Towards FPGA Acceleration of CKKS in Microsoft SEAL: A Microkernel-Based Architectural Exploration

Massimiliano Donati[1], Samuele Bartorelli[1], Guido Falai[1], Daniele Rossi[1], and Sergio Saponara[1]

Department of Information Engineering, University of Pisa, Pisa, Italy
massimiliano.donati@unipi.it

Abstract. Homomorphic Encryption enables computation on encrypted data, making it a key technology for privacy-preserving cloud computing and artificial intelligence workloads. Among modern schemes, CKKS is well suited for approximate numerical computation, but its adoption remains limited by the cost of polynomial arithmetic over large residue-number-system moduli. Microsoft SEAL, a widely adopted open-source library, implements CKKS through reusable arithmetic primitives. Based on this decomposition, this paper presents a preliminary architectural study of FPGA-based microkernels to accelerate these arithmetic operations. The work targets modular addition, modular multiplication, and forward and inverse Number Theoretic Transforms as the core microkernels for CKKS ciphertext operations and investigates their implementation on pipelined and parallel FPGA architectures. The proposed microkernels have been implemented and evaluated on a Xilinx UltraScale+ device to assess the performance, scalability, and feasibility of an FPGA accelerator tightly integrated with Microsoft SEAL.

Keywords: Homomorphic Encryption · CKKS · FPGA · Microsoft SEAL · Modular arithmetic · Hardware acceleration.

1 Introduction

Homomorphic Encryption (HE) enables arithmetic operations directly on encrypted data, allowing computations to be outsourced to untrusted platforms without revealing the underlying cleartexts [1]. This capability makes HE a key technology for privacy-preserving cloud computing and artificial intelligence.

Among modern HE schemes, the Cheon–Kim–Kim–Song (CKKS) scheme [2] is particularly attractive because it supports approximate arithmetic over real and complex numbers, making it suitable for encrypted signal processing, scientific computing, and machine learning workloads. Based on the Ring Learning With Errors (RLWE) problem, CKKS operates over the cyclotomic polynomial ring $R_q = \mathbb{Z}_q[X]/(X^N+1)$, where N is the polynomial degree and q is the ciphertext modulus represented in Residue Number System (RNS) form as $q = \prod_{i=0}^{L} q_i$,

with pairwise coprime moduli $q_i < 2^{60}$. Practical implementations employ polynomial degrees from 2^{12} to 2^{15} and modulus chains composed of four to eight moduli, each between 30 and 60 bits. A CKKS ciphertext consists of two or more polynomials and packs up to $N/2$ cleartext values. As a result, ciphertext evaluation repeatedly performs coefficient-wise modular additions, modular multiplications, and forward/inverse Number Theoretic Transform (NTT/INTT) operations over thousands of polynomial coefficients.

Microsoft SEAL [3] is one of the most widely adopted HE libraries. It decomposes high-level evaluator operations into reusable arithmetic primitives acting on polynomial coefficients and RNS residues, exposing the computational kernels that dominate execution time of ciphertext elaborations. Despite significant algorithmic and implementation advances, HE remains several orders of magnitude slower than cleartext computation, limiting its practical deployment [4].

Several approaches have been proposed to improve CKKS arithmetic. On general-purpose processors, SIMD vectorization accelerates modular arithmetic and NTT operations through AVX-512 and SVE on Intel and Arm CPUs [5, 6], while GPU implementations such as HEonGPU [7] exploit massive data parallelism to accelerate ciphertext evaluation. On FPGA, HEAX [8] and Medha [9] accelerate arithmetic operations and evaluation pipelines. More recently, heterogeneous hardware-software architectures such as CraterLake [10] and ASIC solutions such as BASALISC [11] demonstrate the benefits of a customized architecture for lattice-based HE. Collectively, these works highlight the effectiveness of hardware acceleration and the central role of efficient arithmetic kernels.

This paper investigates FPGA architectures for the arithmetic microkernels underlying the CKKS evaluator in Microsoft SEAL. The proposed architectures have been implemented on a Xilinx UltraScale+ device to assess their performance and scalability as building blocks for a tightly integrated accelerator.

The remainder of this paper is organized as follows. Section II identifies the arithmetic microkernels underlying the CKKS evaluator in Microsoft SEAL. Section III presents the proposed FPGA architectures, Section IV reports the implementation results, and Section V concludes the paper.

2 Arithmetic Microkernels in Microsoft SEAL

The modular organization of Microsoft SEAL provides a basis for hardware acceleration. Instead of targeting complete CKKS evaluator functions, this work follows library decomposition to identify reusable arithmetic kernels that dominate execution time and can be efficiently mapped onto FPGA architectures.

As shown in Fig. 1, evaluator operations are decomposed into kernels repeatedly applied to all polynomial coefficients and RNS moduli. Ciphertext addition is dominated by modular additions, ciphertext multiplication by Barrett modular multiplications [12], whereas ciphertext rotation relies on forward and inverse NTT together with key-switching, which internally reuses the addition and multiplication microkernels. The transform kernels are based on Harvey and inverse Harvey butterflies [13].

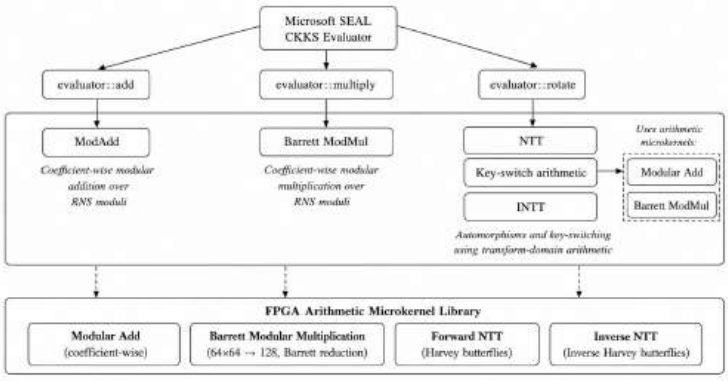

Fig. 1. CKKS arithmetic microkernel decomposition in Microsoft SEAL.

3 FPGA Arithmetic Microkernels

3.1 Modular Addition

The addition microkernel implements a 64-bit modular addition. Given two coefficients $a, b < q$, it computes $s = (a+b) \bmod m$ using a 64-bit adder followed by a comparator and a subtractor that conditionally performs the modular reduction whenever the sum exceeds the modulus m.

The datapath has a latency of one clock cycle and an initiation interval of one cycle (II=1). Its compact architecture and limited hardware cost enable straightforward replication to increase throughput. A complete ciphertext addition is realized by applying this microkernel element-wise to the N coefficients of each polynomial for every modulus q_i in the modulus chain.

3.2 Modular Multiplier

The multiplication microkernel implements a 64-bit modular multiplication. Given two operands $a, b < q$, it computes $p = (a \cdot b) \bmod m$, where the 128-bit product is reduced to a 64-bit residue modulo m.

As shown in Fig. 2, the microkernel consists of a pipelined 64×64→128 Karatsuba multiplier [14], followed by a pipelined Barrett modular reduction unit [12]. Karatsuba reduces the number of large integer multiplications by recursively decomposing the operands into smaller products, whereas the subsequent Barrett reduction computes the 128→64 modular reduction through precomputed constants associated with the modulus $\mu = \left\lfloor \frac{2^{2k}}{q} \right\rfloor$, where $k = \lceil \log_2 q \rceil$, followed by shifts and a final conditional correction to obtain the residue modulo m.

The datapath has a fixed latency of five clock cycles and II=1. By combining four processing elements in parallel and an additional modular adder, the *Quad MulBarrett* architecture performs the four modular multiplications required for one coefficient-wise dyadic product every clock cycle. Repeated over all N coefficients and RNS moduli q_i, it realizes a complete ciphertext multiplication.

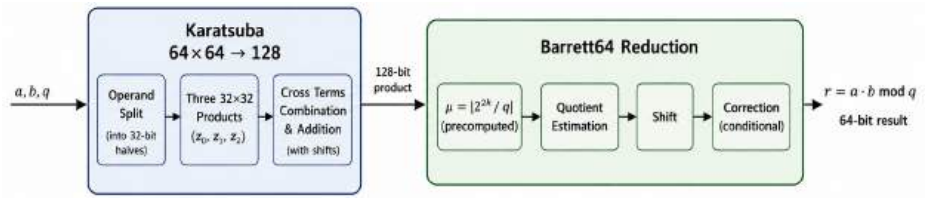

Fig. 2. Architecture of the Barrett modular multiplication microkernel.

3.3 Number Theoretic Transform

This microkernel implements the polynomial transform underlying CKKS arithmetic. Since the NTT and INTT share the same architecture, differing only in the twiddle-factor sequence and in the use of Harvey or inverse Harvey butterflies [13], the following discussion focuses on the INTT microkernels.

The INTT includes $\log_2(N)$ butterfly stages. At each stage, coefficient pairs (u, v) are processed according to the inverse butterfly, $u' = u + v, \quad v' = (u - v) \cdot \omega^{-i} \bmod q$, where ω^{-i} is the inverse twiddle factor (i.e. the i-th inverse power of a primitive N-th root of unity modulo q).

As shown in Fig. 3, the architecture comprises an input coefficient memory, a dual-path *iTwiddle* unit, and a dual-path butterfly engine. The *iTwiddle* unit generates the inverse twiddle factors in parallel for even and odd indices, storing two factors per clock cycle in a dedicated dual-port memory. The butterfly engine processes two coefficient pairs per clock cycle by simultaneously reading the corresponding twiddle factors from the dual-port memories, performing inverse Harvey butterfly operations, and writing the intermediate and final results back into the coefficient memory. Both the *iTwiddle* unit and the butterfly engine adopt a dual-path organization exploiting FPGA dual-port memories, allowing two independent butterfly operations to be executed in parallel.

The pipelined butterfly engine achieves a latency of seven clock cycles and an II=1. A complete INTT is realized by generating the inverse twiddle factors for each modulus q_i and applying the butterfly engine to all coefficient pairs across the $\log_2(N)$ transform stages. The same architecture is independently applied to every ciphertext polynomial and every modulus in the RNS chain.

4 Results and Discussion

The proposed microkernels were synthesized and implemented on an AMD/Xilinx Zynq UltraScale+ ZCU106 FPGA using Vivado. Table 1 summarizes the post-implementation timing and resource utilization estimates.

The modular adder requires minimal hardware resources, making it suitable for extensive replication. The Barrett multiplier provides the basic processing element, while the *Quad MulBarrett* exhibits an almost linear increase in hardware resources through datapath replication, with only a limited reduction in operating frequency. Conversely, the transform kernels are primarily memory-bound:

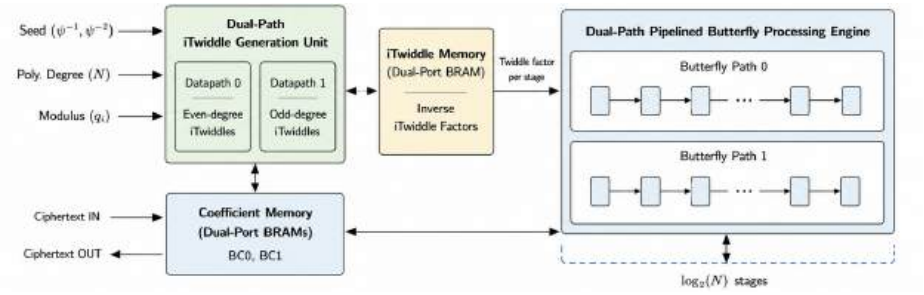

Fig. 3. High-level architecture of the dual-path INTT microkernel.

the dual-path implementation increases computational resources while preserving both memory utilization and operating frequency by exploiting dual-port memories instead of replicating coefficient and twiddle memories.

To assess the potential impact on CKKS computation at the microkernel level, the clock cycles of the proposed FPGA microkernels were compared with the corresponding software kernels on an NVIDIA Grace processor using the `PERF_COUNT_HW_CPU_CYCLES` counter, for a typical CKKS configuration with $N = 8192$ and RNS modulus chain $60, 40, 40, 60$. The software baseline consists of scalar C++ kernels derived from Microsoft SEAL, compiled with GCC 11.4.0 using `-O3 -mcpu=native` and executed on a pinned CPU core under Ubuntu 22.04. The FPGA kernels significantly reduce the required clock cycles. The modular addition, single Barrett multiplier, and *Quad MulBarrett* achieve reductions of 31.7%, 39.7%, and 84.9%, respectively. Similarly, the single- and dual-path INTT reduce the cycle count by 46.1% and 73.2%, confirming the effectiveness of datapath parallelism for arithmetic and transform kernels. Since the FPGA and CPU operate at different frequencies, these reductions should not be interpreted as elapsed-time speedups.

These results reveal complementary scalability trends: modular arithmetic mainly benefits from datapath replication, whereas transform kernels are also constrained by memory organization. This preliminary evaluation is limited to microkernel-level estimates and excludes data transfers, host–FPGA communication, data-layout conversion, and invocation overhead. Nevertheless, it demonstrates the potential of a microkernel-oriented FPGA design for CKKS arithmetic and provides a basis for future integration and end-to-end evaluation.

Table 1. Post-implementation results of the microkernels on UltraScale+ ZCU106.

Microkernel	Pipeline Stages	Freq. (MHz)	LUT	FF	DSP	BRAM
ModAdd	1	400	192	0	0	0
MulBarrett	5	145	1160	418	64	0
Quad MulBarrett	5	120	4714	1671	256	0
Single INTT	7	132	3836	868	64	171
Dual INTT	7	130	6320	1447	128	171

5 Conclusion

This paper presented an FPGA-oriented study of the arithmetic microkernels underlying the CKKS evaluator in Microsoft SEAL. Four reusable microkernels were identified and implemented as pipelined FPGA architectures. The results show that datapath replication effectively accelerates arithmetic kernels, whereas memory organization becomes the main scalability factor for transform kernels. Compared with software-equivalent implementations, the proposed microkernels achieve clock-cycle reductions ranging from 31.7% to 84.9%, providing a solid foundation for a tightly integrated FPGA accelerator for Microsoft SEAL.

Acknowledgments. This work was supported by Horizon Europe AERO (GA No. 101092850), EPI SGA2 (GA No. 101036168), and the Italian Ministry of University and Research (MUR) through FoReLab (Departments of Excellence).

References

1. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*. ACM, 2009.
2. Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology – ASIACRYPT 2017*, pages 409–437. Springer, 2017.
3. Microsoft Research. Microsoft seal, 2024. Release 4.1.
4. Marten van Dijk, Shai Halevi, Jung Hee Cheon, et al. Homomorphic encryption standard, 2023. Version 3.0, https://homomorphicencryption.org.
5. Fabian Boemer et al. Intel hexl: Accelerating homomorphic encryption with intel avx512-ifma52. In *IEEE High Performance Extreme Computing Conference*, 2021.
6. Massimiliano Donati, Guido Falai, Samuele Bartorelli, Daniele Rossi, and Sergio Saponara. SVE-based acceleration of homomorphic encryption arithmetic on ARM neoverse processors. In *2026 IEEE 32nd International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE, 2026.
7. Jaehoon Kim, Donghun Kim, et al. Heongpu: Homomorphic encryption on gpus. *ACM Transactions on Architecture and Code Optimization*, 20(3), 2023.
8. Muhammad Shafique et al. Heax: An architecture for computing on encrypted data. In *Design, Automation and Test in Europe Conference (DATE)*, 2020.
9. Ahmet Can Mert et al. Medha: Microcoded hardware accelerator for computing on encrypted data. In *IEEE Int. Symposium on HPC Architecture*, 2023.
10. Ahmet Can Mert et al. Craterlake: A hardware accelerator for efficient unbounded computation on encrypted data. In *Proceedings of the ACM Int. Conference on Architectural Support for Programming Languages and Operating Systems*, 2022.
11. Tom Geelen et al. Basalisc: Flexible architecture for fully homomorphic encryption. In *IEEE Int. Symposium on HPC Architecture*, 2023.
12. Paul Barrett. Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor. In *Advances in Cryptology – CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*. Springer, 1987.
13. David Harvey. Faster arithmetic for number-theoretic transforms. In *Journal of Symbolic Computation*, volume 44, pages 1502–1517, 2009.
14. A. A. Karatsuba and Yu. Ofman. Multiplication of multidigit numbers on automata. *Soviet Physics Doklady*, 7(7):595–596, 1963.