Invited Paper: HERMES: Homomorphic Encryption over Residual Number System for Multi-level EvaluationS

Antian Wang
antian.wang@pfw.edu
Purdue University, Fort Wayne
Fort Wayne, Indiana, USA

Kaiyuan Zhang
kaiyuan.zhang@tufts.edu
Tufts University
Medford, Massachusetts, USA

Keshab K. Parhi
parhi@umn.edu
University of Minnesota
Minneapolis, Minnesota, USA

Yingjie Lao
yingjie.lao@tufts.edu
Tufts University
Medford, Massachusetts, USA

Abstract

Homomorphic encryption enables computations on the ciphertext to preserve data privacy. However, its practical deployment has been hindered by the significant computational overhead compared to the plaintext computations. In response to this challenge, we present HERMES, a novel hardware acceleration system designed to explore the computation flow of the CKKS homomorphic encryption bootstrapping process. Among the major contributions of our proposed architecture, we first analyze the properties of the CKKS computation data flow and propose a new scheduling strategy by partitioning the computation modules into general-purpose and special-purpose modular computation modules to allow smaller resource consumption and flexible scheduling. The computation modules are also reconfigurable to reduce the memory access overhead during the intermediate computation. We also optimize the CKKS computation dataflow to improve the regularity with reduced control overhead.

CCS Concepts

• **Computer systems organization** → **Architectures**; • **Security and privacy** → **Cryptography**.

Keywords

Homomorphic encryption, CKKS scheme, Bootstrapping, FPGA

ACM Reference Format:
Antian Wang, Kaiyuan Zhang, Keshab K. Parhi, and Yingjie Lao. 2024. Invited Paper: HERMES: Homomorphic Encryption over Residual Number System for Multi-level EvaluationS. In *Proceedings of International Conference on Computer-Aided Design (ICCAD'24).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3676536.3697124

ICCAD'24, Oct 27–31, 2024, New Jersey, NJ

ACM ISBN 979-8-4007-1077-3/24/10
https://doi.org/10.1145/3676536.3697124

1 Introduction

With the growing demands for computational privacy and security, the development of privacy-preserving technology has become preeminent. Applications involving outsourced data collection, model inference, and cloud-based deployment are the ones with the highest urgency in seeking proper technology to face these challenges. Homomorphic Encryption (HE) [36] is one of the emerging technologies addressing such a developing need from potential stakeholders (i.e., private data owners, high-performance model providers, and efficient cloud-computing servers) when deploying and using cloud computing. The clients can send their encrypted private data to the server, and receive the evaluation results from the server without decryption during the evaluation. The results can only be decrypted by the user using their secret keys. It holds significant potential for various industries, including healthcare [31], financial data management [30], and databases [40] to preserve user privacy. While the HE maintains computation confidentiality on the cloud, the security comes with the expense of high computation costs over encrypted data. Such computation overhead This has been a major barrier to the widespread adoption of HE as a standard solution for secure cloud computing.

Most of the existing HE schemes [13, 15] are designed based on the Ring-Learning With Errors (R-LWE) problems introduced in [29], which ensure the post-quantum security by adding noise to the ciphertext. In this work, we target the CKKS scheme [13] using the bootstrapping operation proposed in [8, 9, 11, 12, 20] for hardware acceleration. The scheme supports homomorphic addition, homomorphic multiplication, and homomorphic rotation (automorphism). The support of approximated floating point homomorphic arithmetic evaluations boosts CKKS's popularity for privacy-preserving computation applications [28, 35].

In this paper, we present HERMES, a hardware acceleration architecture design for the CKKS scheme. The design focuses on the evaluations performed on the server side for multi-level evaluations. The contributions of this paper are summarized below:

- We introduce a novel computational paradigm where the Inverse Number Theoretic Transform (INTT) is performed before the Number Theoretic Transform (NTT), extending prior polynomial multiplication architecture designs. To optimize hardware resource utilization and achieve high throughput, we unify the butterfly module for a Multi-path Delay

Commutators (MDC) architecture, maximizing efficiency and performance.
- We leverage the optimized modular multiplication design with consecutive primes and further empower the computation modules to support element-wise modular operations with cascading capabilities. The capability fully leverages existing processing elements. This approach reduces the memory access footprint while enhancing performance and minimizing data transfer overhead.
- Finally, we analyze the CKKS bootstrapping computation flow and introduce optimizations that reduce the computational load. The effectiveness of our architecture is demonstrated under the demanding bootstrapping evaluation.

The rest of the paper is organized as follows. Section 2 briefly reviews the prior works on hardware acceleration for lattice-based cryptography, HE implementations, and bootstrapping evaluation. We present the proposed HERMES, a custom design supporting CKKS bootstrapping in Section 3. In Section 4, we show the experimental results and compare them with prior works. Finally, Section 5 concludes the paper.

2 Background

2.1 Hardware Architectures for Lattice-based Cryptography

Most of the HE schemes are based on lattice-based cryptography, which includes various modular addition/subtraction, modular multiplication and polynomial multiplications. Modular multiplication and polynomial multiplication are the primary research focuses in the literature [47]. Barrett reduction [6] has been adopted for both Post-quantum cryptography [5, 26, 43, 49] and HE [14, 32, 39] hardware implementations for modular multiplication. To reduce the computation resource overhead in modular multiplication, the costly multiplication can be replaced by shifting and additions considering the special structure of the selected moduli [27, 41]. The NTT-based approaches are mainly used in lattice-based cryptography for prime moduli [42, 48] due to the log-linear complexity in polynomial multiplications compared with quadratic complexity in non-NTT approach [44]. Prior architecture designs of FFT butterfly operation using Single-Delay-Feedback (SDF) [21, 34] and Multi-path-Delay-Commuter (MDC) [3, 18, 33, 42] show the capability in supporting high throughput architecture design. These FFT butterfly module designs can be seamlessly transferred to the NTT/INTT butterfly module. Additionally, NTT and INTT can be integrated into a unified module to increase hardware resource utilization [17, 52]. Twiddle factor storage can be halved by examining access patterns [7, 19]. These prior designs provide building blocks for efficient HE hardware acceleration.

2.2 HE Implementations

SEAL [10], and OpenFHE [4] are two representative open-source libraries used for developing and benchmarking various HE designs. However, these software libraries have limited support for custom architectures, which constrains their capability to handle computation-intensive applications like neural network inference. To this end, custom designs over ASIC, FPGA, and GPU micro-architecture have been extended to fulfill the needs for specific computations with superior performance than the CPU [1, 22, 24, 25, 37, 38, 51]. FHE ASIC designs [2, 16, 24, 37, 38] primarily target reducing memory consumption and alleviating data transfer bottlenecks, aiming for scalability and high throughput with superior performance compared to CPU implementations. However, these designs often lack the flexibility needed to adapt to new computational patterns in evolving HE algorithms. In contrast, FPGA acceleration presents a promising alternative, offering greater adaptability to the continuous advancements in HE algorithm development.

The early work in [45] leveraged the AWS cloud FPGA to implement a non-bootstrappable BFV with dedicated modules for the evaluation steps. However, it had limitations in adapting new computational intensive steps, like bootstrapping. Bootstrapping is the challenging part of the HE design, with most of the works in supporting bootstrapping implemented over ASIC given its unconstrained computation and memory resources to handle the massive modular operations [16, 24, 37, 38]. F1 [37] supports bootstrapping for the BGV/BFV scheme while it is designed for computation with small size rather than private deep neural networks inference. A wide range of CKKS-based hardware designs focusing on enhancing performance and bootstrapping acceleration with huge on-chip memory requirements [24, 38]. In the meantime, a 36-bit FHE accelerator with hierarchical micro-architecture is proposed to achieve high performance with a smaller functional block area, compact on-chip memory and lower power consumption [23]. The work in [16] explores the acceleration of inner-product operation and optimizes the NTT and BConv operations through algorithmic derivation. It minimizes the on-chip bandwidth requirements and provides high performance. Apart from that, a chiplet-based HE accelerator is proposed in [2] to preserve monolithic chip design's advance while meeting the emerging demand for privacy-preserving computing.

Only a few works have been proposed to support bootstrapping in FPGA [1] and GPU [22]. The work in [1] improved the algorithm computation flow and memory architecture. It focuses on optimizing the key-switching computation flow to maximize the reuse of ciphertexts, considering the limited on-chip BRAM/URAM resources. However, its design mainly focused on memory-centric optimization. In contrast, our work also explores low-level computation module designs to improve performance and scheduling for HE.

2.3 CKKS Scheme and Bootstrapping

CKKS is a leveled HE for efficient arithmetic operations on encrypted data [13] in real or complex numbers using approximate arithmetic. Due to the inherent noise in HE, iterative evaluations over ciphertexts can lead to increased noise levels, raising the risk of decryption errors. While the original leveled-HE algorithm supports a limited number of computations over a single ciphertext, fully homomorphic encryption (FHE) extends this capability by enabling unlimited computations through a process known as bootstrapping. Bootstrapping effectively refreshes the ciphertext by reducing accumulated noise. Instead of performing additional homomorphic evaluations with the risk of decryption errors, bootstrapping homomorphically decrypts the ciphertext while keeping it encrypted.

The result is a new ciphertext with significantly reduced noise, allowing further computations without compromising the accuracy of decryption.

In CKKS, encoded plaintext $pt^{i'}, i' \in \{0, 1\}$ is encrypted by public keys pk_0, pk_1 as

$$(ct_0^{i'}, ct_1^{i'}) = (pk_{0,i} \cdot e_{0,i}^{i'} + e_{1,i}^{i'} + pt^{i'}, pk_{1,i} \cdot e_{0,i}^{i'} + e_{2,i}^{i'}), \quad (1)$$

where $e_{0,i}^{i'}, e_{1,i}^{i'}, e_{2,i}^{i'}$ are error terms that follow the individual distribution using cryptographic secure sample generation over q_i. $\chi_{\sigma_{err},i}$ denotes a discrete Gaussian distribution over q_i with standard deviation σ_{err}, U_i is an uniform distribution over q_i, and $\mathcal{ZO}_i$ is an distribution over q_i with sample value $\in \{-1, 0, 1\}$ with 0.5 probability for ± 1, and with 0.5 probability for 0. For the CKKS scheme, $e_{i,0}^{i'} \sim \mathcal{ZO}_i$, $e_{i,1}^{i'}, e_{i,2}^{i'} \sim \chi_{\sigma_{err},i}$. The evaluated ciphertext $(ct_{0,i}^{i'}, ct_{1,i}^{i'})$ is decrypted using secret key sk_i:

$$pt_{eval,i} = ct_{0,i}^{i'} + ct_{1,i}^{i'} \cdot sk_i, \quad (2)$$

From Equation (2), decryption operation is essentially a modular reduction of $ct_{0,i}^{i'} + ct_{1,i}^{i'} \cdot sk_i$ over q_i.

To enable recryption over ciphertext, an evaluation key for bootstrapping using sk_i is generated. Then, a modular reduction over q_i is evaluated homomorphically in the ciphertext domain, which is essentially a periodic function of q_i. The function can be approximated with the following scaled sin function [12]:

$$[pt_{eval,i}]_{q_i} = \frac{q_i}{2\pi} \sin\left(\frac{2\pi}{q_i}(ct_{0,i}^{i'} + ct_{1,i}^{i'} \cdot sk_i)\right) + O(\epsilon^3 \cdot q_i) \quad (3)$$

where $[pt_{eval}]_{q_i} \leq \epsilon \cdot q_i$. Equation (3) provides high-level insights into the bootstrapping process, with the following works finding different approximations of the modular reduction function [8, 9, 20]. Bootstrapping is a special type of evaluation step and is more computationally intensive than homomorphic multiplication, as it approximates a high-degree polynomial while maintaining low decryption error. Although scheduling strategies to reduce computational overhead have been studied [46], further optimization of the low-level bootstrapping computations is necessary to accelerate the process. In this work, we analyze the computation flow graph of CKKS bootstrapping to reduce operation overhead.

3 HERMES Architecture

3.1 Scaling Operation for Multi-Level Evaluation

As the leveled CKKS homomorphic evaluation progresses, the ciphertext modulus diminishes, eventually becoming insufficient to support further computations without errors. To this end, the process of so-called rescaling in the Residue Number System (RNS) is needed to reduce the noise level in the ciphertext, thereby minimizing the likelihood of decryption errors. Rescaling operation essentially drops the last modulus of current l moduli, and then scale the q_{l-1} to the remaining ciphertext as $ct' = [q_{l-1}^{-1} \cdot ct] \bmod (\prod_{i=0}^{l-2} q_i)$, decreasing the number of q_i from l to $l-1$, as shown in the high-level computation flow is presented in Fig. 1. This leads to two important insights:

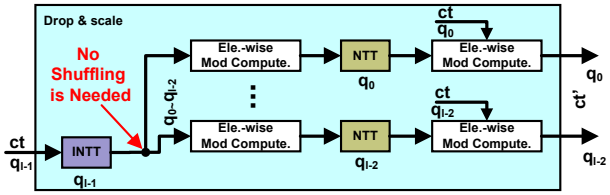

Figure 1: Drop and scale operation computation flow with ct with moduli $q_0, \ldots q_{l-1}$ as the input ciphertext, and ct' with moduli $q_0, \ldots q_{l-2}$ as the output ciphertext.

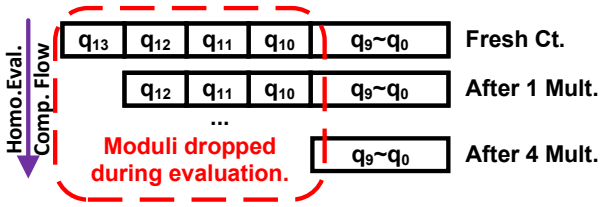

Figure 2: High-level moduli computation profile example for multi-level homomorphic evaluation.

- Not all q_i are used for all homomorphic evaluations. Some of the q_i will be discarded to constrain the noise level for the computation, as demonstrated in Fig. 2.
- The rescaling operations are performed over the coefficient domain between consecutive polynomial multiplication over the evaluation domain.

The first insight suggests that deriving the computation profile for q_i allows for more efficient scheduling of operations across different types of modular computation modules. Excessive resources are required to perform all modular computations using a dedicated module with low utilization for individual q_i. However, by grouping q_i, we can reduce the need to schedule computations on less efficient general-purpose modules and explore resource reduction. Fig. 3 shows a detailed moduli computation profile for the CKKS bootstrapping process. We can observe the non-uniform computation over different moduli for the actual bootstrapping evaluation. We grouped these moduli into *Custom Moduli* and *General Moduli* based on their computation frequency, which balances the workload and reduces resource consumption.

The second insight encourages reconsidering the sequence of modular operations. Rather than following the computation flow in the prior designs, i.e., from NTT → element-wise operation → INTT. Instead, we propose to use the computation from INTT → element-wise operations → NTT as shown in Fig. 1. It eliminates the shuffling operations needed as in NTT → element-wise operation → INTT paradigm. *Switching the computation flow also unlocks new possibilities for designing high-throughput modular computation modules.*

3.2 Custom and General Modular Operations

As discussed in Section 3.1, not all q_i are used along the homomorphic evaluation as shown in Fig. 2, Some of the q_i are dropped along the evaluation process. Therefore, it is possible to improve the overall performance by developing both custom and general

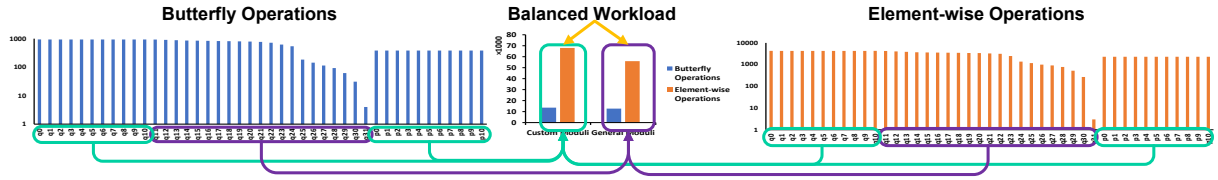

Figure 3: Moduli computation profile for the CKKS bootstrapping implemented in this work.

modules to separate different modular operations. While supporting all moduli in a single module simplifies the computation flow, it leads to excessive resource consumption and low efficiency. On the other hand, a custom module tailored to support specific q_i may suffer from under-utilization. Balancing custom and general modular computation modules is a key design objective of the proposed HERMES.

In our proposed architecture, the custom modular computation modules are designed to support all modular computations over a set of q_i used throughout the homomorphic evaluation. This approach reduces the need for extensive configuration logic that would otherwise be necessary to support a broader range of q_i compared to the general-purpose modular computation modules. In this design, we group all q_i used for the entire bootstrapping process to be computed over these custom modular modules. Meanwhile, the general modular computation modules will support all the remaining q_i that are dropped before the end of the bootstrapping evaluation.

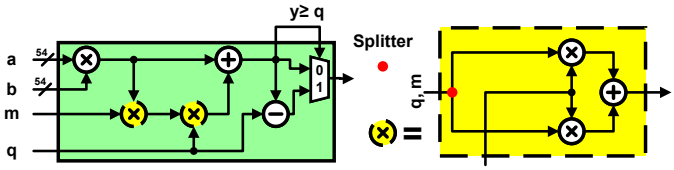

Figure 4: Highly reconfigurable modular multiplier using Barrett reduction module.

Algorithm 1 Modular multiplication with Barrett Reduction.

Input: a and $b \in Z_q$, $m = \lfloor 2^k/q \rfloor$, $k = 2 \cdot \lceil \log_2 q \rceil$
Output: $y = a \cdot b \mod q$
1: $z = a \cdot b$
2: $t = (z \cdot m) >> k$
3: $y = y - (t \cdot q)$
4: **if** $y \geq q$ **then**
5: $y = z - q$
6: **end if**

For the CKKS multi-level evaluations, the q_i values are typically selected starting from the largest possible q_i for the desired security level, gradually decreasing to smaller values until the desired computation level is reached. These selected q_i values are closely spaced, which facilitates optimization for both custom and general modular computation modules. For Barrett reduction shown in steps 2 and 3 of Algorithm 1, it requires a multiple of q and $\lfloor 2^k/q \rfloor$, which also allows the q and $\lfloor 2^k/q \rfloor$ only differ in a small range of bits. The observation enables us to partially decompose the costly multiplication to shift-add operations of partial multiplication results. These two steps are indicated using a dashed line multiplier on the left of Fig. 4 with a splitter to split the constants into multiple separate parts (2 parts as an example shown on the right of Fig. 4). The specific shift-add configuration depends on the values of the q and $\lfloor 2^k/q \rfloor$. This design can effectively reduce the amount of needed DSP for modular multiplications.

3.3 MDC-based Bi-directional Reconfigurable Butterfly Module Design

MDC for NTT/INTT hardware implementation is highly suitable for custom HE hardware design with high throughput [3, 18, 33, 42]. It offers promising scalability for larger HE hardware systems. Unlike traditional memory-based methods that require two dedicated memory blocks to facilitate ping-pong operations during NTT/INTT computation—thus limiting the number of concurrent butterfly operations due to constraints on onboard memory bandwidth and depth—the MDC-based approach utilizes a chain of Delay-Switch-Delay (DSD) modules. This configuration, as illustrated in the top left of Fig. 5, supports full streaming operations with minimal memory overhead, effectively eliminating the need for extensive memory blocks.

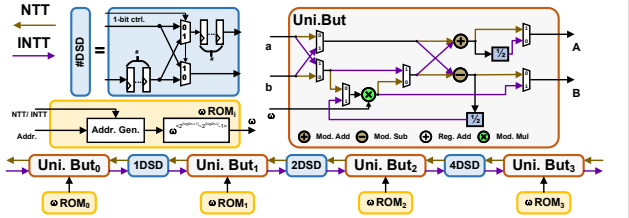

Figure 5: Unified Bi-directional reconfigurable MDC-based butterfly module design for $n = 8$.

A significant limitation of the original MDC approach is its lack of flexibility in utilizing a unified butterfly design [52], as shown in the top right of Fig. 5. The number of registers required varies between NTT and INTT stages, complicating the design. To address this issue, we propose a unified bi-directional reconfigurable MDC-based butterfly module for HERMES. An example with degree-8 is depicted at the bottom of Fig. 5. Given that the size of the registers within the DSD is only related to the polynomial degree n and parallel level of the butterfly operation, the reconfiguration enables the configuring of the unified butterfly computation modules to support both NTT and INTT. Twiddle factors are stored in the ROM, which is directly connected to the corresponding butterfly

operation module for the individual stages. Given that the ω_{2n} is the $2n$-th root of unity, satisfying $\omega_{2n}^{2n} \bmod q = 1$, then by the definition of $2n$-th root of unity, $\omega_{2n}^{n} \bmod q = -1$, thus $\omega_{2n}^{-i} \bmod q = \omega_{2n}^{-i} \cdot (-1) \cdot \omega_{2n}^{n} \bmod q = -\omega_{2n}^{n-i} \bmod q$. The unified butterfly at stage-i needs twiddle factor from $\omega_{2n}^{<2^{\log_2 n-i-1}>}$ to $\omega_{2n}^{<2^{\log_2 n-i}-1>}$ for both NTT and INTT computation, where $< \cdot >$ represents bit-reverse operation for $\log_2 n$-bit. The twiddle factor of INTT can be derived from the twiddle factor ROM stored for NTT. Overall, the proposed architecture can achieve high speed and high throughput.

3.4 Reconfigurable Modular Computation Module

To further accelerate element-wise modular operations, we propose a reconfigurable modular computation module, as illustrated in Fig. 6. The Processing Element (PE) in this module can operate in two modes: independent mode for element-wise operations, and cascade mode for combined multiplication and addition/subtraction operations. This reconfigurability minimizes data transfer overhead between the host and the computation module during bootstrapping, eliminating the need for dedicated memory as in previous works [37, 50]. By reducing control overhead associated with element-wise computations, our design not only improves the efficiency but also enables the design-space exploration of different computation patterns within the CKKS scheme.

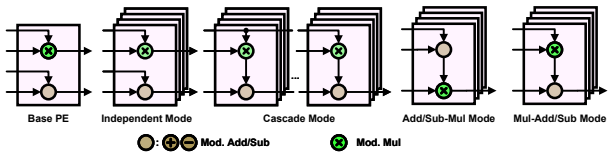

Figure 6: Reconfigurable modular computation module.

In the proposed HERMES, 16 PEs are used for both custom and general modular computation modules for CKKS bootstrapping evaluations. It provides diverse connection patterns based on the target CKKS bootstrapping computation profile, reducing data access time. The reconfigurable modular computation modules decrease data transfer overhead in the bootstrapping evaluation to reduce the running time.

3.5 Bootstrapping Computation Flow Optimization

We further explore the data dependency to remove redundant NTT/INTT operations and streamline the data transfer process, reducing workload and accelerating the computation with high-level concept is shown in Fig. 7. The bootstrapping for CKKS comprises 3 major steps [8], namely coefficients to slots, homomorphic modular reduction, and slots to coefficients. The purpose of coefficients to slots is to map the ciphertext coefficients in plaintext slots to evaluate the modular reduction coefficient-wise. The result of the coefficient to slots is composed of two ciphertexts given each CKKS ciphertext with polynomial degree n, which can store up to at most $n/2$ plaintext values. After the slots to coefficient process, the bootstrapped ciphertext is restored with a low noise level.

In HERMES, we support Chebyshev approximation of sin function as proposed in [8, 9, 12]. The approximation method has been shown to reduce bootstrapping overhead and improve efficiency. We optimize the computation by restructuring the pre-computation phase of intermediate polynomial evaluation to minimize redundant drop and scale operations for identical polynomial results, thereby simplifying the computation flow. This pre-computation strategy decreases butterfly operations by 26% and element-wise modular operations by 5% compared to the software counterpart.

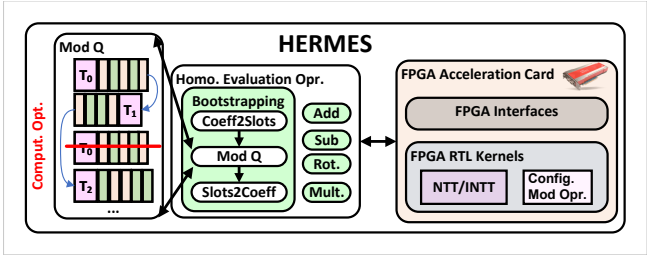

Figure 7: Bootstrapping computation schedule optimization with hardware-software co-design.

4 Experiment

We implement HERMES hardware implementation for homomorphic evaluation in Verilog with 300MHz operation clock frequency. The host CPU code is written in XRT native API for data transfer and scheduling, and HERMES RTL code is packaged into kernel code using the Vitis 2023.1 development platform. The kernel code is compiled and linked into an FPGA executable by the Vitis compiler. The performance of the FPGA implementation is detailed in Table 1.

4.1 HERMES Architecture Design and Step Profiling

For a fair comparison, we evaluate the operation time as bootstrapping per slot. We observe that our proposed architecture can achieve better area-time product (ATP) over LUT, FF, and DSP. The gain is attributed to the grouping of the low-level modular computations into custom moduli and general moduli, as well as the computation flow optimization dedicated to the CKKS bootstrapping process. Besides, we provide the running time breakdown for one round of CKKS bootstrapping, as summarized in Table 2. We compare the running time on FPGA with the software implementation in OpenFHE. The OpenFHE software implementation was running over Intel(R) Core(TM) i7-14700K in Ubuntu 22.04.4 LTS. A total 4.72× speedup for the entire process is achieved, demonstrating the effectiveness of the proposed acceleration.

4.2 Scalability Evaluation

HERMES support MDC bi-directional butterfly modules with 8 coefficients fed simultaneously and polynomial degree of 4096 for NTT/INTT butterfly operations, and 16-parallel computation modules for element-wise modular operations, which include modular addition/subtraction, and modular multiplication. HERMES can be generalized to larger polynomial degrees and higher parallelisms.

Table 1: HERMES FPGA performance comparison of HERMES with prior works of CKKS implementations. Area-time product (ATP) is computed by using the similar metrics as in [25]: (LUT/FF/DSP/BRAM)×Bootstrapping Per Slot (μs)/($\log Q$).

Design	$\log n/\log Q$	Board	Freq (GHz) BootPerSlot (μs)	kLUT/ATP	kFF/ATP	DSP/ATP	BRAM(36K)/ATP	URAM/ATP
HERMES	$(12, 54 \times 33)$	U280	0.3/0.112	466/29.295	462/29.037	3844/0.242	319.5/0.02	36/0.0003
FAB[1]	$(16, 54 \times 23)$	U280	0.3/0.477	899/345.27	2073/796.15	5120/1.966	1920/0.738	960/0.369

Table 2: Running time breakdown in seconds for one round of CKKS bootstrapping and comparison with software.

Evaluation Type	HERMES	OpenFHE	Speedup vs OpenFHE
Coeff2Slots	0.072	0.284	3.94×
Mod Q	0.164	0.941	5.73×
Slots2Coeff	0.046	0.106	2.30×
Total	0.282	1.331	4.72×

Table 3: Resource consumption estimation with the increase of polynomial degree n for HERMES.

$\log_2(n)$	12	13	14	15	16	17	18	19
kLUT	466	487	508	528	549	570	590	611
kFF	462	479	496	512	529	546	563	579
DSP	3844	4084	4324	4564	4804	5044	5284	5524

Table 4: Resource consumption estimation with increased parallelism of modular computation modules for HERMES. Butterfly module size is fixed with a polynomial degree of 4096**. Rows colored in gray cannot be accommodated in Xilinx U280 FPGA Data Center Accelerator Card.**

Num. of Modular Computation Module	kLUT	kFF	DSP
16	466	462	3844
32	536	508	4804
64	676	599	6724
128	955	782	10564
256	1514	1149	18244

For larger polynomial degrees, the computation resource consumption of the MDC bi-directional butterfly modules increases proportionally with n. As illustrated in Table 3, HERMES can be estimated to support polynomial degree up to 2^{19}, utilizing MDC bi-directional butterfly modules with simultaneous processing of 8 coefficients and 16-parallel computation modules for element-wise modular operations. These large polynomial degrees would meet most homomorphic evaluation requirements. Additionally, by maintaining a polynomial degree of 4096 with the current MDC bi-directional butterfly module structure and employing up to 64-parallel modular computation modules, the computations can be further accelerated within the available FPGA resources. We present the resource consumption estimation with increased parallelism of modular computation modules for HERMES in Table 4,

5 Conclusion

This paper presents HERMES, a hardware acceleration system specifically optimized for the CKKS homomorphic encryption scheme, supporting operations such as addition, multiplication, and rotation. Our architecture introduces an innovative computation flow designed to minimize control overhead through high-throughput butterfly operation modules. We also propose a novel partitioning strategy for the underlying computation modules, categorizing them into general and custom types to optimize scheduling and reduce computational resource overhead based on the profile of individual q_i. Additionally, we analyze data dependencies to further reduce evaluation clock cycles.

ACKNOWLEDGMENT

This work is supported in part by the National Science Foundation (NSF) under grant numbers CCF-2243053, CCF-2412357, and SaTC-2426299.

References

[1] Rashmi Agrawal, Leo de Castro, Guowei Yang, Chiraag Juvekar, Rabia Yazicigil, Anantha Chandrakasan, Vinod Vaikuntanathan, and Ajay Joshi. 2023. FAB: An FPGA-based accelerator for bootstrappable fully homomorphic encryption. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, IEEE, 882–895.

[2] Aikata Aikata, Ahmet Can Mert, Sunmin Kwon, Maxim Deryabin, and Sujoy Sinha Roy. 2023. REED: Chiplet-based scalable hardware accelerator for fully homomorphic encryption. *arXiv preprint arXiv:2308.02885* (2023).

[3] Manohar Ayinala, Michael Brown, and Keshab K Parhi. 2011. Pipelined parallel FFT architectures via folding transformation. *IEEE Transactions on Very Large Scale Integration Systems* 20, 6 (2011), 1068–1081.

[4] Ahmad Al Badawi, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Ian Quah, Yuriy Polyakov, Saraswathy R.V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. 2022. OpenFHE: Open-Source Fully Homomorphic Encryption Library, In Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography. *Cryptology ePrint Archive*, 53–63.

[5] Utsav Banerjee, Tenzin S Ukyab, and Anantha P Chandrakasan. 2019. Sapphire: A Configurable Crypto-Processor for Post-Quantum Lattice-based Protocols. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2019), 17–61.

[6] Paul Barrett. 1986. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In *Conference on the Theory and Application of Cryptographic Techniques*. Springer, 311–323.

[7] Mojtaba Bisheh-Niasar, Reza Azarderakhsh, and Mehran Mozaffari-Kermani. 2021. High-speed NTT-based polynomial multiplication accelerator for post-quantum cryptography. In *2021 IEEE 28th Symposium on Computer Arithmetic (ARITH)*. IEEE, 94–101.

[8] Jean-Philippe Bossuat, Juan Troncoso-Pastoriza, and Jean-Pierre Hubaux. 2022. Bootstrapping for approximate homomorphic encryption with negligible failure-probability by using sparse-secret encapsulation. In *International Conference on Applied Cryptography and Network Security*. Springer, 521–541.

[9] Hao Chen, Ilaria Chillotti, and Yongsoo Song. 2019. Improved bootstrapping for approximate homomorphic encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 34–54.

[10] Hao Chen, Kim Laine, and Rachel Player. 2017. Simple encrypted arithmetic library-SEAL v2. 1. In *International Conference on Financial Cryptography and Data Security*. Springer, 3–18.

[11] Jung Hee Cheon, Kyoohyung Han, and Minki Hhan. 2018. Faster homomorphic discrete fourier transforms and improved fhe bootstrapping. *Cryptology ePrint Archive* (2018).

[12] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. 2018. 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, 360–384.

[13] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 409–437.

[14] Sin-Wei Chiu and Keshab K Parhi. 2024. Low-Latency Preprocessing Architecture for Residue Number System via Flexible Barrett Reduction for Homomorphic Encryption. *IEEE Transactions on Circuits and Systems II: Express Briefs* 71, 5 (2024), 2784–2788.

[15] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptology ePrint Archive* 2012 (2012), 144.

[16] Shengyu Fan, Xianglong Deng, Zhuoyu Tian, Zhicheng Hu, Liang Chang, Rui Hou, Dan Meng, and Mingzhe Zhang. 2024. Taiyi: A high-performance CKKS accelerator for Practical Fully Homomorphic Encryption. *arXiv preprint arXiv:2403.10188* (2024).

[17] Yue Geng, Xiao Hu, Minghao Li, and Zhongfeng Wang. 2023. Rethinking Parallel Memory Access Pattern in Number Theoretic Transform Design. *IEEE Transactions on Circuits and Systems II: Express Briefs* 70, 5 (2023), 1689–1693.

[18] Florian Hirner, Ahmet Can Mert, and Sujoy Sinha Roy. 2024. Proteus: A Pipelined NTT Architecture Generator. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 32, 7 (2024), 1228–1238.

[19] Xiao Hu, Jing Tian, Minghao Li, and Zhongfeng Wang. 2022. AC-PM: An area-efficient and configurable polynomial multiplier for lattice based cryptography. *IEEE Transactions on Circuits and Systems I: Regular Papers* 70, 2 (2022), 719–732.

[20] Charanjit S Jutla and Nathan Manohar. 2022. Sine series approximation of the mod function for bootstrapping of approximate HE. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 491–520.

[21] Hans Kanders, Tobias Mellqvist, Mario Garrido, Kent Palmkvist, and Oscar Gustafsson. 2019. A 1 million-point FFT on a single FPGA. *IEEE Transactions on Circuits and Systems I: Regular Papers* 66, 10 (2019), 3863–3873.

[22] Yuhui Bao Kaustubh Shivdikar, Michael Shen Rashmi Agrawal, Evelio Mora Gilbert Jonatan, José L Abellán, Alexander Ingare, John Kim Neal Livesay, and David Kaeli Ajay Joshi. 2023. GME: GPU-based microarchitectural extensions to accelerate homomorphic encryption, In Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture. *MICRO'23, October 28-November 1, 2023, Toronto, ON, Canada*, 670–684.

[23] Jongmin Kim, Sangpyo Kim, Jaewan Choi, Jaiyoung Park, Donghwan Kim, and Jung Ho Ahn. 2023. SHARP: A short-word hierarchical accelerator for robust and practical fully homomorphic encryption. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–15.

[24] Jongmin Kim, Gwangho Lee, Sangpyo Kim, Gina Sohn, Minsoo Rhu, John Kim, and Jung Ho Ahn. 2022. ARK: Fully homomorphic encryption accelerator with runtime data generation and inter-operation key reuse. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1237–1254.

[25] Liang Kong, Guojie Qin, and Shuguo Li. 2023. Design an Efficient FPGA-Based Accelerator for Leveled BFV Homomorphic Encryption. *IEEE Transactions on Circuits and Systems II: Express Briefs* 71, 3 (2023), 1381–1385.

[26] Minghao Li, Jing Tian, Xiao Hu, and Zhongfeng Wang. 2023. Reconfigurable and high-efficiency polynomial multiplication accelerator for CRYSTALS-Kyber. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, 8 (2023), 2540–2551.

[27] Zhe Liu, Hwajeong Seo, Sujoy Sinha Roy, Johann Großschädl, Howon Kim, and Ingrid Verbauwhede. 2015. Efficient Ring-LWE encryption on 8-bit AVR processors. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 663–682.

[28] Qian Lou and Lei Jiang. 2021. HEMET: a homomorphic-encryption-friendly privacy-preserving mobile neural network architecture. In *International conference on machine learning*. 7102–7110.

[29] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2010. On ideal lattices and learning with errors over rings. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1–23.

[30] Oliver Masters, Hamish Hunt, Enrico Steffinlongo, Jack Crawford, Flavio Bergamaschi, Maria E Dela Rosa, Caio C Quini, Camila T Alves, Feranda de Souza, and Deise G Ferreira. 2019. Towards a homomorphic machine learning big data pipeline for the financial services sector. *Cryptology ePrint Archive* (2019).

[31] Kundan Munjal and Rekha Bhatia. 2023. A systematic review of homomorphic encryption and its contributions in healthcare industry. *Complex & Intelligent Systems* 9, 4 (2023), 3759–3786.

[32] Kevin Nam, Hyunyoung Oh, Hyungon Moon, and Yunheung Paek. 2022. Accelerating n-bit operations over TFHE on commodity CPU-FPGA. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*. 1–9.

[33] Keshab K Parhi. 2024. A low-latency FFT-IFFT cascade architecture. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 181–185.

[34] Fahad Qureshi, Syed Asad Alam, and Oscar Gustafsson. 2010. 4K-Point FFT Algorithms based on optimized twiddle factor multiplication for FPGAs. In *2010 Asia Pacific Conference on Postgraduate Research in Microelectronics and Electronics (PrimeAsia)*. IEEE, 225–228.

[35] Ran Ran, Xinwei Luo, Wei Wang, Tao Liu, Gang Quan, Xiaolin Xu, Caiwen Ding, and Wujie Wen. 2023. SpENCNN: orchestrating encoding and sparsity for fast homomorphically encrypted neural network inference. In *International Conference on Machine Learning*. PMLR, 28718–28728.

[36] Ronald L Rivest, Len Adleman, and Michael L Dertouzos. 1978. On data banks and privacy homomorphisms. *Foundations of secure computation* 4, 11 (1978), 169–180.

[37] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Srinivas Devadas, Ronald Dreslinski, Christopher Peikert, and Daniel Sanchez. 2021. F1: A fast and programmable accelerator for fully homomorphic encryption. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 238–252.

[38] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Nathan Manohar, Nicholas Genise, Srinivas Devadas, Karim Eldefrawy, Chris Peikert, and Daniel Sanchez. 2022. Craterlake: a hardware accelerator for efficient unbounded computation on encrypted data. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 173–187.

[39] Kaustubh Shivdikar, Gilbert Jonatan, Evelio Mora, Neal Livesay, Rashmi Agrawal, Ajay Joshi, José L Abellán, John Kim, and David Kaeli. 2022. Accelerating polynomial multiplication for homomorphic encryption on GPUs. In *2022 IEEE International Symposium on Secure and Private Execution Environment Design (SEED)*. IEEE, 61–72.

[40] Benjamin Hong Meng Tan, Hyung Tae Lee, Huaxiong Wang, Shuqin Ren, and Khin Mi Mi Aung. 2020. Efficient private comparison queries over encrypted databases using fully homomorphic encryption with finite fields. *IEEE Transactions on Dependable and Secure Computing* 18, 6 (2020), 2861–2874.

[41] Weihang Tan, Benjamin M Case, Antian Wang, Shuhong Gao, and Yingjie Lao. 2021. High-speed modular multiplier for lattice-based cryptosystems. *IEEE Transactions on Circuits and Systems II: Express Briefs* 68, 8 (2021), 2927–2931.

[42] Weihang Tan, Sin-Wei Chiu, Antian Wang, Yingjie Lao, and Keshab K. Parhi. 2024. PaReNTT: Low-Latency Parallel Residue Number System and NTT-Based Long Polynomial Modular Multiplication for Homomorphic Encryption. *IEEE Transactions on Information Forensics and Security* 19 (2024), 1646–1659.

[43] Weihang Tan, Antian Wang, Yingjie Lao, Xinmiao Zhang, and Keshab K Parhi. 2021. Pipelined High-Throughput NTT Architecture for Lattice-Based Cryptography. In *2021 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. IEEE, 1–4.

[44] Weihang Tan, Antian Wang, Xinmiao Zhang, Yingjie Lao, and Keshab K Parhi. 2023. High-speed VLSI architectures for modular polynomial multiplication via fast filtering and applications to lattice-based cryptography. *IEEE Trans. Comput.* (2023).

[45] Furkan Turan, Sujoy Sinha Roy, and Ingrid Verbauwhede. 2020. HEAWS: An accelerator for homomorphic encryption on the Amazon AWS FPGA. *IEEE Trans. Comput.* 69, 8 (2020), 1185–1196.

[46] Tommy White, Charles Gouert, Chengmo Yang, and Nektarios Georgios Tsoutsos. 2023. FHE-Booster: Accelerating fully homomorphic execution with fine-tuned bootstrapping scheduling. In *2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 293–303.

[47] Jiafeng Xie, Wenfeng Zhao, Hanho Lee, Debapriya Basu Roy, and Xinmiao Zhang. 2024. Hardware Circuits and Systems Design for Post-Quantum Cryptography–A Tutorial Brief. *IEEE Transactions on Circuits and Systems II: Express Briefs* 71, 3 (2024), 1670–1676.

[48] Yufei Xing and Shuguo Li. 2019. An efficient implementation of the NewHope-Simple key exchange on FPGAs. *IEEE Transactions on Circuits and Systems I: Regular Papers* 67, 3 (2019), 866–878.

[49] Tianyu Xu, Yijun Cui, Dongsheng Liu, Chenghua Wang, and Weiqiang Liu. 2022. Lightweight and efficient hardware implementation for Saber using NTT multiplication. In *2022 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*. IEEE, 601–605.

[50] Yang Yang, Sanmukh R Kuppannagari, Rajgopal Kannan, and Viktor K Prasanna. 2022. Bandwidth Efficient Homomorphic Encrypted Matrix Vector Multiplication Accelerator on FPGA. In *2022 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 1–9.

[51] Yinghao Yang, Huaizhi Zhang, Shengyu Fan, Hang Lu, Mingzhe Zhang, and Xiaowei Li. 2023. Poseidon: Practical homomorphic encryption accelerator. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 870–881.

[52] Neng Zhang, Bohan Yang, Chen Chen, Shouyi Yin, Shaojun Wei, and Leibo Liu. 2020. Highly efficient architecture of NewHope-NIST on FPGA using low-complexity NTT/INTT. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2020, 2 (2020), 49–72.