CoFHEE: A Co-processor for Fully Homomorphic Encryption Execution

Mohammed Nabeel[1], Deepraj Soni[2], Mohammed Ashraf[1], Mizan Abraha Gebremichael[3], Homer Gamil[1], Eduardo Chielle[1], Ramesh Karri[2], Mihai Sanduleanu[3], and Michail Maniatakos[1]
[1]Center for Cyber Security, New York University Abu Dhabi
[2]Center for Cyber Security, New York University Tandon School of Engineering
[3]Khalifa University

***Abstract*—In this paper, we present the blueprint of a specialized co-processor for Fully Homomorphic Encryption, dubbed CoFHEE. With a small design area of $12mm^2$, CoFHEE incorporates ASIC implementations of fundamental polynomial operations, such as polynomial addition and subtraction, Hadamard product, and Number Theoretic Transform, which are underneath all higher-level FHE primitives. CoFHEE has native support of polynomial degrees of up to $n = 2^{14}$ with a coefficient size of 128 bits. We evaluate our chip with performance and power experiments and compare it against state-of-the-art software implementations and other ASIC designs. A more elaborate description of the CoFHEE design can be found in [1].**

***Index Terms*—Data privacy, Encrypted computation, Fully Homomorphic Encryption, Co-processor, ASIC**

I. Introduction

The migration of computation to the cloud has raised privacy concerns as sensitive data becomes vulnerable to attacks since they need to be decrypted for processing. Fully Homomorphic Encryption (FHE) mitigates this issue as it enables meaningful computations to be performed directly on encrypted data. Nevertheless, FHE is orders of magnitude slower than unencrypted computation, hindering its practicality and adoption. Therefore, improving FHE performance is essential for its real world deployment. Research has progressed in software [2], as well as hardware solutions. In this work we focus on the later.

Given a limited design area of $12mm^2$ available to us given budget constraints, we design an architecture that executes the underlying polynomial computation, and it is the base of all high-level FHE operations. This study details the process of developing the frontend and hardware architecture for CoFHEE, the first ASIC co-processor for Fully Homomorphic Encryption (FHE) to be silicon-validated. CoFHEE consists of special units capable of performing several arithmetic operations, and an AHB lite interconnect. CoFHEE supports polynomial degrees of up to $n = 2^{14}$ with a maximum native coefficient size of 128 bits. The target process is 55nm CMOS Globalfoundries.

II. CoFHEE Design Flow Overview

The chip area available for CoFHEE's design is $12mm^2$ and the available technology node is GF $55nm$. Considering this limitation in area, and common encryption parameters used in FHE applications, we can achieve our goal of performing ciphertext multiplication on chip for a maximum polynomial degree $n = 2^{13}$ with 128-bit coefficient sizes (in case of $n = 2^{14}$, extra communication is needed). A 128-bit coefficient size is the largest coefficient we can fit in CoFHEE given design area limitations. The aforementioned restrictions allowed us to develop a system architecture with 1 Processing Element (PE), 3 dual-port and 5 single-port SRAMs. This selection of units enables support of ciphertext multiplication fully on chip, with an II (Initiation Interval) $= 1$, as the dual-port SRAMs provide the ability to fetch and store 2 different operands in the same cycle. The biggest portion of the provided area is occupied by the dual and single-port SRAMs. In the remaining space, we fit the Processing Elements (PE), Multiplier Data Mover and Controller (MDMC), and an Advanced High-Performance Bus (used for internal data flow), Direct Memory Access controller (DMA), General Purpose Configuration registers (GPCFG), and an ARM Cortex-M0 with its own memory. CoFHEE also provides SPI and UART interfaces for external host communication. Lastly, the chip operates at a target frequency of 250 MHz (bounded by the memory latency), and has two voltage supplies, namely 3.3 V (IO pads) and 1.2 V (logic core).

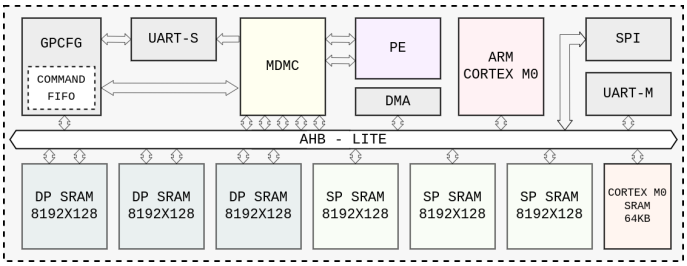


Fig. 1: CoFHEE Top Level Architecture

A. Execution & Operations

The operations of CoFHEE are summarized in Table II. The operations can be divided into two types: Compute and memory operations. Compute operations consist of NTT, inverse NTT, and a set of pointwise operations, namely normal and modular multiplication, modular squaring, modular multiplication by a constant, and modular addition and subtraction. CoFHEE provides an ISA to execute any of these operations when relevant polynomials are loaded into the memory. Meanwhile, memory operations replicate or transfer data from one memory to another. Although compute operations run sequentially similar to memory operations, both types can run simultaneously.

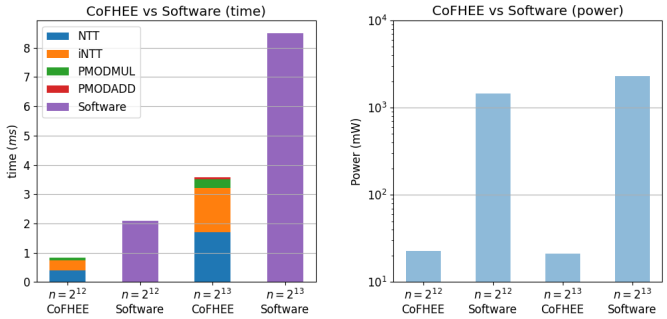
978-3-9819263-7-8/DATE23/© 2023 EDAA

Design	Technology	n	log q* ($bits$)	Area (mm^2 — LUT/FF/BRAM/DSP)	Power (W)	Freq. (Mhz)	Clock Cycles	Efficiency (Performance per mm^2)	Silicon Proven
CoFHEE	ASIC - GF $55nm$	2^{14}	128	12	$2.3 \cdot 10^{-2}$	250	24576	$2.60 \cdot 10^{-4}$	●
F1 [3]	ASIC - GF $14/12nm$	2^{14}	32	151.4	$1.8 \cdot 10^{2}$	1000	16	$1.73 \cdot 10^{-4}$	❍
HEAX [4]	FPGA - Intel Arria10 GX 1150	2^{14}	27	582148 / 1554005 / 3986 / 2018	-	300	1536	†	N/A
Roy [5]	Xilinx Zynq UltraScale+ ZCU102	2^{12}	30	63522 / 25622 / 400 / 200	-	200	16425	†	N/A

TABLE I: Comparative table and performance of the NTT operation against related work
† No information is available to accurately map FPGA resources to silicon area

Command	Inputs n	$[\vec{x}]$	$[\vec{y}]$	$[\vec{\omega}]$	q	n^{-1}	$[\vec{t}]$	δ	$\uparrow$	$\rightarrowtail$	Operations
NTT	•	•		•	•		•				Performs NTT on $\vec{x}$.
iNTT	•	•		•	•	•	•				Performs inverse NTT on $\vec{x}$.
PMODADD	•	•	•		•					•	Pointwise modular addition of $\vec{x}$ and $\vec{y}$.
PMODMUL	•	•	•		•					•	Pointwise mod. multiplication of $\vec{x}$ and $\vec{y}$.
PMODSQR	•	•			•					•	Pointwise modular squaring of $\vec{x}$.
PMODSUB	•	•	•		•					•	Pointwise mod. subtraction of $\vec{x}$ and $\vec{y}$.
CMODMUL	•	•			•					•	Mod. multiplication of $\vec{x}$ by a constant.
PMUL	•	•	•							•	Pointwise multiplication of $\vec{x}$ and $\vec{y}$.
MEMCPY								•	•	•	Memory-to-memory data transfer.
MEMCPYR								•	•	•	Memory data transfer in bit-reverse.

TABLE II: CoFHEE's operations. $[\cdot]$: memory address function, n: polynomial degree, $\vec{x}$ and $\vec{y}$: polynomials, $\vec{\omega}$: twiddle factors, q: modulus, n^{-1}: inverse of n, $\vec{t}$: temporary values, δ: length (in words), $\uparrow$: source address, $\rightarrowtail$: output/destination address



(a) Time for all towers (b) Power

Fig. 2: Comparison to CPU execution

III. Experimental Evaluation

A. Power and latency

We measure the latency and power for several operations supported by CoFHEE using $n = \{2^{12}, 2^{13}\}$. Peak power is observed during NTT, while Hadamard and iNTT consume less power. The iNTT operation involves multiplication with a constant (n^{-1}), and a decimation in frequency operation. Its average power is lower as the constant multiplication consumes less power, and therefore reduces the average. For the same reason, iNTT takes more cycles than NTT to execute. To summarize, CoFHEE needs a power supply with peak power rating of around $30mA$ and an average power of around $25mA$ to run polynomial multiplication in a fraction of a millisecond.

B. Comparison to CPU

We compare CoFHEE against a software implementation in terms of execution time and power consumption using a ciphertext multiplication without relinearization. We use one instance of CoFHEE. For the software implementation, we use the Microsoft SEAL 3.7 library [6] running on an AMD Ryzen 7 5800h (TSMC 7nm FinFET) at 3.8Ghz with 16GB of RAM on Ubuntu 20.04 LTS, and we collect the power measures using `powertop`. We set $n = \{2^{12}, 2^{13}\}$ and $\log q = \{109, 218\}$ bits, which provide a security level of 128 bits against classical computers. Fig. 2 presents the results. For $(n, \log q) = (2^{12}, 109)$, the SEAL implementation takes $2.1ms$, while CoFHEE needs $0.84ms$ to finalize the ciphertext multiplication. When $(n, \log q) = (2^{13}, 218)$, the software implementation spends $8.5ms$, while CoFHEE takes $3.58ms$ to operate on the same task.

IV. Related Work

When comparing CoFHEE with the closest work (F1), to establish a fair evaluation, we normalize the performance in terms of the area and scaling factor between the technology nodes. In order to derive the scaling factor, the Barrett modular multiplier is synthesized using the GF14/12nm technology library (same as F1). Results indicate a scaling factor that reduces the area by $9.8\times$ and the critical path by $8\times$. After normalizing the performance figures, F1 performs $1.73 \cdot 10^{-4}$ NTT operations per ns per mm^2, while CoFHEE achieves $2.6 \cdot 10^{-4}$, which implies a speedup of $1.5\times$. The speed-up is mainly attributed to the use of a pipelined Barrett multiplier, instead of an iterative Montgomery multiplier.

V. Conclusion

In this paper, we presented the basic components of the design, front-end, and architectural of a specialized co-processor for Fully Homomorphic Encryption. An extended version of our work with details on the front-end, back-end, and post silicon validation efforts can be found in [1].

References

[1] M. Nabeel, D. Soni, M. Ashraf, M. A. Gebremichael, H. Gamil, E. Chielle, R. Karri, M. Sanduleanu, and M. Maniatakos, "Silicon-proven ASIC design for the polynomial operations of Fully Homomorphic Encryption," *arXiv preprint arXiv:2204.08742*, 2023.

[2] E. Chielle, O. Mazonka, H. Gamil, N. G. Tsoutsos, and M. Maniatakos, "E3: A framework for compiling c++ programs with encrypted operands," *Cryptology ePrint Archive*, 2018, online: https://eprint.iacr.org/2018/1013, GitHub repository: https://github.com/momalab/e3.

[3] A. Feldmann, N. Samardzic, A. Krastev, S. Devadas, R. Dreslinski, K. Eldefrawy, N. Genise, C. Peikert, and D. Sanchez, "F1: A fast and programmable accelerator for fully homomorphic encryption (extended version)," *arXiv preprint arXiv:2109.05371*, 2021.

[4] M. S. Riazi, K. Laine, B. Pelton, and W. Dai, "HEAX: An architecture for computing on encrypted data," in *ASPLOS '20*, 2020, p. 1295–1309.

[5] S. S. Roy, F. Turan, K. Jarvinen, F. Vercauteren, and I. Verbauwhede, "FPGA-based high-performance parallel architecture for homomorphic computing on encrypted data," in *HPCA*, 2019.

[6] "Microsoft SEAL (release 3.7)," https://github.com/Microsoft/SEAL, Sep. 2021, microsoft Research, Redmond, WA.