High-performance Hardware Acceleration Architecture for Cross-silo Federated Learning Supplementary Materials

Junxue Zhang, Xiaodian Cheng, Liu Yang, Jinbin Hu, Han Tian, Kai Chen

APPENDIX

1 CROSS-SILO FEDERATED LEARNING

Cross-silo federated learning (FL) denotes the scenario where companies or institutions collaboratively train machine learning models without data privacy leakage [1], [2], [3], [4], [5], [6]. Compared with cross-device FL, where participants are mobile devices, cross-silo FL focuses more on data security and incentive mechanism. From the data partition angle, Yang *et al.* proposed to categorize FL into horizontal FL, vertical FL, and federated transfer learning [7]. Federated transfer learning has rarely been applied in the industry and remains in the research stage. In most cases, cross-device FL contains only horizontal FL, while cross-silo FL usually contains both horizontal and vertical FL. Because of the different data partition situations, horizontal and vertical FL are different in model construction, protocol design as well as the utilized cryptographic systems.

1.1 Cross-silo Horizontal FL

Participants in horizontal FL have different sample ID spaces but the same feature space. Each participant owns the labels of their samples. Therefore, horizontal FL enlarges the number of training samples to train a model with better generalization ability. In most cases, there is a third-party central server for parameter aggregation.

The t^{th} iteration of the training process among three participants is shown as follows:

1. All participants negotiate about keys for encryption.
2. Each participant i trains local model $\boldsymbol{w}_i^t$ with its own samples and encrypts its model weights to $[[\boldsymbol{w}_i^t]]$ with either additively homomorphic encryption (*e.g.*, Paillier [8]).
3. Each participant i sends the encrypted weights to the central server.
4. The server receives the encrypted local model weights from all participants and aggregates them to global model weights $\frac{\sum_i [[\boldsymbol{w}_i^t]]}{n}$, where n stands for the number of participants. Because the weights are encrypted via additively homomorphic encryption, we can directly perform an aggregation over the ciphertext.
5. The central server sends the aggregated global weights to all participants.
6. Each participant receives the global weights and decrypts them locally. Then the participant can update its local model $\boldsymbol{w}_i^{t+1}$ with the decrypted global model.

After the federated training process, each participant obtains the same well-trained model. Thus, each participant can perform model inference locally.

1.2 Cross-silo Vertical FL

Participants in vertical FL scenarios have the same sample ID space but different feature spaces. Under normal circumstances, only one participant holds the label of the FL task, which is called the active party. The other participants without labeling information are called passive parties. Compared with horizontal FL, vertical FL could enrich the feature information of samples. Unlike horizontal FL, the training process of vertical FL is conducted after the entity alignment stage, which aligns common samples while protecting privacy. Besides, the training schema of vertical FL is also different from horizontal FL. More specifically, each participant only owns part of the model parameters corresponding to the local feature dimensions. Hence, vertical FL cannot simply conduct the secure aggregation as horizontal FL does. In addition, various machine learning algorithms do not have a unified design of the vertical FL implementation.

Taking the federated linear regression [1] between two participants as an example, we illustrate the training process as follows:

1. Participants and the third-party central server negotiate about keys for encryption.
2. Passive party B computes local point estimate $u_{B,j}^t$ and partial loss $L_{B,j}^t$ for the j^{th} aligned sample, then encrypts them to $[[u_{B,j}^t]]$ and $[[L_{B,j}^t]]$ with Paillier [8]. Active party A calculates local point estimate $u_{A,j}^t$.
3. Passive party B sends the encrypted numbers to the active party A.
4. Active party A receives the encrypted numbers and computes the total loss $[[L_j^t]]$ and the intermediate results $[[d_j^t]]$ used to calculate gradients.
5. Active party A sends $[[L_j^t]]$ to server and $[[d_j^t]]$ to passive party B.
6. Party B and party A separately compute encrypted gradients $[[\frac{\partial L_j^t}{\partial \boldsymbol{w}_B^t}]]$ and $[[\frac{\partial L_j^t}{\partial \boldsymbol{w}_A^t}]]$ and add random masks.

7. Both parties send the encrypted and masked gradients to the central server.
8. The third-party central server decrypts the received ciphertext to get the plain-text masked gradients and sends them back.
9. Party B and party A respectively remove the random masks from gradients and update the local partial model.

All participants of vertical FL should be involved in the inference stage since each of them only owns part of the whole model.

1.3 Security Analysis of Cross-silo FL

The adopted FL algorithms in this paper are proved secure under the semi-honest assumption [1], [9]. The semi-honest assumption means that each party does not violate the federated protocols and only tries to infer the sensitive data of other parties from the received messages. For the horizontal FL models, the transmitted model updates are protected by additively HE for aggregation. Therefore, nothing can be learned by the arbiter. Moreover, each party obtains the aggregated model updates and can only calculate the average model updates of the other parties. Hence, given more than two parties, the model updates computed over the local data of one party cannot be leaked to the other parties [1]. For the vertical federated linear models, the transmitted intermediate results are protected by random masks and HE, which reveals no information. Furthermore, from the obtained model updates, one party cannot infer the sensitive data of other parties without prior knowledge of their data structures [1]. For the vertical SecureBoost model, the active party with labels could learn some information agreed in advance, such as the instance spaces and the responsible parties of splits. However, under the protection of HE, the original data records of one party cannot be revealed to other parties, either [9].

2 Paillier Cryptosystem

Paillier Cryptosystem [10] is a widely-used additively (*i.e.*, partially) homomorphic encryption scheme. Paillier cryptosystem supports two kinds of operations, including the addition of two values of ciphertext and multiplication between ciphertext and cleartext. We will introduce Paillier key generation, encryption and decryption respectively in the following section.

Key Generation: Key generation of Paillier Cryptosystem follows the following steps.

1. Choose two random prime numbers p and q which satisfy that $\gcd(pq, (p-1)(q-1)) = 1$, where gcd stands for the greatest common divisor.
2. Compute $n = p \cdot q$.
3. Compute $\lambda = \text{lcm}(p-1, q-1)$, where lcm means the least common multiple.
4. Randomly select an integer g which satisfies that $\gcd(\text{L}_n(g^\lambda \mod n^2), n) = 1$. Function $\text{L}_n(x)$ is defined as $\text{L}_n(x) = (x-1)/n$.
5. Compute $\mu = [\text{L}_n(g^\lambda \mod n^2)]^{-1} \mod n$.

After the above computation, we will obtain the public key: (n, g) and private key: (λ, μ) respectively.

Encryption: The encryption algorithm of Paillier is straightforward and follows the following equation.

$$c = g^m \cdot r^n \mod n^2. \tag{2.1}$$

Optimization of Encryption: The encryption can be accelerated by assigning public key g as $n+1$. Therefore, the encryption algorithm is simplified as follows.

$$\begin{aligned} c &= g^m \cdot r^n \mod n^2 \\ &= (n+1)^m \cdot r^n \mod n^2 \\ &= [(\sum_{i=0}^{m} \binom{m}{i} \cdot n^i) \cdot r^n] \mod n^2 \\ &= [(1 + m \cdot n) \cdot r^n] \mod n^2 \end{aligned} \tag{2.2}$$

One modular exponentiation operation is saved by this optimization. FLASH uses the optimized encryption for better performance.

Decryption: Paillier ciphertext c is decrypted to plaintext m with both public key (n, g) and private key (λ, μ):

$$m = \text{L}_n(c^\lambda \mod n^2) \cdot \mu \mod n \tag{2.3}$$

Optimization of Decryption: The workload of the decryption algorithm of Paillier can be reduced with the Chinese Remainder Theorem (CRT). In this scheme, prime numbers p and q generated with the key pair are considered as the private key. The process of optimized decryption is shown below:

1. Compute $h_p = \text{L}_p(g^{p-1} \mod p^2)^{-1} \mod p$ and $h_q = \text{L}_q(g^{q-1} \mod q^2)^{-1} \mod q$.
2. Compute $m_p = \text{L}_p(c^{p-1} \mod p^2) \cdot h_p \mod p$ and $m_q = \text{L}_q(c^{q-1} \mod q^2) \cdot h_q \mod q$. Function $\text{L}_p(x)$ and $\text{L}_q(x)$ are defined by $\text{L}_p(x) = (x-1)/p$ and $\text{L}_q(x) = (x-1)/q$. It can be proved that $m_p = m \mod p$ and $m_q = m \mod q$, where m is the plaintext corresponding to ciphertext c.
3. Apply CRT to recombine the modular residues. $m = \text{CRT}(m_p, m_q) \mod pq$.

With the optimization above, the workload can be reduced to only about one-quarter of the original decryption algorithm, leading to better performance. FLASH also uses optimized decryption in its implementation.

3 RSA Intersection

RSA (Rivest–Shamir–Adleman) is an asymmetric public-private key method used to securely transfer data [11]. The whole RSA algorithm mainly contains three operations: key generation, encryption, and decryption.

Key Generation: The generation process is shown below:

1. Randomly choose two distinct prime numbers p and q.
2. Compute $n = p \cdot q$.
3. Compute $\lambda(n) = \text{lcm}(p-1, q-1)$.
4. Randomly choose a number e such that $1 < e < \lambda(n)$ and $\gcd(e, \lambda(n)) = 1$.
5. Compute d by solving $d \cdot e = 1 \mod \lambda(n)$.

Generally speaking, (n, e) is regarded as a public key, while d is regarded as a private key.

Encryption: Using public key (n, e), plain-text message m is encrypted to cipher-text message c:

$$c = m^e \mod n. \tag{3.4}$$

Decryption: Using private key d, cipher-text message c is decrypted to plain-text message m:

$$m = c^d \mod n. \tag{3.5}$$

RSA-based PSI: The RSA-based private set intersection can protect the privacy of sample ID out of the intersection set with the mechanism of blind RSA signature [12], [13]. We take the two-party setting as an example. Party A contains three user IDs, *i.e.*, $\mathcal{X}_A = \{x_1, x_2, x_3\}$, while party B contains four user IDs, *i.e.*, $\mathcal{X}_B = \{x_1, x_2, x_4, x_5\}$. They want to find their common users via RSA-based intersection:

1. Party A generates RSA keys n, e, d and sends public key (n, e) to party B.
2. Party B blinds and encrypts its user IDs $\mathcal{X}_B$ to $\mathcal{Y}_B = \{H(x) \mod n \cdot r^e \mod n) \mid x \in \mathcal{X}_B\}$, where r is a unique random number for each x, and sends $\mathcal{Y}_B$ to party A.
3. Party A signs the received $\mathcal{Y}_B$, obtains $\mathcal{Z}_B = \{y^d \mod n = r \cdot H(x)^d \mod n \mid y \in \mathcal{Y}_B\}$ and sends $\mathcal{Z}_B$ to party B.
4. Party A also signs its own user IDs, gets $\mathcal{D}_A = \{H(H(x))^d \mid x \in \mathcal{X}_A\}$ and sends $\mathcal{D}_A$ to party B.
5. Party B unblinds the received $\mathcal{Z}_B$ and obtains $\mathcal{D}_B = \{H(z/r \mod n) = H(H(x))^d \mid z \in \mathcal{Z}_B\}$.
6. Party B computes $\mathcal{D}_A \cap \mathcal{D}_B = \{H(H(x_1))^d, H(H(x_2))^d\}$ and gets common user IDs $\{x_1, x_2\}$.

$H(\cdot)$ denotes the hash function. After party B knows the overlapping users, it could choose whether to inform party A according to different scenarios.

4 Modular Exponentiation & Multiplication Algorithm Optimization

4.1 Binary Exponentiation

Modular exponentiation is defined as $P = m^e \mod N$. In the naïve algorithm, m is multiplied by itself for e times, and the algorithm uses $e - 1$ multiplications to obtain the result. Therefore, if e is a large integer, the computation time is dramatic. As a result, to optimize the computation, people usually apply binary exponentiation optimization to reduce the dramatic computation time. Algorithm 4.1 shows the process of the binary exponentiation optimization algorithm.

The idea of binary exponentiation is to reduce the number of multiplications by using the binary representation of the exponent e. As a result, we only need to compute at most $2\lfloor \log_2 e \rfloor$ multiplications, which is much smaller than $e - 1$. Since the time complexity of modular exponentiation is determined by the number of multiplications, binary exponentiation can reduce its time complexity from $\mathcal{O}(e)$ to $\mathcal{O}(\log e)$. Worth mentioning, the modulo computation can be performed after each multiplication because of the distribution law in modular arithmetic: $(a \mod N)(b \mod N) \equiv ab \mod N$.

Algorithm 4.1 Binary Exponentiation

Input: m, e, N, where $N > 0$
Output: $P = m^e \mod N$
1: $P = 1$ ▷ Initialization
2: **while** $e > 1$ **do**
3: **if** e is odd **then**
4: $P = P \cdot m \mod N$
5: **end if**
6: $e = e \gg 1$
7: $m = m^2 \mod N$
8: **end while**
9: **return** P

Summary: By using the binary exponentiation optimization algorithm, we can largely optimize the time complexity of modular exponentiation computation.

Algorithm 4.2 Montgomery Modular Multiplication

▷ Given three input numbers X, Y and N, the Montgomery Modular Multiplication outputs $Z = X \cdot Y \cdot R^{-1} \mod N$, where R is a power of 2 and $\lfloor \log_2 R \rfloor = \lfloor \log_2 N \rfloor$.
Input: $X = (X_{d-1}, ..., X_0)$, $Y = (Y_{d-1}, ..., Y_0)$, $N = (N_{d-1}, ..., N_0)$, N', where
$N' = (-N)^{-1} \mod r$, ▷ N' is pre-computed in S1
$r = 2^w$, $d = \lfloor \log_r N \rfloor + 1$, ▷ r and d is used to split data
$\gcd(N, r) = 1$, with $N \times N' \equiv -1 \mod r$
Output: $Z = \text{ModMult}(X, Y, N) = X \times Y \times R^{-1} \mod N$
1: $Z = (Z_{d-1}, ..., Z_0) = 0$ ▷ Initialization
2: **for all** $i = 0, 1, ..., d-1$ **do** ▷ Loop on Y
3: $q = (Z_0 + X_0 \times Y_i) \times N' \mod r$
4: $C = 0$
5: **for all** $j = 0, 2, ..., d-1$ **do** ▷ Loop on X
6: $S = Z_j + X_j \times Y_i + q \times N_j + C$
7: **if** $j > 0$ **then**
8: $Z_{j-1} = S \mod r$
9: **end if**
10: $C = S \gg w$ ▷ Carry higher bits
11: **end for**
12: $Z_{d-1} = C$
13: **end for**
14: **if** $Z \geq N$ **then**
15: $Z = Z - N$
16: **end if**
17: **return** Z

4.2 Montgomery Modular Multiplication

After applying the binary exponentiation optimization algorithm as shown in §4.1, we lower the time complexity of modular exponentiation computation by reducing the number of multiplications. However, after each multiplication, we have to perform one modulo operation. Although we can implement modulo operation on hardware with Cyclic Reduction and Barrett Reduction algorithms [14], the performance of these algorithms is still not satisfying because of the division operations used in these algorithms. Therefore, FLASH utilizes Montgomery Modular Multiplication [15] to replace the modulo operation with a bit-shifting operation, which is more hardware-friendly.

The process of applying Montgomery Modular Multiplication includes three major steps: (1) converting the data into Montgomery space, (2) computing the modular multiplication in the Montgomery space, and (3) converting the data back from Montgomery space. Before going into

details, we will first describe Algorithm 4.2. This algorithm implements efficient $A * B * R^{-1} \mod N$ in a hardware-friendly way. The key optimization of the algorithm is the introduction of the divider $R = r^d$. Thus the division can be easily implemented by bit-shifting since r is a power-of-2 integer, and after the division, the result is an integer within $[0, 2N)$ and no more modulo operation is needed. Afterward, we will show details of each step in the following sections.

Converting the data into Montgomery space: Before applying Montgomery Modular Multiplication, the input numbers should be converted to Montgomery space. The conversion formula is $A = a \cdot R \mod N$. It can also be written as $A = a \cdot R^2 \cdot R^{-1} \mod N$, so we can leverage Algorithm 4.2 to efficiently calculate it. In the formula, a is one of the multiplicands of modular multiplication. A is the Montgomery space of a. N is the modulus. R is a power of 2 and it satisfies the condition that $\lfloor \log_2 R \rfloor = \lfloor \log_2 N \rfloor$.

Computing the modular multiplication in the Montgomery space: We can directly use Algorithm 4.2 to efficiently calculate the modular multiplication.

Converting the data back from Montgomery space: To convert a number out of Montgomery space, the conversion formula is $p = P \cdot R^{-1} \mod N$. p is the result of modular multiplication. P is the Montgomery format of p. Similarly, we can leverage Algorithm 4.2 to complete the computation.

Parameter Computation: In the above steps, we notice that if N, which is usually the public key in cryptosystems, remains unchanged, $R^2 \mod N$ (*e.g.*, used in converting the data into Montgomery space) and $(-N)^{-1} \mod r$ (*e.g.*, line 3 and 8 in Algorithm 4.2) remain constant values. We call these constant values parameters in this paper, and we show that we can compute these parameters in advance and avoid duplicate calculations to improve the performance further.

Summary: By applying the Montgomery Modular Multiplication, we mainly replace the modulo operation with a hardware-friendly bit-shifting operation, which improves the performance of modular multiplication/exponentiation on hardware.

4.3 Algorithm Optimization

Algorithm 1 in the main text shows the optimized version of the Montgomery Modular Multiplication algorithm. In this section, we mainly discuss the equivalence of Algorithm 1 in the main text and Algorithm 4.2 in the previous section.

Please first note that, we set $r = 2^w$ and w is the bit-width of Y_i and X_j. Therefore, the "$\mod r$" modulus operation is equivalent to truncating the lower w bits of a $2w$-bit number while the "$>> w$" shifting operation equates to truncating the higher w bits of a $2w$-bit number.

The third line in Algorithm 4.2 is consequently separated into the operations from lines 3 to 5 in Algorithm 1. And it is easy to observe that the value $X_0 \times Y_i$ in the calculation of q is also required in the first iteration of the inner loop. Therefore, we cache the number as α and reuse it in line 6 in Algorithm 1. Therefore, from line 6 to line 9 in Algorithm 1, we equivalently execute the first iteration of the inner loop in Algorithm 4.2.

For the inner iteration, we also find out that, for the value C in line 6 of Algorithm 4.2, its lower w bits are saved as Z_{j-1}, which is the multiplication result for the current iteration, while its higher w bits are saved as C, which is the value carried for the next iteration. Therefore, we separate the calculation of C into higher w and lower w bits from line 11 to line 15 in Algorithm 4.2.

5 Discussion

Benefit to Future GPU/TPU Design: Nowadays, GPU [16], [17], [18], [19], [20] and TPU [21] have been widely adopted to accelerate deep learning applications. These accelerators mainly target accelerating convolution operations with tensors, where most numbers are short floats. In contrast, FLASH targets accelerating the identified nine cryptographic operations that are widely adopted in cross-silo FL. Moreover, most numbers used in FLASH are large numbers with a bit-width of 2048 bit or even longer. However, in some cross-silo applications, *e.g.*, horizontal deep learning, both convolution and cryptographic operations exist. Therefore, we can foresee a co-design of GPU/TPU with FLASH in the future. We will make FLASH as an IP core in the future, and thus GPU/TPU vendors can use FLASH in their design to accomplish the aforementioned co-design.

FLASH v.s. Other GPU/FPGA Implementations: Some existing works also target accelerating modular exponentiation operations with GPU [22], [23], [24] or FPGA [25], [26], [27], [28], [29], [30], [31], which leverage similar algorithm optimization methods, *e.g.*, Binary Exponentiation [32] and Montgomery Modular Multiplication [15]. Yet, none of them performs a thorough analysis towards *all* cryptographic operations used in cross-silo FL and offloads them efficiently on the hardware-based accelerator as FLASH. Moreover, our idea of composing various cryptographic operations based on the two basic operators via dataflow scheduling is designed for the cross-silo FL scenarios, making FLASH a unique solution compared to prior FPGA-based implementations. As a final note, our design of FLASH is not limited to FPGA but is also applied to ASICs.

Extending to Other Application Domains: While FLASH is introduced for accelerating cross-silo FL, it can speed up applications in other domains as well. First, the Paillier and RSA cryptosystems used in cross-silo FL are also widely adopted in other domains. Thus FLASH can accelerate applications built on them, *e.g.*, electronic voting [33], electronic cash [34], and threshold cryptosystem [35]. Second, since FLASH's core idea is to accelerate modular multiplication and exponentiation operators, cryptographic systems/operations built on them, such as Diffie-Hellman key exchange [36], can also benefit from FLASH.

References

[1] 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. 12:1–12:19, 2019.

[2] O. Fink, T. H. Netland, and S. Feuerriegel, "Artificial intelligence across company borders," *Commun. ACM*, vol. 65, no. 1, pp. 34–36, 2022.

[3] Y. Cheng, Y. Liu, T. Chen, and Q. Yang, "Federated learning for privacy-preserving AI," *Commun. ACM*, vol. 63, no. 12, pp. 33–36, 2020.

[4] D. Chai, L. Wang, J. Zhang, L. Yang, S. Cai, K. Chen, and Q. Yang, "Practical lossless federated singular vector decomposition over billion-scale data," in *KDD '22: The 28th ACM SIGKDD Conference*

on Knowledge Discovery and Data Mining, Washington, DC, USA, August 14 - 18, 2022, 2022.

[5] D. Chai, L. Wang, K. Chen, and Q. Yang, "Secure federated matrix factorization," *IEEE Intell. Syst.*, vol. 36, no. 5, pp. 11–20, 2021.

[6] L. Yang, B. Tan, V. W. Zheng, K. Chen, and Q. Yang, "Federated recommendation systems," in *Federated Learning - Privacy and Incentive*, ser. Lecture Notes in Computer Science, 2020, vol. 12500.

[7] Y. Liu, Y. Kang, C. Xing, T. Chen, and Q. Yang, "A secure federated transfer learning framework," *IEEE Intell. Syst.*, vol. 35, no. 4, pp. 70–82, 2020.

[8] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Trans. Inf. Forensics Secur.*, vol. 13, no. 5, pp. 1333–1345, 2018.

[9] K. Cheng, T. Fan, Y. Jin, Y. Liu, T. Chen, D. Papadopoulos, and Q. Yang, "SecureBoost: A lossless federated learning framework," *IEEE Intell. Syst.*, vol. 36, no. 6, pp. 87–98, 2021.

[10] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, ser. Lecture Notes in Computer Science, vol. 1592. Springer, 1999, pp. 223–238.

[11] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[12] G. Liang and S. S. Chawathe, "Privacy-preserving inter-database operations," in *Intelligence and Security Informatics, Second Symposium on Intelligence and Security Informatics, ISI 2004, Tucson, AZ, USA, June 10-11, 2004, Proceedings*, ser. Lecture Notes in Computer Science, vol. 3073. Springer, 2004, pp. 66–82.

[13] E. D. Cristofaro and G. Tsudik, "Practical private set intersection protocols with linear complexity," in *Financial Cryptography and Data Security, 14th International Conference, FC 2010, Tenerife, Canary Islands, Spain, January 25-28, 2010, Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 6052. Springer, 2010, pp. 143–159.

[14] P. Barrett, "Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor," in *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, ser. Lecture Notes in Computer Science, vol. 263. Springer, 1986, pp. 311–323.

[15] Ç. K. Koç, T. Acar, and B. S. K. Jr., "Analyzing and comparing montgomery multiplication algorithms," *IEEE Micro*, vol. 16, no. 3, pp. 26–33, 1996.

[16] "NVIDIA A100," https://www.nvidia.com/en-us/data-center/a100/, 2020.

[17] "NVIDIA P40 Datasheet," https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/documents/nvidia-p40-datasheet.pdf, 2016.

[18] "NVIDIA P4," https://images.nvidia.com/content/pdf/tesla/184457-Tesla-P4-Datasheet-NV-Final-Letter-Web.pdf, 2016.

[19] "NVIDIA H100," https://www.nvidia.com/en-us/data-center/h100/, 2022.

[20] "NVIDIA V100 Datasheet," https://images.nvidia.com/content/technologies/volta/pdf/volta-v100-datasheet-update-us-1165301-r5.pdf, 2017.

[21] N. P. Jouppi, C. Young, N. Patil, D. A. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, and et al., "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA 2017, Toronto, ON, Canada, June 24-28, 2017*, 2017.

[22] K. Jang, S. Han, S. Han, S. B. Moon, and K. Park, "Sslshader: Cheap SSL acceleration with commodity processors," in *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2011, Boston, MA, USA, March 30 - April 1, 2011*, 2011.

[23] X. Cheng, W. Lu, X. Huang, S. Hu, and K. Chen, "HAFLO: gpu-based acceleration for federated logistic regression," *CoRR*, vol. abs/2107.13797, 2021.

[24] O. Harrison and J. Waldron, "Efficient acceleration of asymmetric cryptography on graphics hardware," in *Progress in Cryptology - AFRICACRYPT 2009, Second International Conference on Cryptology in Africa, Gammarth, Tunisia, June 21-25, 2009. Proceedings*, ser. Lecture Notes in Computer Science, vol. 5580, 2009.

[25] D. Suzuki, "How to maximize the potential of FPGA resources for modular exponentiation," in *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, ser. Lecture Notes in Computer Science, vol. 4727. Springer, 2007, pp. 272–288.

[26] T. Blum, "Montgomery modular exponentiation on reconfigurable hardware," in *14th IEEE Symposium on Computer Arithmetic (Arith-14 '99), 14-16 April 1999, Adelaide, Australia*, 1999.

[27] T. Blum and C. Paar, "High-radix montgomery modular exponentiation on reconfigurable hardware," *IEEE Trans. Computers*, vol. 50, no. 7, pp. 759–764, 2001.

[28] T. Alho, P. Hämäläinen, M. Hännikäinen, and T. D. Hämäläinen, "Compact modular exponentiation accelerator for modern FPGA devices," *Comput. Electr. Eng.*, vol. 33, no. 5-6, pp. 383–391, 2007.

[29] I. San, N. At, I. Yakut, and H. Polat, "Efficient paillier cryptoprocessor for privacy-preserving data mining," *Secur. Commun. Networks*, vol. 9, no. 11, pp. 1535–1546, 2016.

[30] Z. Yang, S. Hu, and K. Chen, "Fpga-based hardware accelerator of homomorphic encryption for efficient federated learning," *CoRR*, vol. abs/2007.10560, 2020.

[31] M. Bahadori and K. Järvinen, "A programmable soc-based accelerator for privacy-enhancing technologies and functional encryption," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 28, no. 10, pp. 2182–2195, 2020.

[32] D. M. Gordon, "A survey of fast exponentiation methods," *J. Algorithms*, vol. 27, no. 1, pp. 129–146, 1998.

[33] I. Damgård, M. Jurik, and J. B. Nielsen, "A generalization of paillier's public-key system with applications to electronic voting," *Int. J. Inf. Sec.*, vol. 9, no. 6, pp. 371–385, 2010.

[34] J. Camenisch, A. Lysyanskaya, and M. Meyerovich, "Endorsed e-cash," in *2007 IEEE Symposium on Security and Privacy (S&P 2007), 20-23 May 2007, Oakland, California, USA*, 2007.

[35] I. Damgård and M. Jurik, "A length-flexible threshold cryptosystem with applications," in *Information Security and Privacy, 8th Australasian Conference, ACISP 2003, Wollongong, Australia, July 9-11, 2003, Proceedings*, ser. Lecture Notes in Computer Science, vol. 2727, 2003.

[36] M. E. Hellman, B. W. Diffie, and R. C. Merkle, "Cryptographic apparatus and method," Apr. 29 1980, uS Patent 4,200,770.