A Survey on FPGA-based Accelerators for CKKS

Wenpeng Zhao[1], Qidong Chen[1], Yijie Wang[1], Haichun Zhang*[1], Zhaojun Lu[1], and Gang Qu[2]
[1]School of Cyber Science and Engineering, Huazhong University of Science and Technology, China.
[2]University of Maryland, College Park, MD, USA.
*Corresponding author: Haichun Zhang, zhanghaichun@hust.edu.cn

2024 IEEE International Test Conference in Asia (ITC-Asia) | 979-8-3315-4033-3/24/$31.00 ©2024 IEEE | DOI: 10.1109/ITC-Asia62534.2024.10661342

***Abstract*—Cheon-Kim-Kim-Song (CKKS) is a Fully Homomorphic Encryption (FHE) scheme that enables computations directly on encrypted real or complex numbers, ensuring the privacy of sensitive information even in untrusted environments. However, the processing of encrypted data incurs significant computational overhead compared to plaintext computations, making CKKS impractical for wider adoption. Field Programmable Gate Arrays (FPGAs) are a promising platform to accelerate CKKS because of parallelism, scalability, flexibility, and widespread availability across cloud providers. This paper systematically surveys the key techniques and current advancements in FPGA-based accelerators for CKKS and discusses future research trends to facilitate real-world homomorphic applications.**

***Index Terms*—Cheon-Kim-Kim-Song, Bootstrapping, Field Programmable Gate Array, Residue Number System.**

I. Introduction

In 2017, Jung Hee Cheon et al. [1] proposed Cheon-Kim-Kim-Song (CKKS), a technique of Fully Homomorphic Encryption (FHE) that enables direct execution of arithmetic operations on ciphertexts without requiring access to the decryption key. CKKS is particularly designed to compute encrypted data that represents approximate numbers, offering a practical and scalable solution for preserving privacy in secure data analysis [2] and cloud computing [3].

Despite the advancements in secure computation, degraded efficiency remains the primary obstacle hindering the adoption of the CKKS scheme. For simple operations, such as multiplication in CKKS, the performance might be thousands of times slower [4]. Bootstrapping [5] is essential to support more homomorphic evaluations, which involves multiple sequential operations that significantly increase the computational and storage overhead. Microsoft SEAL library [6] and Lattigo [7] utilize multi-threading and efficient CPU features to speed up computations for CKKS. By leveraging specialized hardware, including Graphics Processing Unit (GPU) [8], [9], Field Programmable Gate Array (FPGA) [4], [10]–[13] and potentially Application-Specific Integrated Circuit (ASIC) [14], [15], the performance of CKKS can be further improved. FPGA-based accelerators are extensively studied Because of their intrinsic properties, such as parallel processing abilities, adaptable reconfigurability, and widespread availability through cloud service providers [12].

However, several challenges must be addressed to harness FPGA capabilities for CKKS acceleration fully. First, optimizing resource utilization poses a critical challenge as operations on large polynomials and intricate arithmetic rapidly deplete the limited logic blocks, memory, and DSP slices within an FPGA. Second, efficiently implementing the intricate functions onto hardware demands profound knowledge of cryptography and substantial effort in FPGA programming. Third, exploiting the parallelism and reconfigurability of FPGA to accommodate various security requirements or optimization strategies represents a considerable challenge due to the complexity inherent in CKKS algorithms.

This survey can serve as an introductory reference in the field of FPGA-based accelerators for CKKS, while also exploring potential future research directions within this domain. Initially, we explain the core principles of the CKKS scheme through an engineering perspective, focusing on detailing the computational flow of homomorphic evaluation, and bootstrapping algorithm. Then, we conduct a comprehensive performance analysis to identify the performance bottlenecks in the CKKS scheme and provide explanations of these fundamental operations. Finally, we present an in-depth investigation into accelerating basic operators on FPGA platforms and the tailored hardware/software co-design architectures optimized for the CKKS scheme.

II. CKKS Scheme

Fully Homomorphic Encryption (FHE) protocols typically involve integer- and lattice-based schemes. The most efficient lattice-based schemes rely on the Ring Learning with Errors (RLWE) problem [16], which offers strong security guarantees and acceptable performance. The CKKS homomorphic encryption scheme is also designed based on the RLWE problem. Fig. 1 illustrates the application scenarios where the client-side data is first encrypted using a public key and then sent to the server for computation in ciphertext form. This computation requires an evaluation key (evk) from the server, which is a pivotal element for processing encrypted data. Once the computation is completed, the results are sent back to the client

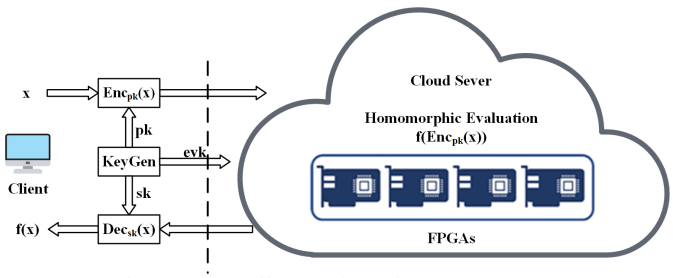


Fig. 1: Overall FHE-based cryptosystem.

TABLE I: Description of Notations.

Notation	Description
N	The degree of a polynomial
$\mathbb{Z}[X]/(X^N+1)$	Polynomial ring over the integers modulo X^N+1
$\mathcal{R}_Q$	Residue ring modulo an integer Q
χ_{key}	Distribution of the secret key
χ_{err}	Error distribution over $\mathcal{R}$
Q	Special (prime) moduli product= $\prod_{i=0}^{L} q_i$
$q_0, \ldots, q_L$	(prime) moduli
P	Special (prime) moduli product= $\prod_{i=0}^{k-1} p_i$
$p_0, \ldots, p_{k-1}$	Specil (prime) moduli
evk	Evaluation key
$dnum$	Decomposition number
$L(\ell)$	Maximum (current) level of a ciphertext
$[\cdot]_q$	Modular reduction by q
$\odot$	Hadamard product

to be decrypted using a private key to retrieve the computed outcomes. Table I clarifies the notations and the descriptions used in subsequent sections.

A. Polynomial Arithmetic

In the CKKS scheme, each ciphertext is represented as a pair of polynomials of degree less than N within the ring $\mathcal{R}_Q = \mathbb{Z}_Q[X]/(X^N+1)$. Operations in FHE consist of polynomial arithmetics. Given that the polynomial degree N is exceedingly high and the coefficient sizes are substantial, such polynomial arithmetic is notably costly within the FHE context. Compared to native multiplication in the unencrypted domain, the latency of FHE multiplication increases by several thousand times.

Polynomial ring multiplication can be efficiently implemented using the Number Theoretic Transform (NTT) and its inverse (iNTT). NTT is a variant of the Discrete Fourier Transform (DFT) that operates over a finite field of integers. Let ω_{q_i} be a primitive N-th root of unity in R_{q_i}. For a polynomial $[a(X)]_{q_i} \in \mathcal{R}_{q_i}$, the NTT function NTT$([a(X)]_{q_i})$ returns $([a(\omega_{q_i}^0)]_{q_i}, \cdots, [a(\omega_{q_i}^{N-1})]_{q_i}) \in (\mathbb{Z}_{q_i}^*)^N$. The inverse NTT function, iNTT, takes the output from NTT and returns the polynomial $[a(X)]_{q_i}$. This approach to polynomial multiplication streamlines computations by transforming polynomials into a domain where pointwise multiplication is feasible, thereby significantly reducing the computational complexity associated with ring operations.

Due to the substantial computational cost associated with polynomial multiplication, FHE schemes based on RLWE typically incorporate variants of the Residue Number System (RNS) to reduce their computational overhead. Initially, by leveraging the Chinese Remainder Theorem (CRT), each large integer coefficient is represented as a set of residues, with each residue being the coefficient modulo a different prime number. The multiplication between two large integers is thus transformed into pointwise multiplication between two sets of residues, thereby avoiding the costly multi-word arithmetic.

For polynomials with a RNS basis $\mathcal{C}_i = \{q_0, \ldots, q_i\}$, the NTT and iNTT are applied for each modulus. Given $[a(X)]_{\mathcal{C}_i}$, $[a]_{\mathcal{C}_i} = \text{NTT}([a(X)]_{\mathcal{C}_i}) = (a^{(0)}, a^{(1)}, \cdots, a^{(i)})$, where $a^{(j)} = \text{NTT}([a(X)]_{q_j})$. Consequently, $\text{iNTT}([a]_{\mathcal{C}_i}) = [a(X)]_{\mathcal{C}_i}$. Therefore, $[a(X)]_{\mathcal{C}_i} \cdot [b(X)]_{\mathcal{C}_i}$ is equal to $\text{iNTT}([a]_{\mathcal{C}_i} \odot [b]_{\mathcal{C}_i})$. For simplicity, subscripts are omitted (e.g., $a \odot b$).

B. Basic Functions

We briefly describe the CKKS scheme and its basic functions. CKKS encodes a message $\vec{z}$, consisting of $N/2$ complex numbers, into a polynomial $m(X) \in \mathcal{R}_Q$, referred to as the plaintext. Each position of the complex values in the message is known as a slot. The encoding process involves certain preprocessing of $\vec{z}$, followed by an Inverse Discrete Fourier Transform (iDFT), then amplification by a scaling factor Δ (referred to as scaling), and finally, a rounding operation. A larger Δ improves the precision of the message but requires larger prime numbers $q_i \in \mathcal{C}_L$, which, under the same conditions, may reduce the security level. Typical values of Δ range from 2^{30} to 2^{50}. The decoding process is the inverse of the encoding process.

CKKS employs an RLWE instance for key generation and encryption. For example, let a secret key be $s(X) \leftarrow \chi_{\text{key}}$. To encrypt a plaintext polynomial $m(X) \in \mathcal{R}_Q$, CKKS first generates a pair $(a(X), b(X))$ within $\mathcal{R}_Q$, where $a(X) \leftarrow \mathcal{R}_Q$ and $b(X) \leftarrow a(X) \cdot s(X) + e(X)$, with $e(X) \leftarrow \chi_{\text{err}}$. Following the creation of this pair, encryption is completed by adding $m(X)$ to $b(X)$. Here, $e(X) \in \mathcal{R}_Q$ and the bit length of the coefficients of $e(X)$ is very short. The secret key $s(X) \in \mathcal{R}_Q$ is essential for recovery of $m(X) + e(X)$ through the operation $(a(X), b(X)) \cdot (-s(X), 1)$.

The HADD operation (Algorithms 1) facilitates the addition of ciphertexts ct_0 and ct_1. Conversely, the HMULT operation (Algorithms 2) involves the multiplication of ct_0 by ct_1. Following HMULT, there is a critical requirement to modify the scaling factor, Δ, of the encapsulated message, which necessitates decreasing the ciphertext level by one. This vital adjustment, termed RESCALE (Algorithms 3), is imperative for preserving the accuracy of computations within the encrypted space. Additionally, the HROTATE operation (Algorithms 4) enables the rotation of the message in ct according to the rotation index sn. Utilizing an evaluation key evk, this operation allows for the rearrangement of elements within the encrypted vector, facilitating sophisticated vectorized computations and manipulations in the encrypted domain. The function FrobeniusMap(m, sn), operating in the

Algorithm 1 HADD$(\text{ct}_0, \text{ct}_1)$

1: $\text{ct}_0 \rightarrow (a_0, b_0)$, $\text{ct}_1 \rightarrow (a_1, b_1)$
2: $d_0 = a_0 + a_1$
3: $d_1 = b_0 + b_1$
4: **return** (d_0, d_1)

Algorithm 2 HMULT$(\text{ct}_0, \text{ct}_1, \text{evk})$

1: $\text{ct}_0 \rightarrow (a_0, b_0), \text{ct}_1 \rightarrow (a_1, b_1)$
2: $d_2 \leftarrow (a_0 \odot a_1), d_0 \leftarrow (b_0 \odot b_1)$
3: $d_1 \leftarrow (a_1 \odot b_0 + b_1 \odot a_0)$
4: $(c_0', c_1') \leftarrow$ Key-Switch(d_2, evk)
5: **return** $\mathbf{ct'} = (d_1 + c_0',\ d_0 + c_1')$

Algorithm 3 RESCALE(ct)

1: ct $\rightarrow ([a]_{\mathcal{C}_\ell}, [b]_{\mathcal{C}_\ell})$
2: $a'^{(j)} \leftarrow [q_\ell^{-1}(a^{(j)} - \text{NTT}([\text{iNTT}(a^{(\ell)})]_{q_j}))]_{q_j, j\in[0,\ell-1]}$
3: $b'^{(j)} \leftarrow [q_\ell^{-1}(b^{(j)} - \text{NTT}([\text{iNTT}(b^{(\ell)})]_{q_j}))]_{q_j, j\in[0,\ell-1]}$
4: **return** $\left([a']_{\mathcal{C}_{\ell-1}}, [b']_{\mathcal{C}_{\ell-1}}\right)$

Algorithm 4 HROTATE(ct, sn, evk)

1: ct $\rightarrow (a, b)$
2: $a' \leftarrow$ FrobeniusMap(a, sn)
3: $b' \leftarrow$ FrobeniusMap(b, sn)
4: $(a'', b'') \leftarrow$ Key-Switch$(a'$, evk$)$
5: **return** $\mathbf{ct}' = (a'', b' + b'')$

NTT domain, achieves the mapping $m(X) \rightarrow m(X^{5^{sn}})$. The Key-Switch operation will be elaborated on in later sections.

C. Bootstrapping

Before the level of the ciphertext is exhausted by successive operations, bootstrapping must be performed on the ciphertext to increase its level before more operations can be performed on the ciphertext. Here we briefly explain the CKKS bootstrapping algorithm. The algorithm consists of four parts: Modulus Raising (ModRaise), Coefficient to Slot (CoeffToSlot), Approximated Modulo Operation (EvalMod), and slot to Coefficient (SlotToCoeff).

1) ModRaise: A ciphertext ct is generated by encrypting a plaintext $m(X)$, with ct initially having a modulus $q = q_0$ and a level of zero. The objective is to increase the modulus (or the level) of ct. Initially, the modulus of ct is elevated to Q_L, which is the modulus of a newly encrypted ciphertext, resulting in an updated ciphertext ct$'$. Although this process increases the level, it also introduces an error polynomial $q \cdot I(X)$ to the plaintext, where Decrypt(ct$'$) = $t(X) = m(X)+q\cdot I(X)$. Here, $I(X)$ is a polynomial whose coefficients are small integers determined by the key distribution. To mitigate this introduced error, an approximated modulo operation is homomorphically applied, aiming to eliminate the error while preserving the encrypted data's integrity. This step is crucial for enabling further operations on the ciphertext without compromising the decryption accuracy.

2) CoeffToSlot: The plaintext $t(X)$, resulting from decrypting ct' in the ModRaise, decodes to $\vec{z}$. The aim of the CoefficienToSlot operation is to compute ct_1 and ct_2, which contain the messages $\vec{z}_1 = (t_0, t_1, \ldots, t_{N/2-1})$ and $\vec{z}_2 = (t_{N/2}, t_{N/2+1}, \ldots, t_{N-1})$, respectively. ct_1 and ct_2 are computed by evaluating the encoding circuit, a linear transformation applied to ct:

$$\vec{z}_1 = 1/N \cdot (\overline{\mathrm{V}}^T \vec{z} + \mathrm{V}^T \overline{\vec{z}}),\ \vec{z}_2 = 1/N \cdot (\overline{\mathrm{W}}^T \vec{z} + \mathrm{W}^T \overline{\vec{z}})$$

, where

$$\mathrm{V} = \begin{pmatrix} 1 & \omega_0 & \cdots & \omega_0^{\frac{N}{2}-1} \\ 1 & \omega_1 & \cdots & \omega_1^{\frac{N}{2}-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_{\frac{N}{2}-1} & \cdots & \omega_{\frac{N}{2}-1}^{\frac{N}{2}-1} \end{pmatrix}, \mathrm{W} = \begin{pmatrix} \omega_0^{\frac{N}{2}} & \omega_0^{\frac{N}{2}+1} & \cdots & \omega_0^{N-1} \\ \omega_1^{\frac{N}{2}} & \omega_1^{\frac{N}{2}+1} & \cdots & \omega_1^{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ \omega_{\frac{N}{2}-1}^{\frac{N}{2}} & \omega_{\frac{N}{2}-1}^{\frac{N}{2}+1} & \cdots & \omega_{\frac{N}{2}-1}^{N-1} \end{pmatrix}$$

3) EvalMod: For the two ciphertexts ct_1 and ct_2, our objective is to apply the modular reduction to each element within their messages, denoted as $(t_i \mod q)$ for all i. Given the absence of direct modular reduction capability in Fully Homomorphic Encryption (FHE), a bipartite approximation strategy for the modulo operation is employed. Initially, the modulo q operation is emulated through a scaled sine function, expressed as $f(t) = \frac{q}{2\pi} \sin\left(\frac{2\pi t}{q}\right)$. This approach leverages the observation that each value t_i within the messages $\vec{z_1}$ and $\vec{z_2}$ closely aligns with $q \cdot I$ for an integer I within the range $(-K, K)$ for a modestly sized K, and that the scaled sine function closely mimics the behavior of the modulo operation in proximity to $q \cdot I$. Subsequently, owing to the sine function's inaccessibility within FHE, it is further approximated by a polynomial representation. Through the application of this evaluation process to ct_1 and ct_2, we generate two resultant ciphertexts, ct_1' and ct_2', whose messages are $\vec{z}_1'$ and $\vec{z}_2'$, respectively, effectively achieving the desired modular reduction approximation.

4) SlotToCoeff: This is the opposite of CoeffcienToSlot. With $\text{ct}_1'$and ct_2', we compute an output ciphertext, ct_{fresh}, which contains message $\vec{z}_{fresh}$, by evaluating a linear transformation, as shown below:

$$\vec{z}_{fresh} = \mathrm{V}\vec{z}_1' + \mathrm{W}\vec{z}_2' = \mathrm{V}(\vec{z}_1' + i\vec{z}_2')$$

Here, ct_{fresh} is approximately equal to the message of ct before ModRaise.

III. Computationally Intensive Operations

We present the primitive operations in CKKS that are computationally intensive for hardware acceleration.

A. Key-Switch

Let an evaluation key be evk $= (\text{evk}_i)_{i\in[0,dnum)}$:

$$\text{evk}_i = (evk_i^{(j)} = (a_{evk_i}^{(j)}, b_{evk_i}^{(j)}))_{j\in[0,k+L]}$$

$(a_{evk_i}^{(j)}, b_{evk_i}^{(j)})$ are a pair of polynomials in the NTT domain. The evk technique reduces computional cost by balancing L, which decomposes the Q into dnum slices $(\{Q_j\}_{0\le j<dnum} = \{\prod_{i=j\alpha}^{(j+1)\alpha-1} q_i\}_{0\le j<dnum}$, where $\alpha = (L+1)/dnum)$.Key-Switch decomposes the input polynomial $[d]_{\mathcal{C}_\ell}$ into $[d_j]_{\mathcal{C}_j'}$, where $j \in [0, \beta)$, extends the moduli of the decomposed parts using ModUp, multiplies them by an evaluation key, and finally reduces the moduli to the original level using ModDown. Please refer to Algorithm 5 for the steps of the algorithm. Key-Switch is a critical component of operations such as HMULT. Based on our testing, this operation occupies more than 90% of the computational time for HMULT, and approximately 50% for Bootstrapping. Based on our tests, this operation also greatly requires acceleration.

B. Fast Basis Conversion

Fast basis conversion (BConv) converts the residue of a polynomial into a new basis that is coprime to the original basis. For a basis $\mathcal{D} = \{p_0, \ldots, p_{k-1}, q_0, \ldots, q_{\ell-1}\}$, let $\mathcal{B} = \{p_0, \ldots, p_{k-1}\}$ and $\mathcal{C} = \{q_0, \ldots, q_{\ell-1}\}$ be its subbases. Then one can convert the RNS representation $[a]_{\mathcal{C}} =$

Algorithm 5 Key-Switch$([d]_{\mathcal{C}_\ell}, \text{evk})$

1: $\vec{d} \leftarrow \text{Dcomp}(d),\ (d_j)_{j\in[0,\beta-1]} \leftarrow \vec{d}$
2: $[\tilde{d}_j]_{\mathcal{D}_\beta} \leftarrow \text{ModUp}([d_j]_{\mathcal{C}'_j})$ for $j \in [0, \beta-1]$
3: $(c_0^{(i)}, c_1^{(i)}) = \Sigma_{j=0}^{\beta-1} \tilde{d}_j^{(i)} \odot \mathbf{evk}_j^{(i)}$ for $i \in [0, k+\alpha\beta-1]$
4: $([c_0]_{\mathcal{C}_\ell}, [c_1]_{\mathcal{C}_\ell}) \leftarrow \text{ModDown}([c_0]_{\mathcal{D}_\beta}), \text{ModDown}([c_1]_{\mathcal{D}_\beta})$
5: **return** $([c_0]_{\mathcal{C}_\ell}, [c_1]_{\mathcal{C}_\ell})$

$(a^{(0)}, \ldots, a^{(\ell-1)}) \in \mathbb{Z}_{q_0} \times \cdots \times \mathbb{Z}_{q_{\ell-1}}$ of an integer $a \in \mathbb{Z}_Q$ into an element of $\mathbb{Z}_{p_0} \times \cdots \times \mathbb{Z}_{p_{k-1}}$ by computing

$$\text{BConv}_{\mathcal{C}\to\mathcal{B}}([a]_C) = \left\{ \left[\sum_{j=0}^{\ell-1} [a^{(j)} \cdot \hat{q}_j^{-1}]_{q_j} \cdot \hat{q}_j \right]_{\boldsymbol{p}_i} \right\}_{0\le i<k}$$

where $\hat{q}_j = \prod_{j'\neq j} q_{j'} \in \mathbb{Z}$. CKKS involves specific operations like modulus switching and the Key-Switch, which can't be directly carried out on RNS components. Therefore, it requires the use of approximate modulus switching techniques designed for RNS representations. The purpose of the approximate modulus raising algorithm, denoted by ModUp, is to find the RNS representation of an integer $a \in \mathbb{Z}_{PQ}$ for the basis $\mathcal{D}$. ModDown reverts the modulus space of a to the original and divides its value further by $P = \prod p_i$. These algorithms are used for modulus switching before and after Key-Switch, respectively:

$$\begin{aligned}
\text{ModUp}_{\mathcal{C}\to\mathcal{D}} &: \prod_{j=0}^{\ell-1} R_{q_j} \to \prod_{i=0}^{k-1} R_{p_i} \times \prod_{j=0}^{\ell-1} R_{q_j} \\
&: [a]_{\mathcal{C}} \to (\text{BConv}_{\mathcal{C}\to\mathcal{B}}([a]_{\mathcal{C}}), [a]_{\mathcal{C}}), \\
\text{ModDown}_{\mathcal{D}\to\mathcal{C}} &: \prod_{i=0}^{k-1} R_{p_i} \times \prod_{j=0}^{\ell-1} R_{q_j} \to \prod_{j=0}^{\ell-1} R_{q_j} \\
&: ([a]_{\mathcal{B}}, [b]_{\mathcal{C}}) \to ([b]_{\mathcal{C}} - \text{BConv}_{\mathcal{B}\to\mathcal{C}}([a]_{\mathcal{B}})) \cdot [P^{-1}]_{\mathcal{C}}
\end{aligned}$$

C. NTT/iNTT

The naive multiplication of two polynomials $a(X) = \sum_{i=0}^{N-1} a_i X^i$ and $b(X) = \sum_{i=0}^{N-1} b_i X^i$ in $\mathcal{R}_q$ results in a polynomial $c(X) = \sum_{i=0}^{N-1} c_i X^i \in \mathcal{R}_{q_i}$, whose coefficients $(c_0, \ldots, c_{N-1})$ are the negacyclic convolution of the two sequences $(a_0, \ldots, a_{N-1})$ and $(b_0, \ldots, b_{N-1})$. This negacyclic convolution is replaced by performing two NTTs on each integer sequence, a pointwise modular multiplication between the two sequences in the NTT domain, and an iNTT on the output of the pointwise multiplication. Given that the computational complexity for both NTT and iNTT is $\mathcal{O}(N \log N)$, and the pointwise multiplication is $\mathcal{O}(N)$, the overall complexity is reduced from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$. This significant reduction in complexity is a crucial advantage for performing polynomial multiplication efficiently in the context of cryptographic applications. Fig. 2 analyses the time consumption proportions for HMULT and Bootstrapping on the CPU platform. The results show that accelerating NTT/iNTT and BConv is crucial for enhancing the overall performance of the CKKS scheme.

IV. Accelerors for CKKS

This section provides an overview of designing a CKKS accelerator. First, we comprehensively understand the computational components and the data flow. Based on this, we analyze a state-of-the-art hardware/software co-design architecture proposed in [12]. Finally, we concentrate on the implementation of reconfigurable Processing Element (PE).

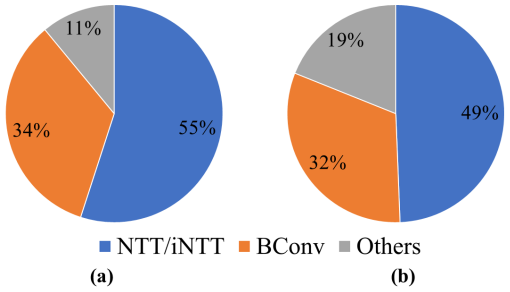


Fig. 2: Proportions of time consumption of NTT/iNTT and BConv in (a) HMULT and (b) Bootstrapping.

A. Computational Components and Data Flow

In the comprehensive CKKS scheme, we focused on the Key-Switch operation, which plays a crucial role in homomorphic encryption workloads and bootstrapping. We employ the Key-Switch process as a paradigm for analyzing computational complexity.

Fig. 3 illustrates the computational flow of Key-Switch, in which multiple NTT/iNTT or BConv operations are executed in parallel. Therefore, the focus of CKKS accelerator research is twofold. On one hand, reducing the calculation latency of the computationally intensive components, and on the other hand, enabling more simultaneous computations of the components. The principal focus in accelerator design is to enhance the parallelism within the overall architecture. Since NTT/iNTT and BConv require different optimization mechanisms, the paramount consideration in enhancing the parallelism of polynomial computations involves balancing the workload between NTT/iNTT and BConv.

As shown in Fig. 4, existing parallelism mechanisms can be categorized into residue parallelism and coefficient parallelism [17]. In residue parallelism, the representation of large integer polynomials through the residue-polynomial method allows for the concurrent computation of multiple residue-

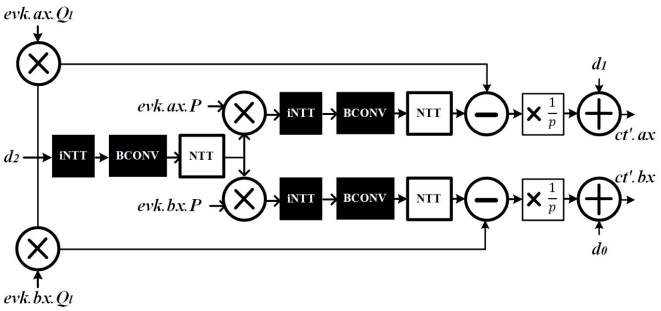


Fig. 3: Computational flow of Key-Switch in HMULT.

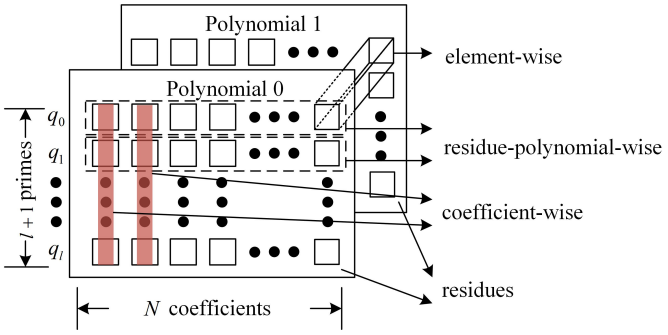


Fig. 4: Two degrees of parallelism.

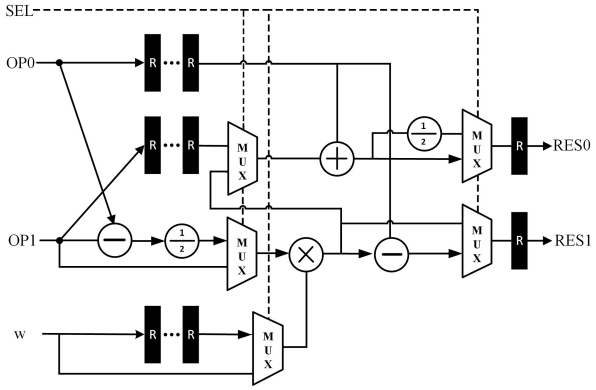


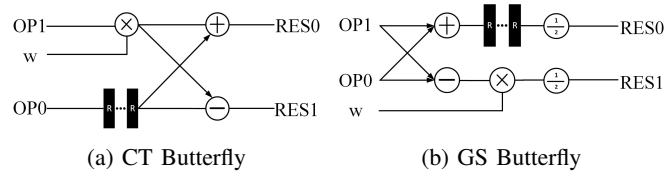
Fig. 6: PE implementation proposed in [18].

polynomial polynomials. This mechanism's parallelism relies on the RNS representation, which allows each accelerator to independently process a given residue-polynomial. This enables each accelerator to compute all coefficients within a small RNS polynomial simultaneously and independently. Each process handles a single RNS polynomial, distributing the polynomial's n coefficients among different processors for computation. In terms of polynomial addition and coefficient-wise operations such as dyadic multiplication, neither scheme exhibits a clear advantage. However, in terms of overall data communication, the coefficient parallelism introduces additional data transfer overhead during NTT operations, whereas the residue parallelism incurs extra data exchange during BConv.

B. Processing Element

From our analysis of the computation flow of NTT/iNTT and BConv, it is evident that these basic operators require substantial computational power and memory throughout the entire CKKS. Therefore, another major research direction is to improve the Area-Time Product (ATP) of the PEs.

CT butterfly and GS butterfly shown in Fig.5 are the basic computational units for NTT/iNTT. Zhang et al. [18] proposed a compact PE merging the two butterfly in Fig. 6. The compact structure also supported modular multiplication, modular addition, and modular subtraction for polynomial operations with little extra control logic. Furthermore, due to the property that $2^{14} \equiv 2^{12} - 1 \pmod{12289}$, they presented a constant-time modular reduction operation without the need for additional multiplication, enabling rapid and low-complexity computations in NTT/iNTT. [18] achieved the best performance and an improvement of approximately 3× in terms of ATP compared with the results of state-of-the-art designs. Aasaraai et al. [19] introduced a fully pipelined butterfly operation unit for NTT/iNTT operations, employing various techniques from [20] to reduce the complexity of computations. By leveraging the unique characteristics of the prime in the Goldilocks field $\mathbb{F}p$, modular multiplication with ω was simplified to a 64-bit unsigned number multiplication, accompanied by a series of 64-bit additions and subtractions. This optimization was facilitated by the construction of an initiation interval 8-level pipeline within the butterfly operation unit.



(a) CT Butterfly (b) GS Butterfly

Fig. 5: Butterfly Unit

C. Architecture

Poseidon [12] designed an FPGA-based accelerator for CKKS that employed the coefficient parallelism mechanism in Fig. 7. It designed a minimal computational core focusing on modular addition and multiplication operations. Utilizing the Barrett reduction algorithm, a fine-grained decomposition strategy was implemented, and a subtractor was engineered to enhance ModAdd operations. They disassemble the Mod-Mult computations and core structure, converting the modular multiplication process into a sequence of small-digit multiplication and subtraction operations. A cascading arrangement of these foundational computational cores was developed, leading to the establishment of the RNSconv architecture, which played a crucial role in expediting the Key-Switch

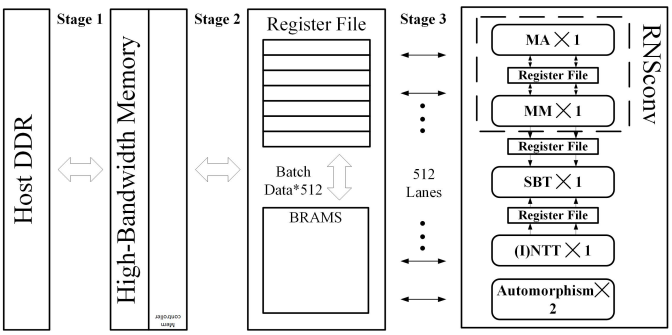


Fig. 7: Architecture proposed in Poseidon [12].

TABLE II: Performance Comparisons of Basic Operations.

Operation	SEAL [6]	HEAX [4]	Poseidon [12]	Speedup
HADD	35.56	4,161	13,310	374×
HMULT	0.38	119	273	718×
NTT	9.25	237	12,474	1,348×
Key-Switch	0.4	104	312	780×
HROTATE	0.39	/	302	774×
RESCALE	6.9	/	3,948	572×

phase. Poseidon achieved maximal parallelization and versatility of the approach by decomposing high-level operations into basic operator-level computations. Utilizing the abundant off-FPGA bandwidth provided by High Bandwidth Memory (HBM) and on-FPGA memory registers and Block RAMs (BRAMs), Poseidon constructed an efficient data transmission link. Moreover, based on the number of HBM channels, a division strategy for a polynomial vector was developed, enabling the parallel computation of multiple coefficients within a polynomial vector through the coefficient parallelism mechanism. Furthermore, leveraging the aforementioned architecture, they devised a highly parallelized automorphism acceleration method, HFAuto, which facilitates the construction of high-performance CKKS accelerators on FPGAs.

As shown in Table II, compared with the FPGA-based accelerator HEAX [4], Poseidon achieves 3× and 50× speedup on Key-Switch and NTT, respectively.

V. Conclusion and Future Directions

Currently, FHE schemes represented by CKKS are rapidly evolving in terms of algorithm optimization and hardware acceleration. This survey deconstructs the computational flow of the CKKS scheme, identifies the performance bottlenecks, and explains the fundamental principles for accelerating computationally intensive operators. By summarizing recent advancements in the field of FPGA-based acceleration of the CKKS scheme, we will continue to explore how to design more efficient dedicated hardware architectures in the future, to accommodate a variety of operators based on different parameter sets in FHE. In conclusion, further unleashing the excellent security features of lattice-based cryptography at the algorithmic level and fully exploiting software/hardware co-design to implement a high-performance and reliable FHE computing platform is one of the most crucial directions for the development of confidential computing technology in intricate privacy-preserving applications.

References

[1] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*. Springer, 2017, pp. 409–437.

[2] V. Demichev, L. Szyrwiel, F. Yu, G. C. Teo, G. Rosenberger, A. Niewienda, D. Ludwig, J. Decker, S. Kaspar-Schoenefeld, K. S. Lilley *et al.*, "dia-pasef data analysis using fragpipe and dia-nn for deep proteomics of low sample amounts," *Nature communications*, vol. 13, no. 1, p. 3944, 2022.

[3] A. Alabdulatif, I. Khalil, A. Y. Zomaya, Z. Tari, and X. Yi, "Fully homomorphic based privacy-preserving distributed expectation maximization on cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 11, pp. 2668–2681, 2020.

[4] M. S. Riazi, K. Laine, B. Pelton, and W. Dai, "Heax: An architecture for computing on encrypted data," in *Proceedings of the twenty-fifth international conference on architectural support for programming languages and operating systems*, 2020, pp. 1295–1309.

[5] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "Bootstrapping for approximate homomorphic encryption," in *Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part I 37*. Springer, 2018, pp. 360–384.

[6] "Microsoft SEAL (release 4.1)," https://github.com/Microsoft/SEAL, Jan. 2023, microsoft Research, Redmond, WA.

[7] "Lattigo v5," Online: https://github.com/tuneinsight/lattigo, Nov. 2023, ePFL-LDS, Tune Insight SA.

[8] W. Jung, S. Kim, J. H. Ahn, J. H. Cheon, and Y. Lee, "Over 100x faster bootstrapping in fully homomorphic encryption through memory-centric optimization with gpus," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 114–148, 2021.

[9] S. Fan, Z. Wang, W. Xu, R. Hou, D. Meng, and M. Zhang, "Tensorfhe: Achieving practical computation on encrypted data using gpgpu," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 922–934.

[10] M. Han, Y. Zhu, Q. Lou, Z. Zhou, S. Guo, and L. Ju, "coxhe: A software-hardware co-design framework for fpga acceleration of homomorphic computation," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2022, pp. 1353–1358.

[11] M. Van Beirendonck, J.-P. D'Anvers, F. Turan, and I. Verbauwhede, "Fpt: A fixed-point accelerator for torus fully homomorphic encryption," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 741–755.

[12] Y. Yang, H. Zhang, S. Fan, H. Lu, M. Zhang, and X. Li, "Poseidon: Practical homomorphic encryption accelerator," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 870–881.

[13] R. Agrawal, L. de Castro, G. Yang, C. Juvekar, R. Yazicigil, A. Chandrakasan, V. Vaikuntanathan, and A. Joshi, "Fab: An fpga-based accelerator for bootstrappable fully homomorphic encryption," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 882–895.

[14] L. Ding, S. Bian, and J. Zhang, "Pima-lpn: Processing-in-memory acceleration for efficient lpn-based post-quantum cryptography," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[15] J. Kim, S. Kim, J. Choi, J. Park, D. Kim, and J. H. Ahn, "Sharp: A short-word hierarchical accelerator for robust and practical fully homomorphic encryption," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–15.

[16] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Advances in Cryptology–EUROCRYPT 2010: 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30–June 3, 2010. Proceedings 29*. Springer, 2010, pp. 1–23.

[17] J. Zhang, X. Cheng, L. Yang, J. Hu, X. Liu, and K. Chen, "Sok: Fully homomorphic encryption accelerators," *arXiv preprint arXiv:2212.01713*, 2022.

[18] N. Zhang, B. Yang, C. Chen, S. Yin, S. Wei, and L. Liu, "Highly efficient architecture of newhope-nist on fpga using low-complexity ntt/intt," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 49–72, 2020.

[19] K. Aasaraai, E. Cesena, R. Maganti, N. Stalder, J. Varela, and K. Bowers, "Cyclonentt: An ntt/fft architecture using quasi-streaming of large datasets on ddr-and hbm-based fpga platforms," *Cryptology ePrint Archive*, 2022.

[20] A. P. Goucher, "An efficient prime for number-theoretic transforms," https://cp4space.hatsya.com/2021/09/01/an-efficient-prime-for-number-theoretic-transforms, Sep. 2021, accessed: 2022-09-23.