aware Montgomery Algorithm (HWMA)

Input: \( X = (x^1 \ldots x^{V_{\text{num}}}), Y = (y^1 \ldots y^{V_{\text{num}}}), m = (m_{K+1} \ldots m_0)_{232}, m_{\text{inv}} \)

Output: \( X Y R^{-1} \mod m \)

\( \Rightarrow \) each \( X, Y \) contains \( vs \cdot V_{\text{num}} \) scalars

read \( F_{\mu}, V_{\text{num}} \)

if \( F_{\mu} \) is True then

\[ \text{update} \, m, m_{\text{inv}} \]

for \( v = 1 \) to \( V_{\text{num}} \) do

\( \Rightarrow \) Outer loop

read \( x^v, y^v \)

\( \Rightarrow \) \( x^v, y^v \) are SIMD vectors

\( s, \tilde{s} \leftarrow 0 \)

\( \Rightarrow s, \tilde{s} \) are SIMD vectors

\[ q_0 \leftarrow (s_0 + x_0^v y_0^v) m_{\text{inv}} \mod 2^{32} \]

for \( i = 0 \) to \( K - 1 \) do

\( \Rightarrow \) Second loop

for \( j = 0 \) to \( K + 1 \) do

\( \Rightarrow \) Inner loop

\[ s_j \leftarrow x_j^v y_j^v + q m_j + s_j \]

if \( j = 2 \) then

\[ q_{i+1} \leftarrow (s_0 + x_0^v y_0^v) m_{\text{inv}} \mod 2^{32} \]

if \( j > 0 \) then

\[ s_j - 1 \leftarrow \tilde{s}_j \]

write \( s \) into FIFO

compare \( s \) and \( m \), write result into FIFO

Hardware-aware Montgomery Algorithm (HWMA) on circuit, and write the results into the corresponding result FIFO. The MMU is configured as an OpenCL auto-run kernel, which is launched automatically after the FPGA is powered on and remains running all the time. Like DLU and DSU, the task scheduler in MMU is responsible for the control of the processing flow.

As shown in Algorithm 1, Montgomery algorithm is widely used to accelerate ModMult. \( L \) is the bit width of a large number, and subscript 2 indicates the radix of the data.

We extended the original algorithm to the form of high radix 2^{32} to utilize DSP resources on FPGA chip, and optimized it to Algorithm 2. Bold variables in this paper represent SIMD vectors composed of vs big numbers.

In the algorithm, input \( X, Y \) consists of \( V_{\text{num}} \) SIMD vectors, each vector is referred to as \( x, y \), each vector has vs big numbers and each big number has the same structure as modular \( m \). \( K \) is the number of 32-bit integers, consistent with previous data structure definition in Section IV-C1, and it is determined by the bit width of the input data. For example, when the calculation process of 2048-bit input data needs to be accelerated, the value of \( K \) is 64. Compared with the original Montgomery algorithm, the optimizations in our HWMA are mainly as follows:

1) The frequency of updating \( m, m_{\text{inv}} \) is reduced. As is summarized in Section III-C, in the PHE algorithm of federated learning, the key will be updated only once during a training task and it usually appears as the modulus of Montgomery algorithm. Therefore, we enabled the accelerator to receive the control signal from the DLU and update the key only when needed, so as to avoid repeatedly transmitting the same key for multiple vectors.

2) We advance the update of the intermediate result \( q \) to \( j = 2 \) of the inner loop, so that the time of \( q \) update can coincide with the calculation of the next inner loop.

3) The multi-level pipeline and data parallelism are designed. All the loops in HWMA are pipelined. The outermost pipeline is used to calculate each vector pair of input \( x, y \), and the second loop and inner loop form the main loop of high-radix Montgomery algorithm. Input \( x, y \) and intermediate operators involved in the calculation are data parallel vectors composed of vs large numbers, which realizes parallel calculation in SIMD manner.

In addition to the above optimization in MMU, we decoupled the computation and data access successfully through on-chip FIFOs. Since the calculation of subtracting \( m \) at the end of the original Montgomery algorithm involves a branch structure, which will cause operations of different large numbers in one SIMD vector various, directly placing it in MMU will cause the outer loop fail to generate an efficient pipeline. Therefore, we moves it to DSU, and releases the branch structure in the loop through a mask calculation, thus successfully
Fig. 9. Top-level block diagram of the proposed Operator Scheduling Engine. The AP parses the target algorithm into multiple ModMult/ModExp Module calls, and each call of the module performs multiple HWMA calls to accelerate the calculation.

Pipeline all of the loops in the accelerator. Fig.8 shows the proposed computing pipeline in MMU.

D. SW DESIGN

On the SW side, we present an Operator Scheduling Engine (OSE) working on CPU to resolve different algorithms into multiple efficient calls to FPGA operators. As shown in Fig. 9, the OSE includes an OpenCL Initialize Module, an Algorithm Parser (AP), and a pair of ModMult and ModExp Module.

The detail workflow of the proposed OSE is shown in Fig.10. The initialization of OpenCL framework only needs to run once for a federated learning task and is completed by the OpenCL Initialize Module. Other modules including the AP and ModMult/ModExp Module will be called many times according to algorithms to be accelerated.

1) ALGORITHM PARSER

The Algorithm Parser (AP) is the first link between SW and HW. It is responsible for parsing the PHE related algorithms (including encryption, decryption and ciphertext-space computation) in federated learning into multiple ModMult and ModExp operations, and invoking the ModMult/ModExp Module on CPU accordingly, then ModMult/ModExp Module will call hardware operators on FPGA to perform high speed HWMA operations, thus accelerate the target algorithms. For example, for each encryption or decryption operation of RSA algorithm, the AP parses it as one call to the ModExp Module. In the real application, AP will use the pre-defined batch size and the $v_s$ of FPGA operator to automatically generates the number of HW required operator calls $V_{num}$. Table 2 shows the AP’s parsing of some commonly used algorithms in an industrial grade federated learning open source framework FATE [20].

It is worth noting that, thanks to the flexibility of the proposed architecture, when there is a need to accelerate a new algorithm other than the algorithms shown in Table 2, as long as the algorithm can be realized by ModMult or ModExp calculation, the acceleration can be realized by defining a new analytical method in AP at the SW side, without modifying or reprogramming any HW circuit on FPGA.

2) ModMult MODULE

The ModMult/ModExp Modules are the second link between SW and HW. The ModMult Module designed in this paper decomposes one ModMult calculation into four HWMA calculations through Algorithm 3, thus, provides hardware accelerated ModMult operations. We design two buffers $\hat{X}$, $\hat{Y}$ to cache intermediate results and realize the reuse of buffer $X$, $Y$. Each HWMA operation is realized by launching DLU and DSU once as described in Section IV-B.

The workflow of ModMult Module is shown in the orange part of Fig. 10. When key $n$ does not need to be updated, ModMult Module will transmit $F_u = False$ when calling DLU, and make FPGA skip the step of updating $m$. It should be noted that when a input parameter of HWMA is not composed of $V_{num}$ vectors (e.g., $p$ in Algorithm 3), the module will expand it into vector to meet the input requirements of hardware.

3) ModExp MODULE

The ModExp Module provides hardware accelerated ModExp operations. In order to efficiently call the designed FPGA operator to realize the ModExp function, we use

| Target Algorithm | #of ModMult Module call | #of ModExp Module call |
|------------------|------------------------|------------------------|
| RSA Encryption   | 0                      | 1                      |
| RSA Decryption   | 0                      | 1                      |
| RSA Multiplicationa | 1                     | 10b                    |
| Pailler Encryption | 1                     | 1                      |
| Pailler Decryption | 3                     | 2                      |
| Pailler Additiona | 1                      | 0                      |
| Pailler Scalar Multiplicationa | 0 | 1                    |

\* Cipher-text-space homomorphic computation.
\* Cipher-text-plaintext multiplication requires ModExp call for 1 time, ciphertext-ciphertext multiplication does not require ModExp call.
To evaluate the effectiveness of the proposed HW/SW on FPGA.

Since there is no data dependency between the multiplication and the exponentiation calculation in Algorithm 4, the proposed ModExp Module will add DLU/DSU for both operations to the command queue in the same cycle without blocking. In this way, the time of data access in the two calls can overlap, so as to ensure the continuous running of MMU on FPGA.

**V. EXPERIMENTAL RESULTS**

**A. EXPERIMENTAL SETUP**

To evaluate the effectiveness of the proposed HW/SW co-design, we deployed the final design on the Intel PAC D5005 FPGA Development Kit with an Intel Stratix 10 GX2800 FPGA on board. The server is equipped with an Intel Xeon E7-4830 V3 CPU with 128GB memory, and the FPGA card is connected to the server motherboard through PCIe. The software development environment is Intel FPGA SDK for OpenCL v20.3.

In the design of OpenCL HW code, we use `pragma unroll` instruction.

**B. PERFORMANCE OF ACCELERATING SPECIFIC ALGORITHMS**

Table 3 and Table 4 show the acceleration effect of our accelerator on common algorithms in federated learning, including ModMult/ModExp (RSA algorithm) and Paillier algorithm. In order to facilitate the comparison with other work, we have carried out experiments on the modular bit width of 1024-bit and 2048-bit respectively, and measured the throughput of the CPU as the baseline. When the modular bit width is increased from 1024-bit to 2048-bit, there are two main reasons for the decrease in throughput: The first is that a higher data bit width will lead to higher computational complexity. The second is that the data path and computation logic on the FPGA chip with a larger bit width will occupy more resources, resulting in lower parallelism. In our final deployment, vS is configured as 4 at 1024-bit modular bit width and 2 at 2048-bit to ensure a relatively high frequency.

Mark “-” in the table indicates that the result is not reported in the study, and “N/A” indicates that the design does not support the corresponding algorithm. From the comparison in Table 3 and Table 4, it can be seen that in the case of 1024-bit modulus, our PipeFL can achieve 4.0x, 2.1x and 2.6x performance improvement in ModMult, ModExp and Paillier algorithm respectively compared with the server CPU. When the modular bit width increases to 2048-bit, the improvement compared with the CPU increase to 5.3x, 3.0x and 3.2x respectively.

It can be seen from Table 3 and Table 4 that, our accelerator can support ModMult, ModExp and Paillier algorithm in the same time by using a unified hardware circuit, but most existing work [14], [16], [17], [19] are fixed to only one algorithm, which made them impossible to accelerate multiple algorithms involved in one federated learning task.

Besides, as the target platform of many works is embedded FPGA, and the parallelism is not designed for large-scale data to improve throughput, their reported performance is only slightly higher (e.g., work [14], [18]) or even worse (e.g., work [16], [17]) than that of our baseline implementation on CPU. It can be seen from Table 4 that Yang et al. [19] implemented the Paillier accelerator on a Xilinx high-end FPGA, and its accelerated performance on the Paillier algorithm can reach about 5200 OP/s, which is 3.49x than ours. There are two main reasons for their higher performance. One is that...
their design only supports the Paillier algorithm, so they can achieve more efficient on-chip data flow. The other is that their target FPGA has 19% more DSP resources and 41% more RAM resources than ours, which will further improve their performance. However, due to the lack of flexibility, when their accelerator is integrated into practical federated learning applications, the end-to-end performance is not as good as ours, which will be illustrated in detail in the next subsection.

C. END-TO-END PERFORMANCE IN REAL APPLICATIONS

In order to verify the effectiveness of our accelerator in practical federated learning applications, we integrated it into the existing open source federated learning training framework FATE [20].

As far as we know, only Yang et al. [19] have reported the performance in actual federated learning application. Therefore, we use the same dataset and training task for evaluation. Two datasets, Kaggle datasets on breast cancer [27] and motor temperature [28], are used to train linear regression and logistic regression models respectively.

As our CPU is more powerful than the Intel Xeon E5-2686 V4 used in comparative work, and we have made some streamlining in the FATE framework, when running federated learning training in pure SW, our implementation has about 30% performance improvement. In order to make the comparison more fair, we compared the performance improvement based on our respective CPU implementations, the results are shown in Fig.11.

Our PipeFL adopts a flexible HW/SW co-design method, which can speed up Paillier encryption, decryption and ciphertext-space calculation in federated learning at the same time, whereas the work of Yang et al. [19] can only speed up Paillier encryption and decryption. Therefore, although their work has higher performance in Paillier encryption and decryption than ours, its effect on end-to-end training tasks is not as significant as ours.

From the comparison results shown in Fig.11, we can see that our accelerator can achieve 1.33s and 1.02s for each iteration in end-to-end linear regression and logistic regression tasks respectively, achieving 2.28x and 3.30x time reduction than SW implementation. Compared with the previous work, our training time is reduced by 2.48x and 3.04x respectively.

VI. CONCLUSION AND FUTURE WORKS

Federated learning has been widely studied in recent years. Its privacy protection features make it suitable for China’s national cloud strategy and worldwide privacy protection.
policies like European Union’s General Data Protection Regulation. It has been applied in many scenarios such as government, telecom operators, finance and hospitals. However, the computational overhead caused by cryptography related algorithms is a key factor that restricts the large-scale commercial use of federated learning.

In this paper, a flexible HW/SW co-designed FPGA accelerator for federated learning is proposed. The accelerator can support the most time-consuming encryption, decryption and ciphertext-space calculation in secure federated learning in the same time without reprogramming FPGA, and has proved to be very effective.

We have quantitatively shown the time-consuming of each stage of the federated learning task, and have analyzed the main computational phases that need to be optimized. Through the review of the principle of PHE algorithms used in real federated learning applications, we put forward some theoretical bases which may guide accelerator designs in the future. We have also shown the HW/SW co-design implemented by OpenCL framework in detail and compared the performance with related work.

The accelerator proposed in this research has been used as an infrastructure of China Telecom cloud computing service. In the future, we will try to apply the designed accelerator to more private computing scenarios, and verify its acceleration effect on other algorithms based on ModMult and ModExp. In addition, we also plan to combine more technologies, such as compression and sparse coding to further improve performance.

**ACKNOWLEDGMENT**

The authors would like to thank the anonymous reviewers for their valuable comments and constructive suggestions, which helped in improving the quality of the paper.

**REFERENCES**

[1] European Union. General Data Protection Regulation. Accessed: Aug. 1, 2022. [Online]. Available: https://gdpr-info.eu/

[2] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, “Communication-efficient learning of deep networks from decentralized data,” in Proc. Int. Conf. Artif. Intell. Stat. (AISTATS), vol. 54, A. Singh and J. Zhu, Eds., Apr. 2017, pp. 1273–1282. [Online]. Available: https://proceedings.mlr.press/v54/mcmahan17a.html

[3] Q. Yang, Y. Liu, T. Chen, and Y. Tong, “Federated machine learning: Concept and applications,” ACM Trans. Intell. Syst. Technol., vol. 10, no. 2, pp. 1–19, 2019.

[4] P. Paillier, Public-Key Cryptosystems Based on Composite Degree Resid- uosity Classes (Lecture Notes in Computer Science), J. Stern, Ed. Berlin, Germany: Springer, 1999, pp. 223–238.

[5] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” Commun. ACM, vol. 21, no. 2, pp. 120–126, Feb. 1978, doi: 10.1145/359340.359342.

[6] Q. Jing, W. Wang, J. Zhang, H. Tian, and K. Chen, “Quantifying the performance of federated transfer learning,” 2019, arXiv:1912.12795.

[7] W. Jung, T. T. Dao, and J. Lee, “DeepCuts: A deep learning optimization framework for versatile GPU workloads,” in Proc. ACM SIGPLAN Conf. Program. Language Design Implement. (PLDI), 2021, pp. 190–205.

[8] S. Tan, B. Knott, Y. Tian, and D. J. Wu, “CryptGPU: Fast privacy-preserving machine learning on the GPU,” in Proc. IEEE Symp. Secur. Privacy (SP), May 2021, pp. 1021–1038.

[9] H. Hu, W. Wang, C. Chen, Y. Lee, B. Lin, H. Wang, Y. Lin, Y. Lin, C. Hsieh, C. Hu, Y. Lai, H. Chen, Y. Chang, H. Li, T. Kuo, K. Wang, M. Chang, C. Hung, and C. Lu, “A 512Gb in-memory-computing 3D-NAND flash supporting similar-vector-matching operations on edge-AI devices,” in IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers, vol. 65, Feb. 2022, pp. 138–140.

[10] Z. Wang, K. Xu, S. Wu, L. Liu, L. Liu, and D. Wang, “Sparse-YOLO: Hardware/software co-design of an FPGA accelerator for YOLOv2,” IEEE Access, vol. 8, pp. 116569–116585, 2020.

[11] D. Wang, J. An, and K. Xu, “PipeCNN: An OpenCL-based FPGA accelerator for large-scale convolution neuron networks,” 2016, arXiv:1611.02450.

[12] D. Wang, K. Xu, Q. Jia, and S. Ghiasi, “ABM-SpConv: A novel approach to FPGA-based acceleration of convolutional neural network inference,” in Proc. 56th ACM/IEEE Design Autom. Conf. (DAC), Jun. 2019, pp. 1–6.

[13] P. L. Montgomery, “Modular multiplication without trial division,” Math. Comput., vol. 44, no. 170, pp. 519–521, Apr. 1985.

[14] I. San and N. At, “Improving the computational efficiency of modular operations for embedded systems,” J. Syst. Archit., vol. 60, no. 5, pp. 440–451, 2014.

[15] A. Karatsuba and Y. Ofman, “Multiplication of multidigit numbers on automata,” Sov. Phys. Doklady, vol. 7, no. 5, pp. 595–596, 1963.

[16] I. San, N. At, I. Yakut, and H. Polat, “Efficient Paillier cryptoprocessor for privacy-preserving data mining,” Secur. Commun. Netw., vol. 9, no. 11, pp. 1535–1546, 2016.

[17] W. Dui, D. Chen, R. C. Cheung, and C. K. Koc, “FFT-based McLaughlin’s Montgomery exponentiation without conditional selections,” IEEE Trans. Comput., vol. 67, no. 9, pp. 1301–1314, Mar. 2018.

[18] M. Bahadori and K. Järvinen, “A programmable SoC-based accelerator for privacy-enhancing technologies and functional encryption,” IEEE Trans. Very Large Scale Integrate (VLSI) Syst., vol. 28, no. 10, pp. 2182–2195, Jul. 2020.

[19] Z. Yang, S. Hu, and K. Chen, “FPGA-based hardware accelerator of homomorphic encryption for efficient federated learning.” 2020, arXiv:2007.10560.

[20] Linux Foundation. Federaed AI Technology Enabler. Accessed: Aug. 1, 2022. [Online]. Available: https://github.com/FederaedAI/FATE

[21] N. S. Keskar, D. M. M. Nocedal, M. Smelyanskiy, and P. T. P. Tang, “On large-batch training for deep learning: Generalization gap and sharp minima,” 2016, arXiv:1609.04836.

[22] Khronos Group. Open Standard for Parallel Programming of Heterogeneous Systems. Accessed: Aug. 1, 2022. [Online]. Available: https://www.khronos.org/opencl/

[23] Python Software Foundation. Python Wrapper for Opencl. Accessed: Aug. 1, 2022. [Online]. Available: https://pypi.org/project/pyopencl/

[24] Xilinx. Corp. Vitis High-Level Synthesis User Guide. Accessed: Aug. 1, 2022. [Online]. Available: https://docs.xilinx.com/en/en-US/vitis/1399-vitis-lhs

[25] Intel Corp. Intel FPGA SDK for Opencl Pro Edition Programming Guide V20.3. Accessed: Aug. 1, 2022. [Online]. Available: https://www.intel.com/

[26] A. Daly and W. Marnane, “Efficient architectures for implementing Montgomery modular multiplication and RSA modular exponentiation on reconfigurable logic,” in Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays (FPGA), 2002, pp. 40–49.

[27] KIRGSN. Breast Cancer Wisconsin (Diagnostic) Data Set. Accessed: Aug. 1, 2022. [Online]. Available: https://www.kaggle.com/uic/mlcancer-wisconsin-data

[28] UCI MACHINE LEARNING. Electric Motor Temperature. Accessed: Aug. 1, 2022. [Online]. Available: https://www.kaggle.com/wskirgns/electric-motor-temperature

ZIXIAO WANG received the M.S. degree in signal and information processing from the Beijing Key Laboratory of Advanced Information Science and Network Technology, Beijing Jiaotong University, Beijing, China, in 2021. He is currently working as an AI Algorithm Engineer at the China Telecom Research Institute, Beijing. His research interests include heterogeneous computing systems, neural network compression, and high performance computing architectures for deep learning applications.
BIYAO CHE received the B.S. degree in financial mathematics from the Beijing University of Chemical Technology, Beijing, China, in 2018, and the M.S. degree in statistics from Beijing Jiaotong University, Beijing, in 2021. She currently works as an AI Algorithm Engineer at the China Telecom Research Institute, Beijing. Her research interests include mathematics, statistics, deep learning, and interpretability of neural networks.

LIANG GUO is currently a Senior Engineer (Professor-level) and the Deputy Chief Engineer of the Institute of Cloud Computing and Big Data of CAICT, Beijing, China. He is mainly engaged in policy support, technical research, and standard setting related to computational infrastructure. He wrote four monographs and published more than 20 journal articles.

YING CHEN received the B.S. and M.S. degrees in control theory from the Institute of Robotics and Intelligent Equipment, Beijing University of Technology, Beijing, China, in 2012 and 2019, respectively. He was the Computer Vision Algorithm Engineer at Beijing Vion Technology Inc., Beijing, from 2019 to 2021. He currently works as an AI Algorithm Engineer at the China Telecom Research Institute, Beijing. His research interests include neural network algorithm and architecture, computer vision, high performance computing, and optimization theory.

JIZHUANG ZHAO received the M.S. degree from the Beijing Institute of Technology, Beijing, China, in 2006. He currently works as a Senior Engineer at the China Telecom Research Institute, Beijing. He is also the Deputy Leader of the Advance Technology Testing Group, Open Data Center Committee (ODCC), and the Chief of the Requirements & Architecture Group, Diversified Computing Industry Alliance (DICA). His research interests include cloud computing and software/hardware optimization of high-performance computing.

YANG DU received the M.S. degree in computer science and technology from the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China, in 2021. He currently works as an AI Algorithm Engineer at the China Telecom Research Institute, Beijing. His research interests include neural network compression, and high performance computing architectures for deep learning applications.

WEI HE received the bachelor’s and master’s degrees from Beijing Jiaotong University, China, in 2006 and 2009, respectively, and the Ph.D. degree from the Universidad Politecnica de Madrid, Spain, in 2014. He was a Research Scientist at Nanyang Technological University, Singapore, from 2014 to 2016, and then he served as a Senior Researcher at Huawei International, Singapore, until January 2019. He is currently leading the Blockchain Research Institute, China Telecom BestPay Company Ltd. His research interests include crypto security, blockchain, privacy-preserving computation, hardware design, and with over 40 published academic papers.

* * *