TREBUCHET Fully Homomorphic Encryption Accelerator: Phase Two Performance Estimation Results

David Bruce Cousins, Yuriy Polyakov
Ahmad Al Badawi
Duality Technologies
{dcousins, ypolyakov, aalbadawi}@dualitytech.com

Matthew French, Andrew Schmidt, Ajey Jacob, Benedict Reynwar, Kellie Canida, Akhilesh Jaiswal, Clynn Mathew
USC, Information Sciences Institute
{mfrench, aschmidt, ajacob, breynwar, kcanida, akjaiswal, cmathew}@isi.edu

Austin Ebel, Negar Neda,
Brandon Reagen
New York University
{abe5240, nn2231, bjr5}@nyu.edu

Naifeng Zhang, Franz Franchetti
Carnegie Mellon University
{naifengz, franzf}@cmu.edu

Patrick Brinich, Jeremy Johnson
Drexel University
{pbrinich, jjohnson}@drexel.edu

Mike Franusich
SpiralGen, Inc
{mike.franusich}@spiralgen.com

Bo Zhang, Zeming Cheng, Massoud Pedram
University of Southern California
{zhangb, chengz, pedram}@usc.edu

***Abstract*— Secure computation is critically important across DoD, finance, healthcare -- anywhere personally identifiable information (PII) is accessed. Traditional encryption-based security requires data to be decrypted before computation, making it vulnerable when processed on untrusted systems. Fully Homomorphic Encryption (FHE) keeps data encrypted during computation, even in untrusted environments. However, FHE requires significant computer power compared to similar operations on unencrypted data. FHE must significantly close this computation gap (to within 10x) to make encrypted processing practical for everyday use. The TREBUCHET project is developing a hardware accelerator for deep FHE machine learning applications, under the DARPA MTO Data Privacy for Virtual Environments (DPRIVE) program. We integrate with the OpenFHE library and accelerate its standard FHE schemes. This paper is an update to the progress made in Phase 2 of the program.**

I. Introduction

As the TREBUCHET project completes the second phase of the DPRIVE program we present results on the estimated performance of the system accelerating the OpenFHE public domain software library [1] with a custom ASIC based accelerator. Previously our team presented the design approach for an FHE accelerator [2] to support privacy-preserving computation, enabling computation on encrypted data without the need to decrypt the input. In addition to encryption at rest, and in transit, data can now be encrypted in use, providing end to end encrypted analysis of sensitive data, even if computation is done on untrusted computer resources.

The TREBUCHET custom hardware accelerator was designed to address the root causes of existing high compute overhead of software-only FHE implementations – TREBUCHET takes advantage of the underlying mathematical transforms used with a very large numbers of parallel ALUs performing vectorized modular arithmetic. The target performance is to achieve 10^4 speedup over a single thread software only implementation.

II. Technical Approach

The fundamental design goal of TREBUCHET is to support 1) a wide array of complex and very deep encrypted computing applications, 2) all the most important lattice based FHE schemes with 3) a modular design that maps to a wide range of chip sizes with 4) runtime performance orders of magnitude faster than other solutions. We do this by providing basic design blocks and a system stack architecture (see Figure 1), that is highly adaptable and extensible. TREBUCHET provides mix-and-match layers for applications, software system components and hardware, respectively.

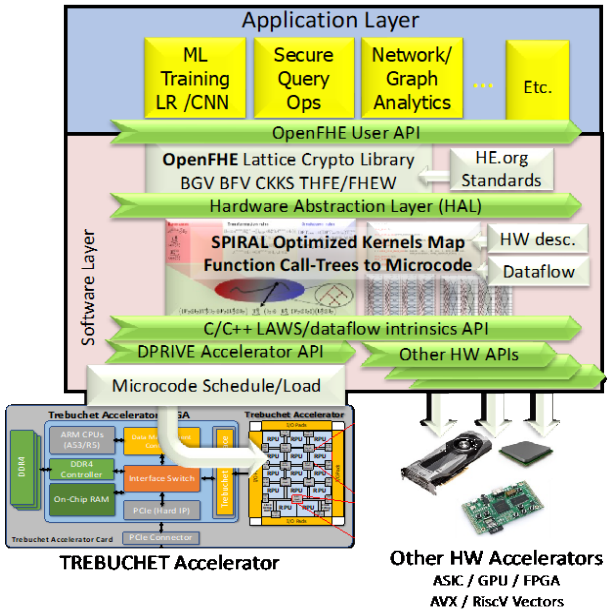

Figure 1 - TREBUCHET Layered System Architecture.

Our final design was driven by the needs of Convolutional Neural Net (CNN) training using the CKKS approximate floating point FHE scheme with full CPA-IND^D security [3]. This level of security allows sharing of all decrypted results without restriction, but also increases the bitwidth requirements of the underlying moduli used. This drives fundamental system design decisions that support very large vector sizes (2^17 elements) and a data path width of 128 bits. OpenFHE supports 128-bit arithmetic using standard C++ software emulation, so we directly accelerate encrypted analysis workloads where results are shared among multiple participants and the algorithms implemented require a minimum of single precision floating point (precision is selectable by the choice of runtime parameters).

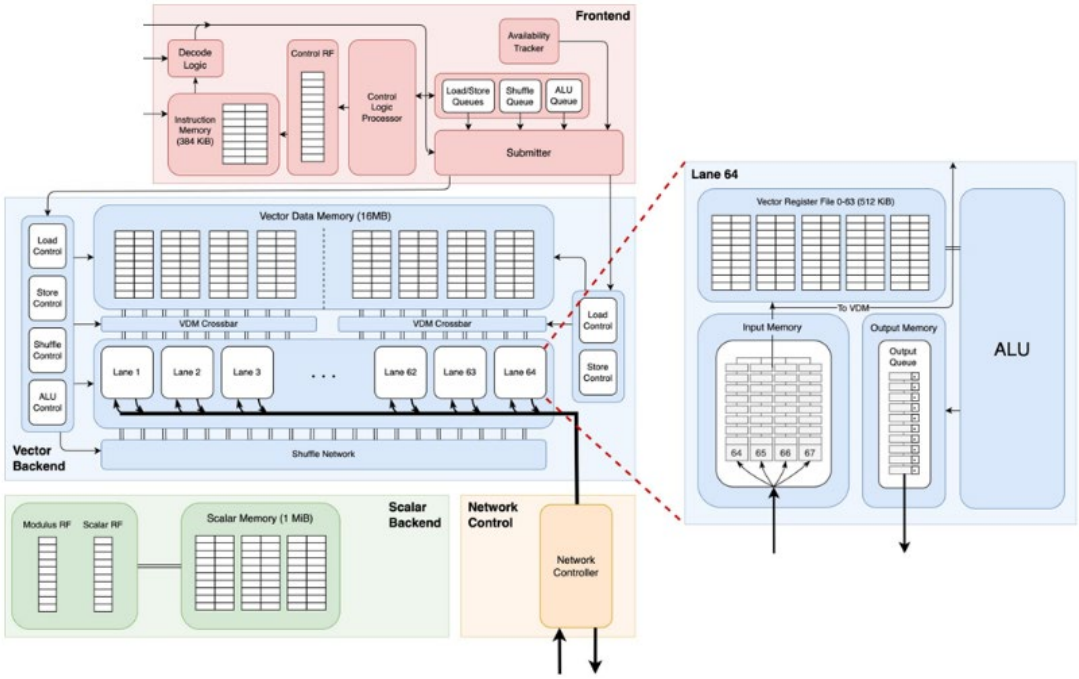

Figure 2 - Trebuchet Ring Processor Unit (RPU) Architecture.

III. THE HARDWARE LAYER

The TREBUCHET Accelerator was designed specifically to balance the compute, memory, and I/O needs of FHE processing. FHE processing benefits from parallelization. The processor was designed to be modular, allowing parallelism at multiple levels. The basic component of the architecture is the Ring Processing Unit (RPU) [4] (Figure 2). These are large on-chip tiles, that contain multiple ALU lanes for vectorized processing of modulo math via specialized scalable modulo multipliers and adders, connected to shared vector-data SRAM to buffer ciphertext(s) and keys. Tiles also facilitate memory management by scheduling data to be near computational elements through the use of a shuffle network and a vector register file. Our tool flow enables generation of RPUs with user defined mixes of ALUs and memory, allowing us to scale the RPU to different chip areas or even multiple chiplets. In this paper we will describe two different configurations.

IV. THE SOFTWARE LAYER

We improved the structure and functionality of OpenFHE Hardware Abstraction Layer (HAL). We extensively reorganized the Lattice and Math interfaces for increased consistency, efficiency, and futureproofing. As part of this, we finalized a new NativePoly interface - critical for the Trebuchet backend design - and identified optimization opportunities within existing backends. Initial implementation of some optimizations, such as a 30% boost in single-core forward NTT performance for the default native backend, has already been integrated into OpenFHE. These advancements lay the groundwork for a more flexible and powerful HAL infrastructure.

A. HAL Integration with Surrogate FPGA Accelerator Carrier Board

To facilitate early testing, we relied on a readily available KCU105 FPGA development board. This setup features a 32-bit MicroBlaze processor with 512KB of addressable on-chip memory, memory mapped registers and FIFO-based mailboxes, 2GB of DDR4, DMA controller and Gen3x8 PCIe interface. This processor handling all control tasks, mimicking a future MSoC ARM CPUs. We've configured it with memory, FIFOs, and PCIe connectivity to facilitate communication with the host machine. Once backend development progresses, we plan to implement a smaller version of the Trebuchet RPU within the FPGA itself, allowing us to verify the software interface and refine communication protocols before the final hardware arrives. This interim solution ensures smooth development progress until the full-fledged card is ready.

The initial development stage employs a host-card program pair for remote polynomial processing. This setup utilizes an implicit memory coherence protocol with a dedicated context object managing card initialization, communication, and data transfers. Modified DCRTPoly/Poly classes in OpenFHE accommodate card-side storage and validity flags, enabling specialized card-side arithmetic operations. This temporary design facilitates testing and development until the full hardware architecture is available.

In OpenFHE a Poly class represents a polynomial as a fixed length vector of integers (representing the coefficient of each term of the polynomial, defined at compile time as being 32-, 64-, or 128- bit in order to use native compiler integers. The length of vector is the number of terms or the polynomial, usually a large power of two (in our case most commonly 2^16 or 2^17). There is also a modulus and other ancillary metadata

variables associated with each polynomial. A DCRTPoly class represents a polynomial with LAWS integer coefficients (thousands of bits) but represented in residue form using the Chinese Remainder Theorem (CRT). Thus, instead of a vector of LAWS integers, there is a "tower" or vector of native integer vectors, with associated moduli. Our design supports O(32) towers. Thus, all math on the CPU and the Accelerator can be performed in the native word size (in our case 128- or 64-bits. The list of implemented and tested operations can be found in Table 3.

Table 3. Current implementation list of card-side arithmetic operations. "∘" denotes multiplication, addition, or subtraction.

`DCRTPoly ∘= DCRTPoly`
`DCRTPoly ∘= Poly`
`DCRTPoly ∘= Integer`
`DCRTPoly = DCRTPoly ∘ DCRTPoly`
`DCRTPoly = DCRTPoly ∘ Poly`
`DCRTPoly = DCRTPoly ∘ Integer`
`Poly ∘= Poly`
`Poly ∘= Integer`
`Poly = Poly ∘ Poly`
`Poly = Poly ∘ Integer`

B. *On-chip Pseudo Random Number Generation Algorithm Development*

FHE Keys represent a large part of our I/O budget. We determined that approximately half the key is a random number field that could be co-generated in both hardware and software from a single seed, using a cryptographic pseudo-random number generator. Our design requires a robust random number generator for its hardware architecture, particularly for efficient 128-bit integer generation modulo q <128-bit strictly without modular bias. We designed a new "hardware-friendly" rejection sampling algorithm (See Figure 3) with significantly lower rejection rates compared to OpenFHE's current method. Specifically, the rejection rate is estimated as $(2^{128} - y)/2^{128}$. By utilizing this algorithm we expect cryptographically secure randomness for FHE sampling, minimizing unnecessary processing overhead associated with rejected samples.

- Given a 128-bit random integer x (generated with Blake2 either on the host or on the TREBUCHET RPU from a synchronized seed)
- Given constant $y = k * q$, where k is the largest integer such that $y < 2^{128}$
- reject x if $x > y$ else return $x \% q$.

Figure 3 - Low rejection-rate PRNG sampling algorithm for arbitrary uniform variables $[\mathbf{0} \ldots \boldsymbol{q})$.

C. *SPIRAL LAWS Optimizations*

We use the SPIRAL system to generate NTT and inverse NTT microcode generation. These made significant performance strides: 1) vectorized bit-reversal algorithm specifically for Automorphism Transforms was developed and validated in the RPU simulator; 2) vectorized twiddle generation techniques, both unrolled and looped, were implemented, slashing the number of twiddles loaded from memory for a 2^{17} NTT by over 99%. This optimized microcode was also validated in the RPU simulator; and 3) A crucial integration effort linked SPIRAL-generated code through the RPU simulator to large scale simulation on the AFRL PALLADIUM chip emulation system, successfully verifying microcode correctness and resolving any integration issues.

V. Performance Model

Trebuchet's performance analysis relies on a comprehensive model known as the "speed-of-light" model. This model focuses on Hybrid Key Switching (HKS), the most significant on-chip operation with minimal off-chip traffic due to intermediate result caching. The crux of the model lies in optimizing data movement for HKS. It calculates the precise computational requirements (ring additions and multiplications) and data transfers between on-chip and off-chip memory for each HKS step. Armed with this detailed breakdown, the model employs four key parameters:

1. ALU clock rate: Determines the processing speed of on-chip computations.
2. Data movement speed (I/O bandwidth): Defines the rate of data transfer between on-chip and off-chip memory.
3. On-chip memory size: Dictates the capacity for storing intermediate results, minimizing off-chip communication.
4. FHE scheme cryptographic parameters such as ring dimension, number, and size of moduli.

By taking these factors into account, the model calculates the ideal "speed-of-light" runtime for HKS, encompassing both computation and data transfer. For workload estimations, we assume sequential execution of HKS operations on the hardware accelerator. Therefore, the estimated workload runtime is simply the product of the individual HKS latency and the total number of HKS operations required. It's essential to note that HKS complexity is depends on the size of the input DCRTPoly. The model effectively incorporates this factor, offering a more nuanced assessment than a simple worst-, best-, or average-case analyses. This refinement further strengthens the model's reliability and accuracy in real-world workload projections.

A. *Hardware Configurations*

We modelled the performance of two hardware configurations, GPIO and SerDes, for two potential shuttle runs. Table 4 compares the two hardware configurations highlighting their strengths and weaknesses for different applications. SerDes packs a smaller chip and faster processing in a tighter package with higher data transfer rate. GPIO is a simpler chip to prove out the fundamental RPU designs.

Table 4. Chip configurations for GPIO and SerDes designs.

Specification	GPIO	SerDes
Total area of chip	175 mm2	150 mm2
Process size for chip [nm]	GF12LP	GF12LP
Base clock rate [GHz]	0.5	1.0
Power consumption	< 100 W	< 100 W
Integer size	128-bit	64-bit
Device Bandwidth (2 RPUs)	12.8 GB/s	200 GB/s
I/O requirements	536 pins	760 pins
# Computational ALUs	32	256

B. Speed of Light Model

We developed the speed of light model to estimate workload performance based on the total number of Hybrid Key Switching (HKS) operations. Our model minimizes off-chip data movement and calculates the required computation (in ring additions and multiplications) and data transfer between off-chip and on-chip memory for each HKS step. Given the ALU clock rate, data movement speed (I/O bandwidth), and on-chip memory size, the model determines the theoretical minimum runtime for computation and data transfer within HKS. Assuming serialized HKS execution in the hardware accelerator, workload runtime is estimated by multiplying the latency of a single HKS operation by the total number of HKS operations in the workload.

To obtain the total number of HKS operations in the testing workloads, we instrumented OpenFHE to report all basic ring operations of interest and ran the workloads to print out the counts of these operations.

C. Workloads Characteristics

To evaluate the performance of our chip, we used the two CKKS-based workloads identified by DARPA: logistic regression training and CNN inference. We implemented both workloads in OpenFHE: We log the HKS operation counts from the instrumented code for comprehensive analysis. One iteration of logistic regression training includes **83 full HKS operations and 1 bootstrapping operation**, whereas single image inference of CNN requires **196 HKS operations and 3 bootstrapping operations**.

Note that to significantly speed up CNN inference on our hardware, we applied three key optimizations: reducing number of bootstrapping operations, switching to a real-number-friendly variant of CKKS [5], and processing images in batches. Our optimizations are motivated by the capacity of ciphertexts to hold many slots, which would be underutilized without batching.

VI. Results

Table 5 presents a comparison of the performance estimation of their performance on two tasks: Logistic Regression (LogReg) and Convolutional Neural Network (CNN) inference using the GPIO and SerDes hardware designs. The parameters used for comparison are benchmark time and speedup for LogReg, and inference time for CNN. For the LogReg task, SerDes outperforms GPIO significantly. SerDes takes slightly less time at 89 milliseconds and meets the performance metric target. In contrast, GPIO takes a much longer time of 2.67 seconds. The speedup achieved by SerDes is also much higher, with SerDes achieving a 3,146x speedup, compared to GPIO's 109x. For the CNN inference task, SerDes completes the task in about 200 milliseconds, while GPIO takes a significantly longer time of 3.8 seconds. This suggests that for both tasks, the SerDes design demonstrates enhanced performance.

VII. Conclusions

We showed that the TREBUCHET project, under the DARPA MTO Data Privacy for Virtual Environments (DPRIVE) program, has made significant strides in advancing the field of secure computation. By integrating with the OpenFHE library and accelerating its standard Fully Homomorphic Encryption (FHE) schemes, the project has developed a custom hardware accelerator designed to address the high computational overhead of software-only FHE implementations. This has been achieved by leveraging the underlying mathematical transforms used with many parallel ALUs performing vectorized modular arithmetic.

Table 5. Performance estimation of Logistic Regression (LogReg) Training and CNN Inference.

Parameter	Phase 2 Goal	GPIO	SerDes
LogReg benchmark	<= 100 ms	2.67 sec	89 ms
LogReg speedup	5,000x	109x	3,146x
CNN Inference time	<= 250 ms	3.8 sec	200 ms

As the project completes its second phase, it presents promising results on the estimated performance of the system. The target performance is to achieve 4 orders of magnitude in speedup over a single-threaded software-only implementation, significantly closing the computation gap of FHE. This advancement brings us closer to making encrypted processing practical for everyday use, enabling end-to-end encrypted analysis of sensitive data, even if computation is done on untrusted computer resources. This has profound implications for sectors where personally identifiable information (PII) is accessed, such as the Department of Defense (DoD), finance, and healthcare, marking a significant milestone in the journey towards secure computation.

VIII. References

[1] D. Cousins et al. "TREBUCHET: Fully Homomorphic Encryption Accelerator for Deep Computation" GOMACTech 2023, https://arxiv.org/abs/2304.05237

[2] A. Al Badawi et al., "OpenFHE: Open-Source Fully Homomorphic Encryption Library", https://eprint.iacr.org/2022/915

[3] Li, B., Micciancio, D., Schultz, M., & Sorrell, J. (2022, August). Securing approximate homomorphic encryption using differential privacy. In Annual International Cryptology Conference (pp. 560-589). Cham: Springer Nature Switzerland.

[4] D. Soni, et al., "RPU: The Ring Processing Unit," in 2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Raleigh, NC, USA, 2023 pp. 272-282.

[5] Kim, D., & Song, Y. (2019). Approximate homomorphic encryption over the conjugate-invariant ring. In Information Security and Cryptology–ICISC 2018: 21st International Conference, Seoul, South Korea, November 28–30, 2018, Revised Selected Papers 21 (pp. 85-102). Springer International Publishing.