Accelerating CKKS Homomorphic Encryption with Data Compression on GPUs

Quoc Bao Phan, Linh Nguyen, and Tuy Tan Nguyen
School of Informatics, Computing, and Cyber Systems
Northern Arizona University
Flagstaff, AZ 86011, USA
tuy.nguyen@nau.edu

2024 IEEE 67th International Midwest Symposium on Circuits and Systems (MWSCAS) | 979-8-3503-8717-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/MWSCAS60917.2024.10658747

***Abstract*—Homomorphic encryption (HE) algorithms, particularly the Cheon-Kim-Kim-Song (CKKS) scheme, offer significant potential for secure computation on encrypted data, making them valuable for privacy-preserving machine learning. However, high latency in large integer operations in the CKKS algorithm hinders the processing of large datasets and complex computations. This paper proposes a novel strategy that combines lossless data compression techniques with the parallel processing power of graphics processing units to address these challenges. Our approach demonstrably reduces data size by 90% and achieves significant speedups of up to 100 times compared to conventional approaches. This method ensures data confidentiality while mitigating performance bottlenecks in CKKS-based computations, paving the way for more efficient and scalable HE applications.**

***Index Terms*—Homomorphic encryption, graphics processing units, data compression, CKKS, privacy-preserving.**

I. Introduction

The ever-growing volume of personal, organizational, and transactional data in the global intelligent technology landscape necessitates efficient storage and robust security measures [1]. This data surge, fueled by e-commerce, online transactions, and the Internet of Things (IoT), intensifies cybersecurity threats, even for seemingly innocuous data like personal details [2]–[4]. As a response, advancements in cryptographic techniques, specifically homomorphic encryption (HE), are gaining traction. HE allows computations on encrypted data, preserving confidentiality. Cheon-Kim-Kim-Song (CKKS) algorithm [5] exemplifies this innovation, addressing key HE challenges by enabling computations on real numbers and performing arithmetic operations on encrypted data.

Existing approaches for the CKKS algorithm are often hampered by large datasets and CPU-bound processing delays [6], [7]. This paper proposes a novel method to improve the efficiency of the CKKS algorithm by combining advanced data compression and acceleration techniques. Our contributions include: (1) employing two distinct data compression algorithms to process data in CKKS algorithm: dictionary-based compression (DBC) and discrete cosine transform (DCT), tailored for text and images, with efficacy analysis; (2) leveraging the parallel processing power of graphics processing units (GPUs) with compute unified device architecture (CUDA) cores to accelerate the number theoretic transform (NTT)-based polynomial multiplication algorithm and the entire system; and (3) conducting a comparative performance analysis between our system and Microsoft simple encrypted arithmetic library (SEAL) [8].

The remaining sections of the paper are structured as follows: In Section II, we introduce the background of CKKS and data compression algorithms. Section III introduces the acceleration techniques. Section IV simulates and compares the performance of the proposed approach with Microsoft SEAL. The paper is concluded in Section V.

II. Background

A. CKKS Algorithm

The CKKS scheme initializes with a security parameter λ. The ring dimension N is determined, being a power of two. Small distributions (χ_{key}, χ_{err}, and χ_{enc}) over the ring $\mathbb{R}$ are established for secret key, error, and encryption, respectively. Key generation produces both public and secret keys. The secret key (sk) is a random polynomial s sampled from χ_{key}, while the public key (pk) comprises a random polynomial a from $\mathbb{R}ql$ (for given ciphertext modulus level l) and an error polynomial e from χ_{err}. The public key pk is shared with clients for encryption. During encryption, plaintext m is scaled and encoded into polynomial $\hat{m}$ using the Encode function with scaling factor γ. Random polynomials v are sampled from χ_{enc}, and error polynomials e_0 and e_1 from χ_{err}. The ciphertext (ct) is computed as $ct = v \cdot pk + (\hat{m} + e_0, e_1) \mod ql$, where ql denotes ciphertext modulus for level l. For decryption, ciphertext ct decrypts as $\hat{m} = c_0 + c_1 \cdot s \mod ql$, where $ct = (c_0, c_1)$. Homomorphic operations are supported, including adding and multiplying encrypted ciphertexts, relinearization for noise reduction, and rotation for operations on different ciphertext parts.

B. Data Compression Techniques

The DBC algorithm [9]–[11], a widely used technique for lossless data compression, achieves compression by identifying and replacing recurring sequences of symbols with shorter codes. Initially, it builds a dictionary containing all unique symbols from the input data. As processing progresses, the DBC algorithm dynamically expands this dictionary to include frequently encountered longer substrings. During the encoding phase, these identified substrings are replaced with their corresponding codes from the dictionary, resulting in compressed data.

Fig. 1. Diagram illustrating accelerated compression techniques on GPU for (a) series data and (b) image data.

Furthermore, DCT serves as another crucial tool for signal and image processing, as detailed in [12]. It operates by transforming data from the spatial domain, where it represents positions in space, to the frequency domain, which reveals the distribution of energy across different frequencies. This transformation is achieved by decomposing the signal or image into smaller blocks, applying the DCT to each block individually, and then quantizing the resulting coefficients. Notably, quantization reduces the precision of these coefficients, enabling efficient compression while preserving an acceptable level of visual fidelity.

III. Accelerating CKKS Homomorphic Encryption

This work investigates various methods for compressing CKKS input data, along with their corresponding GPU implementations, to accelerate computations within the CKKS homomorphic encryption algorithm. These compression techniques are also integrated with NTT polynomial multiplication to further improve performance for CKKS-based applications.

A. Accelerating CKKS Algorithm with Data Compression

We introduce CKKS-based security for both series and image data. However, compressing image data presents unique challenges due to the inherent complexity of large datasets, making it computationally expensive to achieve effective compression [13]. To address these compression challenges, we propose a two-part approach. First, we employ DBC for efficient compression of series data. Second, we leverage DCT for effective image compression, utilizing dimension expansion to handle the inherent complexity of image data. Both techniques are optimized for CUDA cores to maximize performance. The detailed structures of these techniques are illustrated in Fig. 1.

1) Series Data: The DBC compression algorithm leverages the parallel processing capabilities of GPUs by distributing the workload across numerous cores for concurrent execution. This approach significantly reduces processing times. To achieve this parallelism, each CUDA block is configured with n threads. Each thread handles a specific portion of the input data, enabling simultaneous processing. For instance, if the input data size is 1024 bits, it is initially divided into n chunks, with each chunk containing $1024/n$ bits. Each CUDA thread concurrently executes dictionary operations, searching for sequences and adding new sequences. This distribution of tasks among threads facilitates efficient management of the

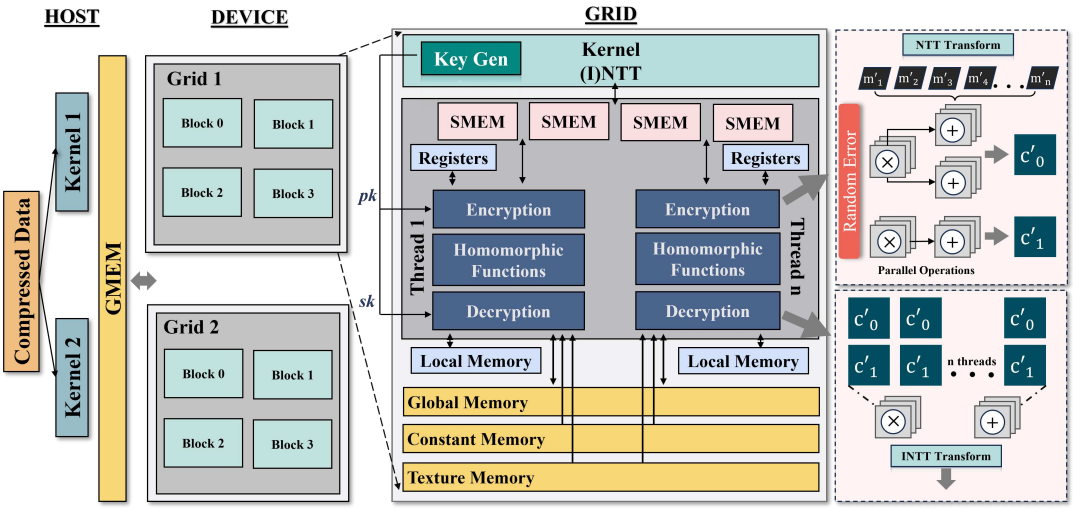


Fig. 2. GPU settings and communications.

dictionary, hash tables, and dictionary partitioning. Specifically, each thread is responsible for searching the dictionary to identify the longest prefix of the current sequence found in the dictionary: $\mathsf{LongestPrefix}(C) = \max\{j - i \mid C[i:j] \in D\}$. Upon identification, the thread outputs the corresponding code. It contributes to adding the new sequence to the dictionary with a new code, represented as $D[S] = \mathsf{NextCode}$ and $\mathsf{NextCode} = \mathsf{NextCode} + 1$. This parallel process continues iteratively until the entire input sequence is processed. As a result, the algorithm generates a compressed output with variable-length codes representing the input data.

2) Image Data: The image compression process is optimized through the use of the CUDA parallelism paradigm, which ensures efficient execution. The image is first segmented into N 8×8 blocks, denoted by $f(x, y)$. To distribute the workload across multiple threads, we assign n threads, with each thread handling N/n blocks. These threads then concurrently compute the DCT $F(u, v)$ for their assigned blocks using (1).

$$F(u,v) = \sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y)\cos\left[\frac{(2x+1)u\pi}{2\cdot 8}\right] \cos\left[\frac{(2y+1)v\pi}{2\cdot 8}\right] \tag{1}$$

Once the DCT is computed, parallelism is also utilized in subsequent quantization, where each thread independently quantizes its block subset. Quantization involves dividing DCT coefficients by a pre-defined quantization matrix $Q(u, v)$ and rounding for compression efficiency. Threads then perform inverse quantization in parallel. During decompression, the inverse DCT is used to reconstruct the block from frequency data by (2). This process is executed by threads concurrently.

$$f(x,y) = \frac{1}{4\cdot 8^2}\sum_{u=0}^{7}\sum_{v=0}^{7} C(u)C(v)F(u,v) \cos\left[\frac{(2x+1)u\pi}{2\cdot 8}\right]\cos\left[\frac{(2y+1)v\pi}{2\cdot 8}\right] \tag{2}$$

Here, $C(u)$ and $C(v)$ are normalization factors. This approach harnesses the parallel processing power of the GPU to significantly accelerate image compression and decompression.

B. Accelerating CKKS Algorithm with NTT Polynomial Multiplication on GPU CUDA Cores

We take advantage of the parallel GPU with CUDA cores to speed up the NTT-based polynomial multiplication [14] in the CKKS scheme, as depicted in Fig. 2. During the encryption phase, each element of a two-dimensional input sequence of size $s = x \cdot y$, where x and y are the dimensions along the first and second axes, respectively, is mapped to a unique index using i_x and i_y. These indices range from 0 to $x-1$ and $y-1$, respectively, and are combined as $i = i_x + i_y \cdot x$, linking the matrix to a linear sequence of size s. We introduce k_x and k_y as transformed sequence indices, with k_x ranging from 0 to $x - 1$ and k_y from 0 to $y - 1$. The transformed sequence is accessed via $k_y + k_x \cdot y$. Each element $f_{k_y+k_x\cdot y}$ in the transformed sequence is computed as in (3).

$$f_{k_y+k_x\cdot y} = \sum_{i_x=0}^{x-1}\sum_{i_y=0}^{y-1} x_{i_x+i_y\cdot x}\cdot \theta^{yi_y\cdot k_y}\cdot \theta^{si_x\cdot k_y}\cdot \theta^{xi_x\cdot k_x} \tag{3}$$

Expanding the exponent of θ^n:

$$\theta^{s(i_x+i_y\cdot x)\cdot(k_y+k_x\cdot y)} = \theta^{yi_y\cdot k_y}\cdot \theta^{si_x\cdot k_y}\cdot \theta^{xi_x\cdot k_x} \tag{4}$$

Utilizing the properties of roots of unity:

$$\theta^{s(i_x+i_y\cdot x)\cdot(k_y+k_x\cdot y)} = \theta^{yi_y\cdot k_y}\cdot \theta^{si_x\cdot k_y}\cdot \theta^{xi_x\cdot k_x} \tag{5}$$

After decryption, each thread applies the INTT to recover the original sequence f from its transformed counterpart $\hat{f}$:

$$x_{i_x+i_y\cdot x} = \frac{1}{s}\sum_{k_x=0}^{x-1}\sum_{k_y=0}^{y-1} \hat{f}_{k_y+k_x\cdot y} \cdot \left(\theta^{-s(i_x+i_y\cdot x)\cdot(k_y+k_x\cdot y)}\right)\cdot\left(\theta^{xi_x\cdot k_x}\right)\cdot k_y \tag{6}$$

The final step involves aggregating the results from all threads to generate the final approximate result and saving it to global memory.

IV. Experimental Evaluation

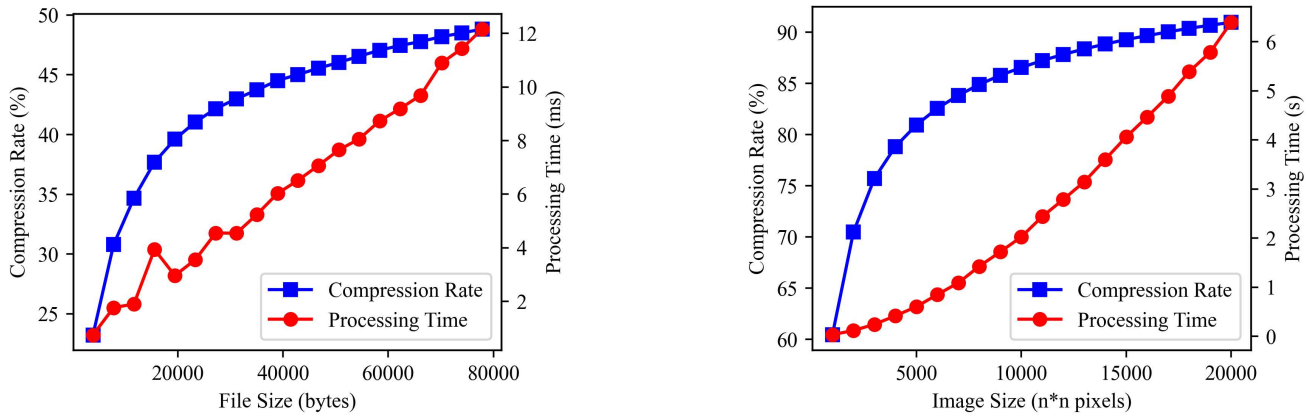
This section evaluates the performance gains achieved by combining data compression and GPU acceleration within the CKKS algorithm. We leverage a computing environment equipped with an Intel Core i9-13900 CPU and an NVIDIA GeForce RTX 3050 GPU for our experiments. These experiments are conducted using CUDA Toolkit 12.3.

A. Impact of Data Compression

Our simulations yielded significant reductions in file size for both general data and images, as demonstrated in Figs. 3a and 3b. By harnessing the power of 128 CUDA threads for parallel processing, our implemented compression algorithm delivered impressive compression ratios ranging from 23% to 48%. This trend was similarly observed in image compression. Notably, larger images displayed a positive correlation with compression ratio, spanning from 60% for smaller images to an outstanding 90% for larger ones. The attained high compression ratios directly contribute to optimized digital asset management and efficient data storage and transmission. However, a trade-off arises between compression ratio and processing time for both file size and image compression. While achieving these remarkable compression ratios, processing time also exhibits a positive correlation with data size, increasing by a factor of up to 12 for larger files.

B. Performance Evaluation on CKKS Operations

Table I presents a comparison of processing efficiency between Microsoft SEAL and our proposed work across various operations. Notably, our work consistently outperforms Microsoft SEAL in terms of encryption and decryption times for different settings when considering randomly generated data. For instance, in the (32768, 740) setting, encryption



(a) (b)

Fig. 3. Compression performance with 128 CUDA threads on (a) series data and (b) image data.

TABLE I
PERFORMANCE EVALUATION

Type of message	Function	Microsoft SEAL (modulus degree, coefficient bits)			This Work (modulus degree, coefficient bits)		
		(8192, 160)	(16384, 280)	(32768, 740)	(8192, 160)	(16384, 280)	(32768, 740)
Random Generated Data	Encode	0.327	0.971	6.092	0.021	0.028	0.374
	Encryption	1.790	5.060	30.135	0.110	0.118	0.453
	Decryption	0.101	0.163	2.401	0.001	0.002	0.018
	Decode	0.504	1.758	15.394	0.039	0.055	0.904
Original Image Data	Encode	10.363	12.038	16.259	0.365	0.535	4.339
	Encryption	2.119	5.285	30.514	0.255	0.510	0.613
	Decryption	0.161	0.288	1.873	0.013	0.015	0.018
	Decode	1.889	2.115	17.709	0.784	1.049	1.319
Compressed Image Data	Encode	5.3661	10.108	12.409	0.311	0.417	3.409
	Encryption	3.137	4.871	25.860	0.127	0.240	0.355
	Decryption	0.061	0.204	1.514	0.011	0.013	0.014
	Decode	1.636	2.032	13.744	0.637	0.973	1.034

time for our work is significantly lower (0.453 ms) compared to Microsoft SEAL (30.135 ms), indicating a substantial reduction in processing overhead. Similarly, decryption time decreases from 2.401 ms to 0.018 ms, further highlighting the superior performance of our work in handling encryption and decryption tasks efficiently. Moreover, the impact of data compression on the efficiency of CKKS operations becomes evident when comparing original and compressed data.

Data compression consistently leads to significant reductions in both encryption and decryption times. In the (32768, 740) setting, for example, encryption time decreases from 0.613 ms to 0.355 ms, and decryption time diminishes from 0.018 ms to 0.014 ms when working with compressed image data instead of uncompressed data. The benefits of data compression extend beyond encryption and decryption, as evidenced by the observed improvements in encoding and decoding tasks, thereby underscoring its positive impact on overall system performance and efficiency.

V. CONCLUSION

Our research highlights the promising integration of the CKKS algorithm with data compression techniques and GPU implementation. Through meticulous analysis of this integration, we achieve impressive efficiency improvements, including a notable compression rate of 90% and processing speeds surpassing the traditional SEAL-based CKKS algorithm by up to 100 times. This convergence not only enables secure and large-scale data computations but also enhances agility and efficacy in handling extensive datasets. By leveraging GPU architecture for accelerated throughput and parallel processing, coupled with the advanced compression techniques, our approach represents a significant stride towards optimizing resource utilization and bolstering system performance across various domains.

ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation under Grant No. 2348464.

REFERENCES

[1] AAG IT, "The latest cyber crime statistics," https://aag-it.com/the-latest-cyber-crime-statistics/, 2024, Accessed: 3/14/2024.

[2] B. Nour, M. Pourzandi, and M. Debbabi, "A survey on threat hunting in enterprise networks," *IEEE Communications Surveys Tutorials*, vol. 25, no. 4, pp. 2299–2324, 2023.

[3] F. Valenza, E. Karafili, R. V. Steiner, and E. C. Lupu, "A hybrid threat model for smart systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 5, pp. 4403–4417, 2023.

[4] N. Tatipatri and S. L. Arun, "A comprehensive review on cyber-attacks in power systems: Impact analysis, detection, and cyber security," *IEEE Access*, vol. 12, pp. 18 147–18 167, 2024.

[5] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *International Conference on the Theory and Application of Cryptology and Information Security*, 2017.

[6] S. Shen, H. Yang, Y. Liu, Z. Liu, and Y. Zhao, "CARM: CUDA-Accelerated RNS multiplication in word-wise homomorphic encryption schemes for Internet of Things," *IEEE Transactions on Computers*, vol. 72, no. 7, pp. 1999–2010, 2023.

[7] T. T. Nguyen, Q. B. Phan, N. T. Bui, and C. daCunha, "High-secure data collection in IoT sensor networks using homomorphic encryption," in *Sensors and Systems for Space Applications XVI*, vol. 12546. SPIE, 2023, pp. 53–60.

[8] Microsoft, "Microsoft SEAL v4.1," https://github.com/microsoft/SEAL, 2023, Accessed: 3/14/2024.

[9] J. Qian, P. Tiwari, S. P. Gochhayat, and H. M. Pandey, "A noble double-dictionary-based ECG compression technique for IoTH," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 10 160–10 170, 2020.

[10] S. B. Raut, "AOCL-compression - A high performance optimized lossless data compression library," in *2023 IEEE High Performance Extreme Computing Conference (HPEC)*, 2023, pp. 1–7.

[11] M. Sahu and J. Panda, "A time efficient approach to data compression for LZW algorithm," in *2023 Annual International Conference on Emerging Research Areas: International Conference on Intelligent Systems (AICERA/ICIS)*, 2023, pp. 1–3.

[12] M. Wang and X. Shang, "A fast image fusion with discrete cosine transform," *IEEE Signal Processing Letters*, vol. 27, pp. 990–994, 2020.

[13] S. Mittal and J. S. Vetter, "A survey of architectural approaches for data compression in cache and main memory systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 5, pp. 1524–1536, 2016.

[14] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Mathematics of computation*, vol. 19, no. 90, pp. 297–301, 1965.