Explore Parallel Processing for Efficient Encryption

Jingyang Ou

School of Information, Renmin University of China
2021201746@ruc.edu.cn

***Abstract*—Homomorphic encryption algorithms such as Paillier [1] and Okamoto-Uchiyama[2] (OU) are critical for privacy-preserving computations. However, their computational performance has often been a bottleneck for large-scale application. This paper presents an acceleration of the Paillier and OU homomorphic encryption algorithms using OpenMP on CPU and CUDA on GPU. We compare the performance of these accelerated versions with the original CPU implementation across different data scales, and it shows that the GPU-accelerated versions consistently outperform the original CPU and OpenMP-optimized CPU implementations, achieving run times orders of magnitude lower.**

I. Introduction

Homomorphic encryption algorithms such as Paillier and Okamoto-Uchiyama (hereafter referred to as OU) have been widely employed in the domain of privacy-preserving computations. Compared to other privacy computation schemas like Federated Learning (FL) and Multi-Party Computation (MPC), the computational performance of homomorphic encryption algorithms is relatively poor, posing as a bottleneck for large-scale application of privacy computations.

Efforts have been made to accelerate the Paillier encryption using hardware like Field Programmable Gate Arrays (FPGA), achieving moderate success. However, these solutions are intrinsically tied to specific hardware models. The drawback with such an approach is that the algorithm becomes inapplicable once the FPGA model changes.

Graphics Processing Units (GPUs), on the other hand, are a type of general-purpose hardware that is widely available and can potentially benefit a larger user base. The acceleration of homomorphic encryption algorithms based on GPUs can lead to more extensive benefits.

In this paper, we present an acceleration of the Paillier and OU homomorphic encryption algorithms using OpenMP on CPU and CUDA on GPU. We then compare these accelerated versions with the original CPU implementation across different data scales, including the encryption and decryption processes. This comparison allows us to evaluate the effectiveness of our acceleration methods and their potential for wider application in the realm of privacy computations.

II. Background of the Paillier and Okamoto-Uchiyama Encryption Algorithms

In both the Paillier and Okamoto-Uchiyama (OU) encryption schemes, the key generation phase involves the creation of public and private keys. For Paillier, the public key is (n, g) and the private key is (λ, μ). For OU, the public key is (n, g, h) and the private key is (p, q). The key generation process, which is performed once and doesn't need parallel optimization, while the encryption and decryption processes are performed for each message and thus are the focus of our acceleration efforts.

In the encryption process, a plaintext message m is encrypted using a randomly chosen number r and a public key (n, g for Paillier and n, g, h for OU) to produce the ciphertext c. while the decryption process involves using the private key (λ, μ for Paillier and p, q for OU) to decode the ciphertext c into plaintext message m. Due to space constraints, we only give the pseudo-code of the OU algorithm as follows:

Algorithm 1 Encryption for Okamoto-Uchiyama Algorithm

Input: Plaintext m, Public Key (n, g, h)
Output: Ciphertext c

1: generate a random integer r where $1 \leq r < n$
2: $c \leftarrow g^m \cdot h^r \bmod n$
3: **return** c

Algorithm 2 Decryption for Okamoto-Uchiyama Algorithm

Input: Ciphertext c, Private Key (p, q)
Output: Plaintext m

1: $a \leftarrow \frac{(c^{p-1} \bmod p^2)-1}{p}$
2: $b \leftarrow \frac{(g^{p-1} \bmod p^2)-1}{p}$
3: $b' \leftarrow b^{-1} \bmod p$
4: $m \leftarrow a \cdot b' \bmod p$
5: **return** m

III. Implementation

We implementated the Paillier and Okamoto-Uchiyama encryption and decryption processes using CUDA and OpenMP respectively. In addition, the correctness of the code is verified through tests, which confirmed the integrity of encrypted and decrypted content and validated the homomorphic properties. The optimized code and tests can be accessed at https://github.com/Dawn-dreamer/parallel-encryption.

IV. Evaluation

Our evaluation of the performance of the Paillier and Okamoto-Uchiyama (OU) encryption and decryption processes, implemented using CUDA and OpenMP, is based on

various data scales. These experiments were conducted on a server equipped with an Intel Xeon Gold 5218 v64CPU and an NVIDIA GeForce RTX 2080 Ti GPU.

Tables 1 and 2 show the run times for the OU algorithm's encryption and decryption processes, respectively. The tables compare the performance of three different implementations: GPU, naive CPU, and OpenMP-optimized CPU, across several message sizes.

As seen in Table 1, the GPU implementation consistently outperforms the CPU implementations, with run times orders of magnitude lower. For example, for a message size of 51200000, the GPU implementation is about 199 times faster than the naive CPU implementation and 14 times faster than the OpenMP-optimized CPU implementation. As the running time grows linearly with the message size for all three implementations, the GPU implementation is expected to maintain its advantage over the CPU implementations when message sizes are large enough to parallelize.

Table 2 shows similar trends for the decryption process. The GPU implementation is consistently faster, however, the performance gap is not as large as in the encryption process, with only a 2-3 times difference between the GPU and CPU-OpenMP implementations. This is likely because the decryption process is computationally less intensive than the encryption process, with a relatively low computation-to-memory access ratio, and there is further room for optimization.

Figures 1 and 2 visualize the running times of the OU and Paillier algorithms respectively, further emphasizing the superior performance of the GPU implementation.

TABLE I
OU ALGORITHM RUN TIME (ENCRYPTION)

m_size	gpu	naive	cpu_openmp
100000	0.000532s	0.062155s	0.009082s
200000	0.000795s	0.123952s	0.015701s
400000	0.001419s	0.247914s	0.025393s
800000	0.002785s	0.496227s	0.044426s
1600000	0.005565s	0.992035s	0.081668s
3200000	0.010761s	1.985010s	0.152032s
6400000	0.019694s	3.974340s	0.299547s
12800000	0.039633s	7.964630s	0.595382s
25600000	0.078314s	15.882800s	1.207300s
51200000	0.159827s	31.751900s	2.295440s

TABLE II
OU ALGORITHM RUN TIME (DECRYPTION)

m_size	gpu	naive	cpu_openmp
100000	0.000393s	0.049923s	0.005106s
200000	0.000602s	0.099734s	0.005831s
400000	0.001102s	0.199807s	0.006019s
800000	0.002131s	0.399257s	0.012765s
1600000	0.004282s	0.796984s	0.015954s
3200000	0.008138s	1.595980s	0.026121s
6400000	0.016131s	3.194750s	0.042396s
12800000	0.031406s	6.390020s	0.078432s
25600000	0.066514s	12.765700s	0.146685s
51200000	0.133470s	25.553600s	0.29466s

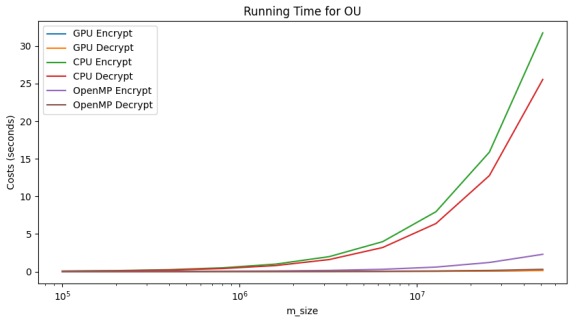

Fig. 1. Running time of OU algorithm

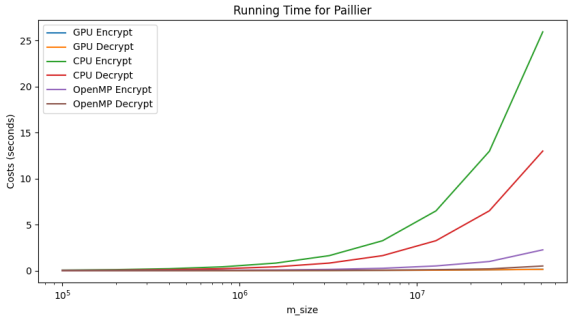

Fig. 2. Running time of Paillier algorithm

V. CONCLUSION

In this paper, we introduced an optimized implementation of the Paillier and Okamoto-Uchiyama homomorphic encryption algorithms using OpenMP on CPU and CUDA on GPU. Our implementation addresses the computational performance issues that have previously limited the wider application of these algorithms in privacy computations.

REFERENCES

[1] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International Conference on the Theory and Application of Cryptographic Techniques*, 1999. [Online]. Available: https://api.semanticscholar.org/CorpusID:9483611

[2] T. Okamoto and S. Uchiyama, "A new public-key cryptosystem as secure as factoring," in *International Conference on the Theory and Application of Cryptographic Techniques*, 1998. [Online]. Available: https://api.semanticscholar.org/CorpusID:43809321