

REED: Chiplet-Based Scalable Hardware Accelerator for Fully Homomorphic Encryption

Aikata Aikata¹, Ahmet Can Mert¹, Sunmin Kwon², Maxim Deryabin², Sujoy Sinha Roy¹
IAIK, Graz University of Technology ¹, Samsung Advanced Institute of Technology, Suwon, Republic of Korea²

{aikata,ahmet.mert,sujoy.sinharoy}@iaik.tugraz.at,{sunmin7.kwon,max.deribin}@samsung.com

Abstract—Fully Homomorphic Encryption (FHE) has emerged as a promising technology for processing encrypted data without the need for decryption. Despite its potential, its practical implementation has faced challenges due to substantial computational overhead. To address this issue, we propose the *first* chiplet-based FHE accelerator design ‘REED’, which enables scalability and offers high throughput, thereby enhancing homomorphic encryption deployment in real-world scenarios. It incorporates well-known wafer yield issues during fabrication which significantly impacts production costs. In contrast to state-of-the-art approaches, we also address data exchange overhead by proposing a non-blocking inter-chiplet communication strategy. We incorporate novel pipelined Number Theoretic Transform and automorphism techniques, leveraging parallelism and providing high throughput.

Experimental results demonstrate that REED 2.5D integrated circuit consumes 177 mm² chip area, 82.5 W average power in 7nm technology, and achieves an impressive speedup of up to 5,982× compared to a CPU (24-core 2×Intel X5690), and 2× better energy efficiency and 50% lower development cost than state-of-the-art ASIC accelerator. To evaluate its practical impact, we are the *first* to benchmark an encrypted deep neural network training. Overall, this work successfully enhances the practicality and deployability of fully homomorphic encryption in real-world scenarios.

I. INTRODUCTION

Data breaches compromising large cloud storage, and jeopardizing millions of private accounts, have become a daily threat [26], [37], [43]. The vulnerability stems from storing data in an unencrypted format, leaving it susceptible to attacks. Even if the server encrypts the data for storage, it needs to decrypt it for processing, exposing it to potential privacy breaches. This is where Fully Homomorphic Encryption (FHE) comes into play. FHE is a promising cryptographic technique that enables secure and privacy-preserving computation, communication, and storage. Servers can compute on homomorphically encrypted data and return encrypted outputs. This approach ensures that every client holds the key to his/her privacy. FHE’s potential utility spans a wide range of applications, including cloud computing, data processing, and machine learning. The concept of Homomorphic Encryption was first presented in 1978 by Rivest, Adleman, and Dertouzos [54], and the first FHE scheme was introduced in 2009 by Craig Gentry [22]. Since then, numerous algorithmic proposals have emerged [6], [11]–[13], [17].

A common limitation shared by these schemes is the significant computation and memory overhead. This results

in a performance degradation of 10,000× to 100,000× [30] compared to unencrypted computation. It can be attributed to the fact that plain data expands into large polynomials during homomorphic computation, and simple operations, like multiplication, translate into complex polynomial operations homomorphically. Consequently, this drawback hinders FHE deployment in real-life scenarios. To bridge this performance gap between plain and homomorphic computations, researchers have proposed acceleration techniques on various platforms, including CPU, GPU, FPGA, and ASIC [1], [4], [18], [19], [21], [29], [32]–[34], [40], [46], [50]–[53], [55]–[57], [61], [64], [67]–[70].

Software implementations offer flexibility but suffer from poor performance. While attempts have been made to bridge this gap with GPU-based solutions [4], [29], they have yet to match the performance of FPGA-based works [1], [40], [53], [70], which not only exhibit superior performance but also provide re-programmability and real-time verifiability. Nonetheless, there is still a need to narrow the runtime gap between plain and homomorphic computations in FPGA-based solutions. Currently, the most notable acceleration results have been achieved through ASIC works. However, it is important to note that in pursuit of maximizing acceleration, several works have deviated from the critical requirement of ensuring real-life deployability. This is mainly due to the monolithic implementation approach adopted by all ASIC-based works in the literature.

In a monolithic design, all components are integrated into a single chip which is relatively easy to design. However, it suffers from various limitations such as inflexibility, low yield, and higher manufacturing costs [23]. Works such as [32]–[34] propose chips with an approximate size of 400mm², resulting in a manufacturing yield of only 67% [39]. They also face high development costs (≈25M\$ [45]) and very long time-to-market (>3 years). In such cases, architecture is excessively large, surpassing manufacturing capabilities and making pre-silicon verification on FPGAs infeasible. These limitations pose a significant obstacle to the fast implementation of homomorphic encryption techniques.

Additionally, these proposals overlook the crucial need for communication-computation parallelism. Off-chip to on-chip communication is considerably slower than the chip’s computation speed. Therefore, while the proposed designs may demonstrate good performance on shallow benchmarks [25],

[59], they are likely to experience significant performance degradation for complex tasks like neural network training. Enhancing the overall efficiency of the design requires leveraging communication-computation parallelism.

Overall, we have observed that research on FHE acceleration has reached a saturation point with limited scalability. The general approach to address this is proposing larger chips. However, this has already reached a stage where manufacturing has become infeasible, making the proposed acceleration unattainable. To overcome this major challenge, we embrace chiplet-based design [24], [39], [72]- a modular approach to building big architectures and offers numerous advantages, including scalability, high yield, low cost, quick pre-silicon verification, and less time-to-market [7], [23]. Since existing designs cannot be efficiently scaled down to chiplet-based designs, we introduce a scalable design methodology that can be easily scaled up or down depending on requirements and constraints. We hope this will open practical possibilities for privacy-preserving computation.

A. Our Contribution

We unfold our major contribution across two dimensions of scalability- configurable design methodology and modular implementation approach. Our contributions are as follows:

- **Scalable hardware design- REED:** We propose a configuration-based ($N_1 \times N_2$) design methodology while incorporating communication bandwidth of N_2 coefficients. All the building blocks under this design methodology offer a thought of f/N_1 operations per second, where f is the design operating frequency. By changing the configuration parameters (N_1, N_2), the architecture can be adapted to the desired area and throughput requirements. Thus, it addresses the various constraints in real-world scenarios, improving utility.
- **Chiplet-based implementation- REED 2.5D:** We take a step back from existing implementation techniques and present a novel and cost-effective chiplet-based FHE implementation approach, which is inherently scalable. REED with 2.5D packaging surpasses state-of-the-art work SHARP₆₄ [32] with $2 \times$ better energy efficiency and $2 \times$ less development cost. To the extent of our knowledge, this is the *first chiplet architecture for accelerating FHE*. We further explore the potential of extending this work by leveraging the promising 3D Integrated Circuit (IC) technology [7].
- **High-performance computation:** We present innovative design techniques for the number-theoretic transform (NTT) and automorphism. Our approach introduces a Hybrid NTT unit that eliminates the need for the expensive transpose operation or scratchpad memory and an easily configurable automorphism unit. These building blocks leverage parallelism and pipelining for high throughput.
- **Communication-computation parallelism:** Chiplet-based architectures may suffer from slow inter-chiplet communication bottleneck. We address this by proposing the *first non-blocking ring-based inter-chiplet*

communication strategy in the context of FHE, ensuring computation-communication parallelism. This is made feasible due to our proposed interleaved data distribution technique, which reduces memory consumption. These optimizations further enhance the overall performance of the hardware acceleration design.

- **Application Benchmark:** We choose parameters to offer high precision and, at the same time, good performance. REED is the *first work to benchmark an encrypted deep neural network (DNN) training*, showcasing practical viability and real-world impact. While CPU (24-core, $2 \times$ Intel Xeon CPU X5690 @ 3.47GHz) requires 29 days to finish it, REED 2.5D takes only 7.7 minutes, a realistic time for an NN training. We also use DNN training to run accuracy/precision experiments and validate our parameter choice.

II. BACKGROUND

Let \mathbb{Z}_Q represent the ring of integers in the range $[0, Q-1]$. $\mathcal{R}_{Q,N} = \mathbb{Z}_Q[x]/(x^N+1)$ refers to polynomial ring containing polynomials of degree at most $N-1$ and coefficients in \mathbb{Z}_Q . A polynomial is denoted as $a \in \mathcal{R}_{Q,N}$. In Residue Number System (RNS) [20] representation, Q is a composite modulus comprising co-prime moduli, $Q = \prod_{i=0}^{L-1} q_i$. Let \mathbf{a} be a vector of residue polynomials, and a^i be the i -th residue polynomial in the vector. We use the ‘mathtt’ font, e.g., \mathbf{c} or \mathbf{sk} , to represent ciphertexts or keys. Operators \cdot and \langle, \rangle denote the multiplication and dot-product between two ring elements. Noise is represented as e and is refreshed for every computation.

A. FHE schemes and HEAAN routines

Multiple FHE schemes exist in literature, including BGV [6], TFHE [13], HEAAN [11], [12], among others. These schemes differ primarily in the types of data they can encode and the supported operations. For instance, BGV can handle integers, while HEAAN can work with fixed-point numbers and is widely adopted for benchmarking machine learning applications [25], [31]. Hence, our accelerator design is in the context of HEAAN [11]. Notably, most schemes also require polynomial computations similar to those in HEAAN. Consequently, our proposed design methodology can be extended to other schemes, like BGV.

We present the HEAAN [11] routines for ciphertexts at level l (multiplicative depth is $l-1$) where $l < L$, $Q_l = \prod_{i=0}^{l-1} q_i$, and L is the maximum level. Please refer to [11] for a detailed description. The first three procedures are computed by the client, and the remaining procedures are evaluated on ciphertexts by the cloud.

1) HEAAN.KeyGen(): This routine generates secret key $\mathbf{sk} = (1, s)$, public key $\mathbf{pk} = (-a \cdot s + e, a) \in \mathcal{R}_{Q_L, N}^2$, and several key-switching keys $\mathbf{ksk}_i = (-a \cdot s + e + P \cdot s', a) \in \mathcal{R}_{PQ_L, N}^2$ for $i \in [0, L)$, where a is uniformly random and s' is a secret. For relinearization, $s' = s^2$.

2) HEAAN.Enc(m, \mathbf{pk}): It encrypts a message m using public key, and returns ciphertext $\mathbf{c} = v \cdot \mathbf{pk} + (m + e, e) \in \mathcal{R}_{Q_L, N}^2$.

TABLE I
HEAAN PARAMETERS

Parameter	Definition	Value
N, n	Polynomial size, Packing slot	$2^{16}, 2^{15}$
Q, q_i	Coefficient modulus and its RNS base	-
L	Number of RNS bases for $Q = \prod_{i=0}^{L-1} q_i$	31
P, p_i	Special modulus and its RNS base	-
K	Number of RNS bases for $P = \prod_{i=0}^{K-1} p_i$	1
w	Word size ($\log p_i, \log q_i$)	54-bit
L_{boot}, L_{eff}	Multiplicative depth of/after bootstrapping	15,15

3) HEAAN.Dec(c, sk): The ciphertext c is decrypted using the secret key sk to return message $m' = \langle c, sk \rangle$.

4) HEAAN.Add(c, c'): It takes two input ciphertexts $c = (c_0, c_1) \in \mathcal{R}_{Q_i, N}^2$ and $c' = (c'_0, c'_1) \in \mathcal{R}_{Q_i, N}^2$ and computes $c_{add} = (d_0, d_1)$ where $d_0 = c_0 + c'_0 \in \mathcal{R}_{Q_i, N}$ and $d_1 = c_1 + c'_1 \in \mathcal{R}_{Q_i, N}$.

5) HEAAN.Mult(c, c'): It multiplies two input ciphertexts $c = (c_0, c_1) \in \mathcal{R}_{Q_i, N}^2$ and $c' = (c'_0, c'_1) \in \mathcal{R}_{Q_i, N}^2$, and computes $d_0 = c_0 \cdot c'_0 \in \mathcal{R}_{Q_i, N}$, $d_1 = c_0 \cdot c'_1 + c_1 \cdot c'_0 \in \mathcal{R}_{Q_i, N}$, and $d_2 = c_1 \cdot c'_1 \in \mathcal{R}_{Q_i, N}$. The output is the non-linear ciphertext $d = (d_0, d_1, d_2) \in \mathcal{R}_{Q_i, N}^3$, which is then linearized using a ‘key-switch’ procedure, as described below.

6) HEAAN.Automorphism(c, rot): The input ciphertext $c = (c_0, c_1) \in \mathcal{R}_{Q_i, N}^2$ is homomorphically rotated by rot using galoi element ($gle = 5^{rot} \bmod 2N$) to return ciphertext $d = (\rho_{rot}(c_0), \rho_{rot}(c_1)) \in \mathcal{R}_{Q_i, N}^2$ after key-switch. A conjugation is a special form of automorphism when ($gle = 2N - 1$).

7) HEAAN.KeySwitch(d, ksk): It performs key-switch, using a relevant key ksk , after HEAAN.Mult/Automorphism so the resultant ciphertext is encrypted under the same secret key as the input ciphertext. It computes $c'' = (c''_0, c''_1)$ where $c''_0 = \sum_{i=0}^{l-1} d_0^i \cdot ksk_0^i \in \mathcal{R}_{PQ_i, N}$ and $c''_1 = \sum_{i=0}^{l-1} d_1^i \cdot ksk_1^i \in \mathcal{R}_{PQ_i, N}$. This is followed by $c = ((d_1, d_2) + (\text{HEAAN.ModDown}(c'')) \in \mathcal{R}_{Q_i, N}^2$. HEAAN.ModDown() scales down the modulus from PQ_i to Q_i .

8) HEAAN.Bootstrap: This routine is designed to refresh the multiplicative depth of ciphertexts. It involves evaluating the decryption operation homomorphically [5], [9], [10]. This is the most computationally expensive routine, and it is not a standalone procedure but rather a combination of the above routines. A certain amount of multiplicative depth (L_{boot}) is consumed during bootstrapping. As a result, the depth of the ciphertext after bootstrapping (L_{eff}) is always lower than the original multiplicative depth L . We closely adhere to the implementation in OpenFHE [2] for benchmarking.

This works uses hardware acceleration to speed up the cloud-side homomorphic procedures. Table I lists the HEAAN parameters with the notation and values we use for our design targeting the 128-bit classical security ($N = 2^{16}, \log PQ = 1728$) [3], [5]. The choice of word-size ($w = 54$ -bit) offers the best balance between performance and precision, as discussed in Section V-A.

B. Number Theoretic Transform (NTT)

NTT is a discrete Fourier transform defined over the ring \mathbb{Z}_q as $\hat{\mathbf{a}}_i = \sum_{j=0}^{N-1} \mathbf{a}_j \omega^{ij}$ for $i \in [0, N)$, where ω is N -th

primitive root of unity. It reduces the complexity of polynomial multiplication from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$ and is excessively utilized in FHE schemes. In a polynomial ring, *negative wrapper convolution* enables reduction-free polynomial multiplication. It requires polynomials to be multiplied with powers of $2N$ -th root of unity, ψ (pre-processing and post-processing). For more details, readers may refer to [58], [62].

C. Monolithic vs Chiplet, and Chiplet packaging techniques

In the context of large Integrated Circuits, authors in [23], [39], [72] discuss the advantages of chiplet-based designs over monolithic designs. The problem with monolithic designs stems from the fact that to keep up with the increasing demand for high performance and functionality, chips need to be scaled up, and advanced technology nodes must be utilized. Manufacturing such big chips reduces the wafer yield as more surface area is exposed to defects per chip and increases the developments cost. Such huge designs take a very long time-to-market, and it is impossible to test and verify them before manufacturing using FPGAs. More factors, such as size limitation and sub-optimal die performance due to overload, contribute to Moore’s law’s slowdown. Hence, there is a shift from these SoC (System on Chip) to SiP (System in Package) [23], as SiP directly addresses these challenges.

Migrating chiplets to advanced technology is easier than an entire monolithic design. In SiP, multiple heterogeneous smaller chiplets can be manufactured separately and later integrated together using various packaging techniques as long as they adhere to common interface standards. This promotes chiplet-reuse, further lowering the development costs, expediting the process, and resulting in a substantial profit margin. The chiplet-packaging techniques can be broadly classified into three main categories: 2D, 2.5D, and 3D [23], [39]. In 2D packaging, different dies are mounted on a substrate, commonly known as a multi-chip module (MCM). It has limitations due to the substrate, resulting in slow die-to-die communication and high power consumption.

To address these limitations, the most reliable technology for integrating chiplets is the silicon interposer, known as 2.5D integration. In this approach, an interposer is placed between the die and the substrate, enabling die-to-die connections on the interposer itself. The use of an interposer significantly enhances interconnectivity, leading to improved performance. Several studies in the literature, such as [49], [73], demonstrate the practicality of this approach. Taking the integration capabilities a step further, 3D packaging involves stacking different dies on top of each other, akin to a skyscraper. In 3D packaging, the dies are interconnected using through-silicon vias (TSVs). 3DIC is gaining significant popularity and serves as the foundation for advancements [8], [48], [65], [66]. A well-known example of 3D packaging is the High Bandwidth Memory (HBM/HBM2/HBM3), where multiple DRAM dies are stacked. This approach significantly reduces the critical path and area, resulting in higher performance, lower power consumption, and increased bandwidth. The slowdown of Moore’s law finds hope in 2.5D and 3D IC.

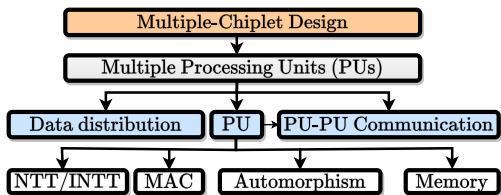


Fig. 1. Design hierarchy for chiptlet-based HE accelerator.

III. THE SCALABLE ARCHITECTURE DESIGN METHODOLOGY FOR REED

Our primary objective is to propose a scalable design methodology that can be utilized by chiptlet-based accelerators to offer superior acceleration. A hardware accelerator design for FHE schemes has three fundamental computation units: Number Theoretic Transform (NTT/INTT), Multiply-and-Accumulate (MAC), and Automorphism. All homomorphic operations can be computed by utilising these units. The computational building blocks are integrated with memory components to create a complete Processing Unit (PU). To achieve a transition from a single PU to multiple PUs and multiple chiplets, we also need to address PU-PU communication and data distribution. This overall design flow for our accelerator design- REED, is illustrated in Fig. 1.

We will present our design methodology starting with the top of the hierarchy- Multi-Chiplet Design. To comprehend the decision-making process behind the middle modules, it is crucial to grasp the design principles employed for the bottom-most modules: NTT, MAC, and Automorphism. Hence, next, we will thoroughly discuss the scalable design of these modules. Subsequently, we will demonstrate how these modules are integrated to form a complete PU, ensuring optimal performance and efficiency. Lastly, we will showcase our efficient data distribution and PU-PU/C2C (chiptlet-to-chiptlet) communication strategies. They enable seamless data exchange, leading to better scalability and acceleration.

A. Multi-Chiplet Design

We presented the advantages of disintegrated systems over monolithic designs in Section II-C. The transition from 2D monolithic packaging to 2.5D or 3D disintegrated chiptlet systems represents both the present and future of architectural designs, as emphasized in [23], [24], [28], [39], [66], [72], [73]. In this context, we present REED 2.5D and RE3D.

1) *REED 2.5D*: We first present a sample two-chiptlet design, depicted in Fig. 2 (a). Here we connect two REED chiptlets and establish connections between PU and HBMs via the interposer. Due to the proposed ring-based communication (Section III-E), scaling this design only increases the interconnects linearly. Hence, we can scale it to four chiptlets, as shown in Fig. 2 (b). We ensure through our FHE design that no HBM-HBM communication is required. Hence, they are positioned on the outer side. Moreover, we avoid sharing a single HBM among multiple chiptlets, ensuring that each HBM is located only in proximity to the one chiptlet it serves.

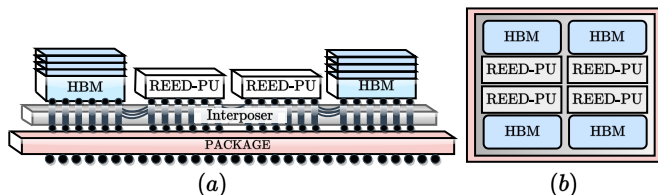


Fig. 2. (a) Side view of two chiptlet-based REED 2.5D, and (b) top view of four chiptlet-based REED 2.5D.

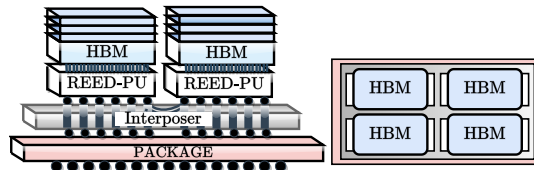


Fig. 3. The side and top view of proposed RE3D. It has four REED 3D IC chiptlets interconnected using the die-to-die link via silicon interposer.

Given that die-to-die communication requires a simple ring-like communication pattern (discussed in Section III-E), the chiptlets are placed in a relatively straightforward manner, as not all dies need to communicate with every other die. In [49], [73], authors propose general-purpose chiptlet-based processors with an actual tapeout. Our placements strategies align with these, demonstrating practical viability. We acknowledge the potential latency issues arising from slow chiptlet-to-chiptlet communication, and this will be addressed in Section III-E.

2) *RE3D: REED's journey from 2.5D to 3D*: After discussing the design for REED 2.5D, we present its extension to a complete 3D IC structure, which holds immense potential for future computing. To achieve this transition, we have two options: connecting the PU with the HBM controller via TSV (as shown in Fig. 3) or merging the PU unit with the lower HBM controller die. By adopting either of these approaches, we can significantly reduce the reliance on the Network-on-Chip (NoC), leading to a compact chip design with lower power consumption. Each chiptlet is a full 3D IC package (PU and Memory) and needs a die-to-die link via interposer for connecting to other chiptlets. The reduction in the area primarily comes from fewer HBM stacks on the lateral area and the integration of the REED-PU unit with the HBM controller. Additionally, the decrease in critical paths due to the reduced interconnects would enhance the design's performance. Thus, RE3D would further bridge the gap between speedup and privacy.

3) *Disintegration Granularity*: It is crucial to note that disintegrated systems face a trade-off between development cost and performance degradation, depending on the disintegration granularity. Existing works, such as [24], [39], [65], [66], [72], [73], show that disintegration improves yield, but it introduces challenges such as floorplanning and post-silicon testing overhead. Since this design is used for accelerating FHE, its full utilization in the long run also weighs in. Hence, we need to address the question: *How much disintegration*

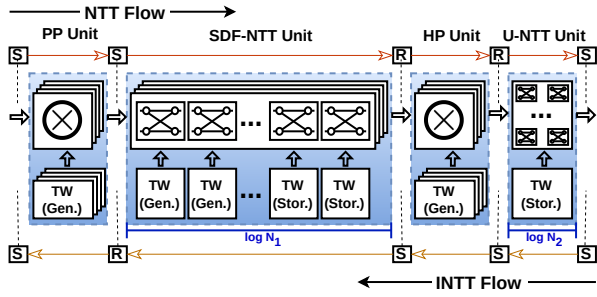


Fig. 4. The proposed novel Hybrid NTT/INTT design flow for $N = N_1 \times N_2$.

is best for our design? Considering a complete die area of 800mm^2 , dividing it into four chiplets offers an $\approx 80\%$ yield, while eight chiplets provide a yield of $\approx 90\%$. Although the eight-chiplet option seems promising, it comes at the cost of additional complexities in floorplanning/routing, testing, and power consumption. In the context of FHE, as the multiplicative depth decreases, we need fewer PUs (discussed in Section III-D). Hence, employing four larger chiplets offers longer utilization compared to eight smaller chiplets, assuming an equal number of PUs per chiplet.

In conclusion, instantiating four chiplets strikes the perfect balance between manufacturing cost and utilization. Next, we will discuss the design of building blocks for REED-PU and see how they help us achieve our acceleration goals.

B. The ingredients of REED Processing Unit

The need for scalability and high throughput drives our design methodology. Before delving into the details of the building blocks, we introduce the REED-configuration (N_1, N_2) for polynomial degree N , where $N_1 \times N_2 = N$, and both N_1 and N_2 are powers of two. This configuration provides a throughput of $\frac{f}{N_1}$ operations per second and can process N_2 coefficients simultaneously, where f is the design's operating frequency. Hence, requiring a memory read/write bandwidth of N_2 . The proposed standard configuration brings forth advantages, such as improved throughput and efficient resource utilization. Mapping all the building blocks to this configuration enhances scalability, enabling easy scale-up or scale-down. Now, let us explore how we design the building blocks to match this configuration and fully exploit its potential.

1) *The Hybrid NTT/INTT (Frankenstein's approach)*: This unit plays a vital role in converting polynomials from slot to coefficient representation and vice versa. It is computationally extensive and occupies over 50% architectural area. Therefore, designing an efficient NTT/INTT unit is crucial as it directly impacts the overall throughput and area-consumption.

There are various approaches in the literature to implement NTT for large-degree polynomials, such as iterative [1], [40], [58], pipelined [71], [74] and hierarchical [19]. While these approaches can offer efficient designs for specific configurations or target platforms, they all suffer from implementation complexity and lack of scalability for large polynomial sizes. Additionally, these approaches rely on scratchpad-like mem-

Algorithm 1 Hybrid NTT with NWC

In: a (a matrix of size $N_1 \times N_2$ in row-major order)
In: ω (N -th root of unity), ψ ($2N$ -th root of unity)
Out: $a = \text{NTT}(a)$ (a matrix of size $N_1 \times N_2$ in column-major order)

- 1: **for** ($i = 0; i < N_1; i = i + 1$) **do**
- 2: **for** ($j = 0; j < N_2; j = j + 1$) **do**
- 3: $a[i][j] \leftarrow a[i][j] \cdot \psi^{i \cdot N_2 + j}$ (mod q) \triangleright Pre-processing (PP)
- 4: **end for**
- 5: **end for**
- 6: Apply N_1 -pt NTT to the columns of a \triangleright using SDF-NTT
- 7: **for** ($i = 0; i < N_1; i = i + 1$) **do**
- 8: **for** ($j = 0; j < N_2; j = j + 1$) **do**
- 9: $a[i][j] \leftarrow a[i][j] \cdot \omega^{i \cdot j}$ (mod q) \triangleright Hadamard product (HP)
- 10: **end for**
- 11: **end for**
- 12: Apply N_2 -pt NTT to the rows of a \triangleright using Unrolled-NTT (U-NTT)
- 13: **return** a

ories, which can serve as prefetch units for other building blocks. The iterative approach enables using multiple processing elements to improve the performance of NTT; however, its implementation complexity increases significantly with the number of processing elements. The pipelined approach (also referred to as single-path delay feedback (SDF)) provides a bandwidth-efficient solution but a diminished performance.

The hierarchical approach (also referred to as four-step NTT), utilized in [19], treats a polynomial of size N as an $N = N_1 \times N_2$ matrix and divides a large NTT into smaller parts. It involves performing N_1 -point NTTs on the N_2 columns of the matrix, then multiplying each coefficient by $\omega^{i \cdot j}$ (where i and j are matrix row and column indices), transposing the matrix, and finally performing N_2 -point NTTs on the N_1 columns. Transposing a matrix of size $N_1 \times N_2$ requires N_1 separate memories and large data re-ordering units. For example, in [19], the transpose unit consumes 14% of the area per compute cluster. Moreover, in terms of time inefficiency, it will require additional N_2 cycles for writing data to the transpose memory and N_1 cycles for reading it.

Although the hierarchical approach simplifies the NTT implementation, it has the following limitations: (i) it requires a costly transpose operation, (ii) N_1 and N_2 are fixed to $N_1 = N_2$ [19], [21], hence offering limited flexibility, and (iii) the reliance on scratchpad leads to large memory fan-in and fan-out, causing routing inefficiencies. We address these challenges by introducing a novel Hybrid NTT using Frankenstein's approach, which utilizes parts of hierarchical, iterative, pipelined, and plain unrolled NTTs.

The proposed NTT/INTT unit is fully pipelined, and its flow is shown in Algorithm 1 and Fig. 4. During the NTT operation, we first perform pre-processing (Step 3 Algorithm 1) using N_2 modular multipliers (PP). The resulting coefficients are sent to N_2 pipelined NTT units (N_1 -pt SDF-NTT) to perform Step 6. The output coefficients of SDF-NTT units are processed via the Hadamard Product unit (HP) that multiplies the coefficients with powers of ω (Step 9) using N_2 modular multipliers. Finally, we employ a N_2 -pt unrolled NTT (U-NTT) unit.

Transpose elimination: The Hybrid NTT eliminates transpose by using two orthogonal NTT approaches, pipelined (SDF) approach for N_1 -sized NTTs and unrolled (U-NTT) approach

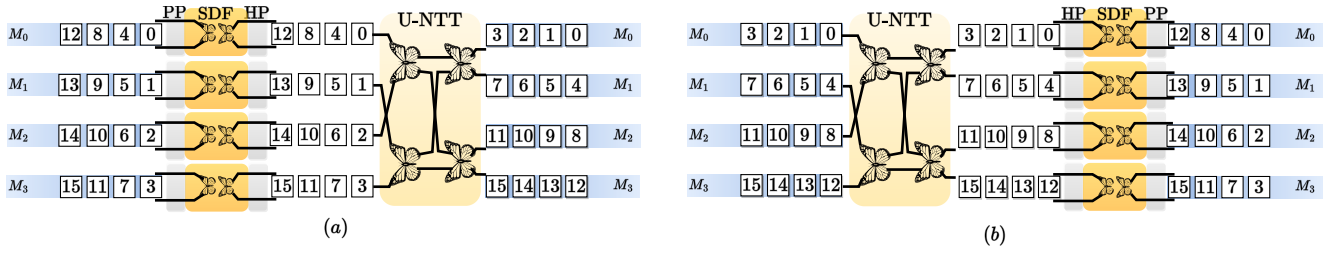


Fig. 5. The proposed novel Hybrid NTT/INTT design flow with Memory access for (a) NTT and (b) INTT with $N_1 = 4$, $N_2 = 4$, and $N = 16$. The butterflies represent the Gentleman-Sande butterfly [58] operation employed in our design.

Algorithm 2 Word-level Montgomery Modular Reduction

In: $d = a \cdot b$, $q = 2^{w-1} + q_H \cdot 2^m + 1$
In: m (Mont. red. size), $L = \lceil \frac{\log_2 q}{m} \rceil$ (number of reduction steps)
Out: $c = a \cdot b \cdot R^{-1} \pmod{q}$, $R = 2^{mL}$

- 1: $T \leftarrow d$
- 2: **for** ($i = 0; i < L; i = i + 1$) **do**
- 3: $T_H, T_L \leftarrow T \ggg m, T \pmod{2^m}$
- 4: $T_2 \leftarrow -T_L \pmod{2^m}$, $cin \leftarrow T_2[m-1] \vee T_L[m-1]$
- 5: $T \leftarrow (q_H \cdot T_2) + T_H + cin + (T_2 \lll (w-1-m))$
- 6: **end for**
- 7: **return** $c \leftarrow (T \geq q) ? T - q : T$

Algorithm 3 Automorphism

In: $a[N_1][N_2]$, gle
Out: $\hat{a} = \rho(a)$

- 1: $index \leftarrow gle$
- 2: **for** ($l_0 = 0; l_0 < N_1; l_0 = l_0 + 1$) **do**
- 3: $l_1 \leftarrow index \pmod{\log(N_1)}$
- 4: $start \leftarrow index \ggg \log(N_1)$
- 5: $addr[j] \leftarrow (start + j \cdot gle) \pmod{\log(N_2)} \forall j \in [0, N_2]$
- 6: $\hat{a}[l_1] \leftarrow shuffle_tree_{N_2 \times N_2}(addr, a[l_0])$
- 7: $index \leftarrow index + gle$
- 8: **end for**
- 9: **return** $\hat{a}[N_1][N_2]$

for N_2 -sized NTTs. The input polynomials for the NTT/INTT operation are stored in N_2 memories of depth N_1 . As shown in Figure 5 (a), the output coefficients of SDF-NTT are processed directly by U-NTT, providing a seamless, natural transpose operation. It also helps make our NTT unit bi-directional, as illustrated in Figure 5 (b).

Low-level optimizations: For modular multiplication and reduction unit, we adopted the word-level Montgomery [41], [42] modular reduction algorithm and optimized it (Algorithm 2) for our special prime form, $2^{w-1} + q_H \cdot 2^m + 1$, where m is Montgomery reduction size, and $\lceil \log_2 q_H \rceil$ is small. For our design, we use $w = 54$, $m = 18$ and $\lceil \log_2 q_H \rceil = 10$. To reduce the on-chip twiddle factor memory requirement, we employ on-the-fly twiddle factor generation using a small constant memory that stores a few initial constants. By utilizing this, we reduce the on-chip constant storage by up to 98.3%.

In summary, the proposed Hybrid NTT/INTT design offers a throughput of $\frac{f}{N_1}$ operations per second and can be scaled for various area/performance trade-offs by adjusting the values of N_1 and N_2 . It eliminates the expensive transpose operation, simplifies routing, and enhances pipelining.

2) *Multiply-and-Accumulate (MAC)*: MAC is a linear unit, and by instantiating N_2 MACs for configuration (N_1, N_2) , we achieve the desired throughput $\frac{f}{N_1}$. Our *triadic* units are capable of performing multiplication and addition/subtraction simultaneously, which is advantageous for the key-switch operation (discussed in Section III-C). It employs the same modular multiplication unit utilized by the NTT/INTT unit.

3) *Automorphism/Conjugation*: This unit permutes ciphertexts using the Galois element (gle) to achieve rotation or conjugation. We note three important properties in automorphism- (i) all N_2 coefficients come and go to N_2 distinct memories, (ii) they are read and written to the same address, and (iii)

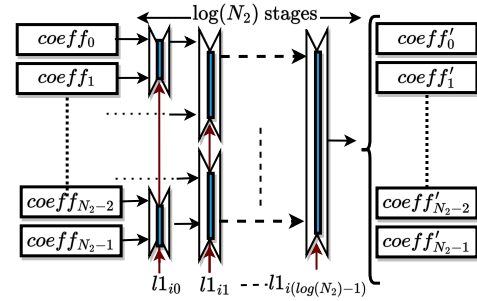


Fig. 6. An example of a $shuffle_tree_{N_2 \times N_2}$ workflow. Every stage here has sufficient registers to hold the N_2 coefficients.

the coefficients move in pairs. To understand this, let us take a brief look at how automorphism works.

A polynomial is stored as a matrix $N_1 \times N_2$ in N_2 memories. When we load N_2 coefficients from memory address l_0 across all N_2 memories, they are shuffled using ρ_{rot} and then written to address l_1 across all N_2 memories. Hence, even though the coefficient order is shuffled, they all go to the same address of N_2 distinct memories. We utilize this property to permute all N_2 coefficients in parallel. This out-of-place automorphism is presented in Algorithm 3. The in-place permutation techniques proposed in previous works [19], [57] increase routing complexity due to memory transposition requirements.

At the end of this, we are still left with a quadratically complex and expensive shuffle among $N_2 \times N_2$ coefficients. As mentioned above, we observed that all the shuffles could be performed pairwise on the coefficient batches, as shown in Fig. 6. After each stage, two batches of coefficients are merged to form a new batch. This helps us replace the naive and expensive operation with a simplified and pipelined binary-

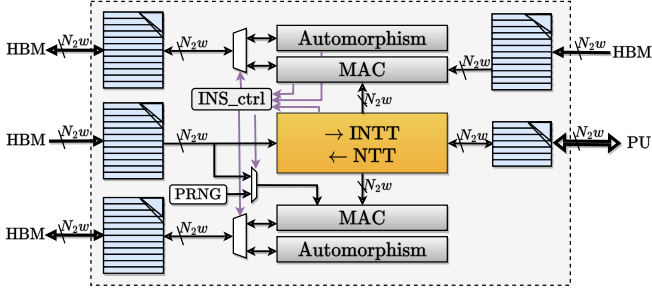


Fig. 7. The REED-PU design. Every data communication (memory to building blocks and off-chip to on-chip) here has a bandwidth of N_2w .

tree-like shuffle. Note that the number of pipeline stages adjusts with N_2 , making the unit scalable and efficient for higher configurations. Moreover, the unit can handle any arbitrary rotation and provide a throughput of $\frac{f}{N_1}$.

We now use these building block designs to design a complete processing unit.

C. Packed REED Processing Unit (PU)

We initiate the PU design (as shown in Fig. 7) by placing the NTT/INTT unit first and providing separate memories for both ends. This ensures straightforward routing and efficient PU-PU communication (Section III-C1). Although the NTT/INTT unit operates on one polynomial at a time, the result is multiplied with two polynomials (key-switching keys) and accumulated. Hence, we instantiate a pair of MAC units capable of simultaneously processing both key components. Similarly, we include two automorphism units as well. A PRNG is deployed to generate the public key component on-the-fly, following the approach in [40]. The design operates based on instructions, wherein a relatively small instruction controller manages the multiplexers and collects ‘done’ signals from these units. Our design choices ensure that the NTT/INTT and MAC/automorphism units can run concurrently in the pipeline.

Among all the routines, the key-switch is the most expensive operation. In this, we transform all L residue polynomials from slot to coefficient representation (INTT), and then each of these is transformed to $L + 1$ NTTs, multiplied with two key components, and accumulated. This requires L INTTs, $L(L + 1)$ NTTs, and $2L(L + 1)$ MACs, making the throughput of this operation: $\frac{f}{L(1+3(L+1)) \cdot N_1}$. We utilize REED’s parallel processing capability to perform all MAC operations concurrent to the NTT operations (shown in Fig. 8). We can also perform multiplication and accumulation simultaneously. Hence, we save $2L(L + 1)$ clock cycles and increase the throughput to $\frac{f}{L(L+3) \cdot N_1}$, resulting in a 66.7% improvement.

1) *Prefetch-Memory*: Previous works [19], [33], [34] refer to their on-chip memory as scratchpads due to the need for intermediate storage and prefetch functionality during NTT/INTT and automorphism operations [19]. This approach increases routing complexity as multiple modules access these scratchpads, requiring them to be in very close proximity.

Our proposed low-level building blocks mitigate the need for intermediate storage, allowing us to use memory units

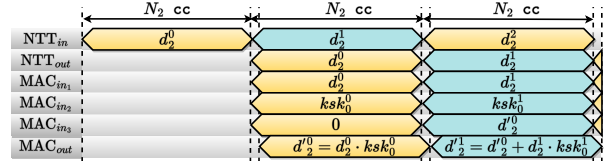


Fig. 8. Timeline demonstrating parallel and pipelined operation flow.

solely as prefetch units. Each memory unit exhibits balanced fan-in and fan-out, and among the five memory units depicted, only four are fed by off-chip memory. The small memory in Figure 7 is responsible for storing and communicating the INTT result to the other PUs (elaborated in Section III-E). Only two of the four memories communicating with off-chip memory need to write back the results, as illustrated by bi-directional arrows in Fig. 7. In total, three memories perform off-chip read/write communication. These memories are physically divided into two parts. When one is utilized for on-chip computation, the other performs off-chip prefetch. This results in a highly streamlined design.

After finalizing REED-PU, we next discuss the optimal data distribution and communication for a multi-PU setting, which lays the foundation for multi-chiplet design.

D. Data distribution and parallel processing for multiple PUs

In a multi-PU setting, data distribution strategies include duplicating data across multiple PUs or utilizing shared memory. These approaches have severe drawbacks, as data duplication requires more storage, and shared memory introduces the risk of deadlocks and needs distributed communication protocols. In our multi-PU design, we focus on ensuring that each PU operates on independent data. Most of the previous works [32]–[34] propose a single monolithic PU and therefore do not require a dedicated study on data distribution. In a multi-PU work- Medha [40], the authors extensively discuss this and propose distributing computation across the RNS bases by employing one PU per RNS base. However, with this approach, as the multiplicative depth decreases, a significant number of PUs become idle, causing underutilization.

Nevertheless, distributing computation across RNS bases enables highly parallel computations. Therefore, we leverage this approach in an r -PU setting, where r is smaller than the number of RNS bases ($r < L + 1$). For data distribution across r -PUs, we utilize an interleaved approach, where the RNS bases of the ciphertexts and keys are distributed among the PUs in an interleaved manner (where, PU_i stores $c_{xj+i} \forall 0 \leq i < r, 0 \leq j < \frac{(L+1)}{r}$), instead of grouping them in a sequential manner (where, PU_i stores c_{xi+j}). This ensures that all PUs are fully utilized in the long run, maximizing the benefits of parallel processing.

It is worth noting that the need for PU-PU communication is inevitable. Distributing data across RNS bases reduces this communication but does not eliminate it. In the following subsection, we will discuss how to handle this.

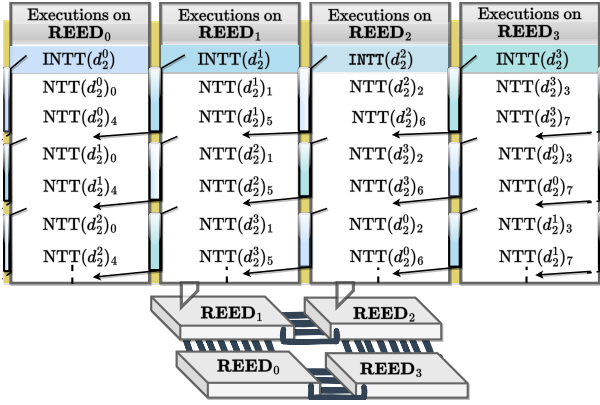


Fig. 9. Non-blocking ring-based communication for four REED chiplets when $L = 7$. The blocks between chiplets represent the long communication window to make up for slow inter-chiplet (C2C) communication.

E. Efficient non-blocking REED-REED communication

Before delving into the solution, let us discuss the need for data exchange across PUs. This is required during the key-switch routine for linearizing, non-linear ciphertext polynomial (d_2) after multiplication. Here an $\mathcal{O}(L^2)$ base conversion is done to switch the modulus of each L residue polynomial of d_2 ($\text{INTT}(d_{2q_i})_{q_i}, \forall 0 \leq i < L$) to $(L+1)$ residues polynomials ($\text{NTT}(d_{2q_j})_{q_j}, \forall 0 \leq j \leq L$) (discussed in Section III-C). The data volume is substantial, and broadcasting each INTT result to all PUs will require a fully-connected communication network among PUs. As the number of PUs (r) increases, this becomes quadratically complex and expensive. When multiple PUs are instantiated in a disintegrated SiP, slow C2C communication becomes a bottleneck.

Instead, we propose an alternative approach illustrated in Fig. 9. Here, we communicate the INTT results to the PUs in parallel with the NTT computations. This approach offers a long communication window for data send/receive, as depicted by the large rectangles between the REED PUs in the figure, which is only made possible due to the proposed interleaved data distribution. Therefore, in cases where C2C communication is slower than computation, this extended communication window prevents PUs/chiplets from experiencing data starvation. Consequently, non-blocking communication is achieved as data computation can proceed concurrently with relatively slower communication. Additionally, note that the communication shown in the figure is only uni-directional. For example, the REED₀ only needs to send data to REED₃ and receive data from the previous REED₁. This enables a simple *ring-based communication* among REED chiplets.

In conclusion, our ring-based communication strategy requires only one read/write port per chiplet, as opposed to $(r - 1)$ ports in a star-like communication network. Furthermore, we address the practical possibility of slower C2C communication by providing a prolonged communication window. Lastly, with our approach, there is no possibility of deadlocks.

Next, we present the area and performance results for 4-chiplet REED 2.5D with one PU per chiplet.

TABLE II
TOTAL AREA CONSUMPTION OF 4-CHIPLET REED 2.5D FOR DIFFERENT CONFIGURATIONS ON 28NM AND 7NM.

Building blocks ↓ Configuration →	28nm (mm ²)		7nm (mm ²)	
	1024×64	512×128	1024×64	512×128
REED	74.9	115	24	43.9
└ REED-PU	58.0	81.0	7.01	9.9
└ NTT/INTT	38.2	56.8	5.61	7.9
└ 2×MAC	3.1	6.6	0.42	0.76
└ PRNG	0.15	0.28	0.02	0.04
└ 2×Automorphism	0.14	0.32	0.02	0.04
└ On-chip memory	16.1	16.1	1.2	1.2
└ HBM3 PHY+NoC	16.9	33.8	16.9	33.8
4×REED	299.6	460	96	175.6
C2C (REED-REED)	12.32	14.64	0.8	1.6
Total Area	311.9	484.6	96.7	177

IV. REED'S IMPLEMENTATION RESULTS

We synthesize our chiplet-based design, REED 2.5D, for configurations 1024×64 and 512×128. For synthesis, we employ TSMC 28nm and ASAP7 [14] 7nm ASIC libraries, with Cadence Genus 2019.11, and use SRAMs for on-chip memories. Our primary objective is to achieve high performance while optimizing area and power consumption. To this end, we set our clock frequency target to 1.5 GHz, use High-vt cells (hvt) configuration for low leakage power, enable clock-gating, and set the optimization efforts to high. We set the input/output delays to 20% of the target clock period and leverage incremental synthesis optimization features.

As off-chip storage, we leverage the state-of-the-art HBM3 [27], [36], [48] memory. Owing to its improved performance and reduced power, it is already deployed in various GPUs and CPUs [15]. It is a 3D IC memory with the memory controller as the bottom layer, and DRAM dies stacked on top of it. HBM3 with 8/12 stacks of 32Gb DRAMs has 32/48 GB storage capacity [27], [44], which is sufficient to store all the key-switching keys. The ciphertexts provided by the client can be transferred to REED using 32 lanes PCIe5 offering a bandwidth of 128 GB/s [63]. The slow communication overhead can be easily masked with computations. In our work, we present results for HBM3 PHY and HBM3 NoC, based on [8], [48] and consider the minimum reported bandwidth of 896GB/s [48]. Some recent studies [15], [48] have reported significantly higher bandwidths of 1.15/3 TB/s. By leveraging these higher bandwidths, our area and power consumption will reduce due to fewer memory requirements.

Table II presents the area results for the REED 2.5D architecture, featuring a 4-chiplet configuration as illustrated in Fig. 10. Configuration 512×128 requires twice the amount of HBM3 compared to the 1024×64 configuration due to the doubled bandwidth requirement. We implement the inner REED-PU, NoC, and HBM3 (shown in Fig. 10) as one chiplet (similar to [49], [73]). In Table III, we present the performance of FHE routines for both configurations with the achieved target clock frequency of 1.5 GHz.

Moreover, we take a step further by prototyping the essential building blocks on Xilinx Alveo U250 to verify functional correctness. It is worth noting that the monolithic designs

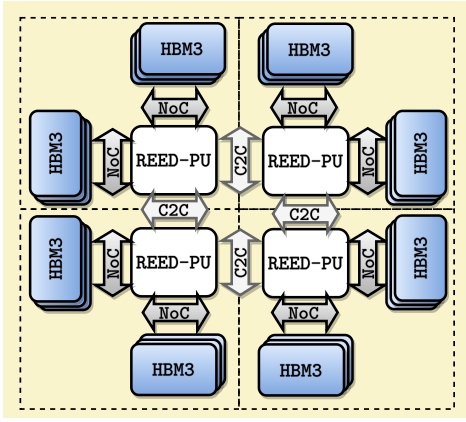


Fig. 10. The complete architecture diagram of 4-chiplet REED 2.5D for 512×128 configuration.

TABLE III
PERFORMANCE MICRO-BENCHMARKS FOR 28NM AND 7NM.

Micro-benchmarks ↓ Configuration →	Level	28nm/7nm (ms)	
	I	1024×64	512×128
ADD/MULT (pt-ct)	31	0.005	0.003
MULT (ct-ct)	31	0.01	0.005
MAC	31	0.01	0.005
Automorphism	31	0.005	0.003
KeySwitch	31→32	0.19	0.08
MULT & Relin.	31→32→31→30	0.22	0.11
Bootstrapping	1→31→16	14.2	7.1

proposed in the literature have excessive size [19], [32]–[34], [57], rendering them unsuitable for pre-silicon verification on FPGA for functionality testing. However, we have successfully overcome this limitation by adopting a chiplet-based implementation strategy and fully leveraging its capabilities. The run-time and power consumption estimates are obtained using a cycle-accurate simulated model.

When we extend REED 2.5D to a 3D IC, one might wonder how to stack the second HBM3 on top of the REED die. In Fig. 10, we include two stacks to achieve the required bandwidth (1.8TB/s), constrained by the interposer and substrate technology. However, 3D IC technology enables direct TSV connections from the chip’s surface to HBM3, enabling the construction of wider buses for higher bandwidth [66]. As per the results on 7nm technology for REED 2.5D (with one HBM3 stack), the REED-PU and NoC account for less than 50% of the area. Hence, by implementing the HBM3 controller on top of it, the lateral surface area would be reduced by $\approx 50\%$. Although this would not directly impact the chiplet manufacturing cost, as the monolithic 3D IC testing and integration costs tend to be higher, it would significantly reduce the cost of the underlying layers, such as the silicon interposer and substrate/package required for using this in a chiplet setting. Thus, the 3D IC integration of REED promises a huge reduction in overall chip area and power consumption. These findings validate our approach’s efficacy, and we believe it will inspire further research in this direction.

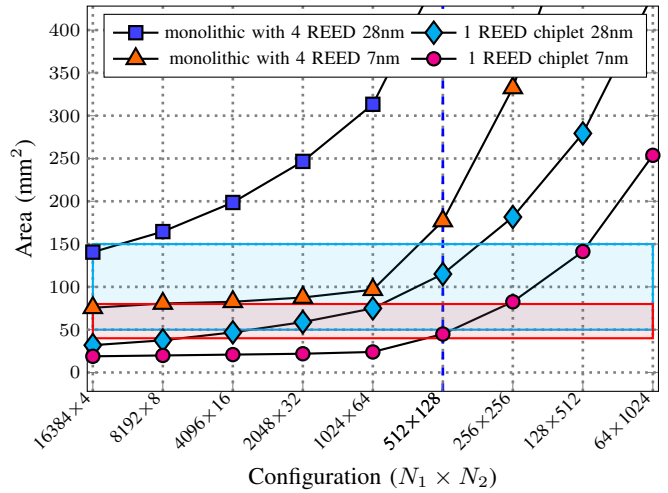


Fig. 11. Demonstration of increase in area with REED configurations, put in the order of increasing throughput [24].

A. What to expect from higher-throughput configurations?

Until now, we have examined two configurations (1024×64 and 512×128) that only partially demonstrate the advantages of our proposed scalable design methodology. As we double the throughput (by doubling the value of N_2), the area of PU only increases by approximately $1.5 \times$. This trade-off arises from the chip area comprising two components—(i) the computation logic area, which scales linearly with throughput, and (ii) on-chip storage that remains fixed to a number of polynomials. As we opt for a higher configuration, the polynomial-size remains the same while the number of coefficients to be processed increases.

However, an important question remains: *what configuration strikes the best balance between throughput and manufacturing cost?* To address this, we turn to [24], where the authors discuss that for 7nm technology, the optimal manufacturing size ranges from 40 to 80 mm^2 , while for 40nm, it ranges from 50 to 150 mm^2 . In Fig. 11, we present two sets of area consumption results for 28nm and 7nm technologies. The first set corresponds to four REED cores produced as a single monolithic chip, while the second set represents one REED chiplet. The optimal area ranges are highlighted in blue and pink. As we can see, for both 7nm and 28nm, the configuration 512×128 falls within the most optimal development area range and offers high throughput. Monolithic designs, within the optimal range, offer 4 to 8 \times less throughput.

B. Comparison with related work

The realization of privacy-preserving computation through FHE holds great potential for the entire community, leading to various research efforts in the literature. These endeavours span from efficient software implementation libraries [2], [16], [60] to ASIC chip proposals [19], [21], [32]–[34], [57]. Among these, the ASIC designs [19], [32]–[34], [57] have achieved the most promising acceleration results. However, their primary drawback lies in the high manufacturing costs associated with

TABLE IV
COMPARISON OF REED 2.5D WITH STATE-OF-THE-ART.

Work	Area (mm ²)	T _{A.S.} [‡] (ns)	P _{Avg} (W)	EDAP _w (M)	EDAP _{w,w²} (/M)
F1 ₃₂	71.02 [†]	470	28.5 [†]	754.5	847
BTS ₆₄	373.6	45.4	163.2	106.0	87.8
ARK ₆₄	418.3	14.3	135	9.74	9.6
CLake ₂₈	222.7 [†]	17.6	124 [†]	16.5	17.7
SH ₃₆	178.8	12.8	94.7	4.1	5.2
SH ₆₄	325.4	11.7	187	7.0	6.2
REED₅₄	177	14.4	83.5	3.1	3.1

[†] Area and power are normalized (14nm/12nm to 7nm) to draw fair comparisons [32], [47]. The word size is given as subscript with the works.
[‡] Amortized time for bootstrapping.

the large monolithic chips, resulting in low yield and further exacerbating the cost. Therefore, our work focuses on reducing manufacturing costs while pursuing accelerated performance.

Table IV and Fig. 12 showcase the successful achievement of our goals. The table compares our design’s area consumption, performance, and power consumption for the packed bootstrapping operation (OpenFHE [2]) with existing works-F1 [19], BTS [34], ARK [33], CraterLake (CLake) [57], and SHARP (SH) [32]. Note that all these works propose monolithic chips and suffer the drawback pertaining to monolithic designs. We utilize the results obtained for 4-chiplet REED 2.5D on 7nm technology as these works also provide results for this specific technology. Several normalizing metrics exist in the literature for comparison, such as the amortized time T_{A.S.} [1], [33], [34] that calculates the bootstrapping time divided by L_{eff} and packing n . However, this metric overlooks factors such as area, power, and precision. Hence, we use EDAP (Energy-Delay-Area product) metric [38] and modify it to accommodate the trade-off of high precision necessary for large applications (discussed in Section V-A).

Higher precision necessitates a larger word size, w . This has a linear impact on some components and a quadratic on others. Our first proposed metric, EDAP_w (Eq. 1), incorporates a linear increase due to word size. It is important to note that the area of the REED-PU increases quadratically with w due to the presence of multipliers. This is addressed in the second metric, EDAP_{w,w²} (Eq. 2), with $w = 54$ as the baseline. Under this metric, we achieve 2×, 1.7× better results compared to the state-of-the-art work SHARP₆₄, SHARP₃₆ [32].

$$EDAP_w = \frac{E \cdot D \cdot \text{Area} \cdot 54}{w} \quad (1)$$

$$EDAP_{w,w^2} = \frac{E \cdot D \cdot \text{Area}_{PU} \cdot 54^2}{w^2} + \frac{E \cdot D \cdot \text{Area}_{Mem} \cdot 54}{w} \quad (2)$$

We also assess the yield and manufacturing cost, as depicted in Fig. 12. For this, we use the original area and not the word-size scaled area. We plot the relative yield [39] and manufacturing cost [24], [45], using our work on 7nm technology as the baseline. As observed, we achieve the highest yield and lowest manufacturing cost for 7nm, resulting in the least overall cost (manufacturing cost/yield), 50% less than state-

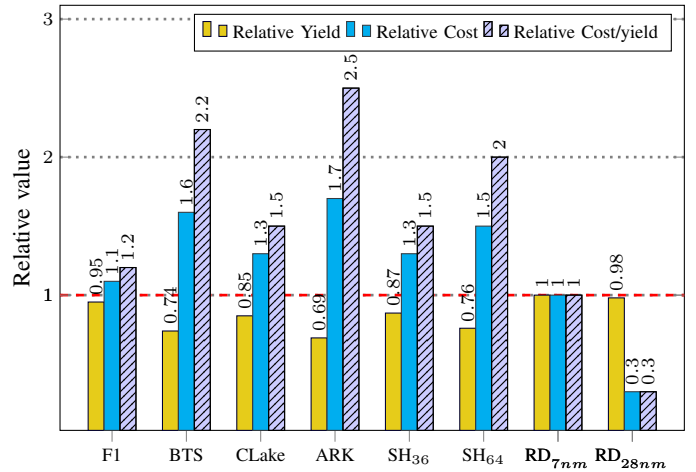


Fig. 12. Relative a) yield of existing monolithic designs versus the proposed 7nm chiplet-based architecture [39], b) development cost (including Interposer cost) [24], [40], [45], and c) cost of SiP development (cost/yield). RD refers to our work REED 2.5D.

TABLE V

A BRIEF OVERVIEW OF THE APPLICATIONS AND THE SPEEDUP ACHIEVED BY OUR PROPOSED REED 2.5D. THE CPU SPEED IS REPORTED BY RUNNING THE APPLICATIONS USING OPENFHE [2] ON A 24-CORE, 2×INTEL XEON CPU X5690 @ 3.47GHZ WITH 192GB DDR3 RAM.

Appl.	Accuracy	Op	CPU-time	HW-time	Speedup
Lin.Reg.	78.12%	Inf.	0.86 s	0.15 ms	5,673×
		Trn.	13.82 s	2.3 ms	5,982×
Log.Reg.	61.8%	Inf.	1.27 s	0.23 ms	5,570×
		Trn.	11.18 s	1.9 ms	5,730×
DNN	95.2%	Inf.	128.7 s	24.3 ms	5,292×
		Trn.	29 days	460 s	5,450×

of-the-art monolithic design SHARP₆₄. On 28nm technology, we achieve 85% cheaper design compared to SHARP₆₄.

In the next section, we will report the application benchmarks and discuss the importance of precision.

V. APPLICATION BENCHMARKS

We benchmark three machine learning applications: linear regression, logistic regression, and a Deep Neural Network (DNN). The speedup results are presented in Table V. Each application is evaluated for *encrypted* training and inference. In this setting, the server provides computational support without knowledge of the data or model parameters, ensuring complete blind computation. Most applications benchmarked in the previous works [19] are partially blind; the server does not see the data but knows the model parameters to evaluate it. To our knowledge, none of the previous works benchmark an encrypted neural network training.

1) *Linear Regression*: We employ the Kaggle Insurance dataset [59] to benchmark linear regression. The model uses a batch size of 1204 and 1338 input feature vectors (each containing six features) for training and inference and achieves an accuracy of 78.1% (same as plain model [59]), as it does not require any approximation and completes training in just two iterations (forward-backward_{×2}-forward).

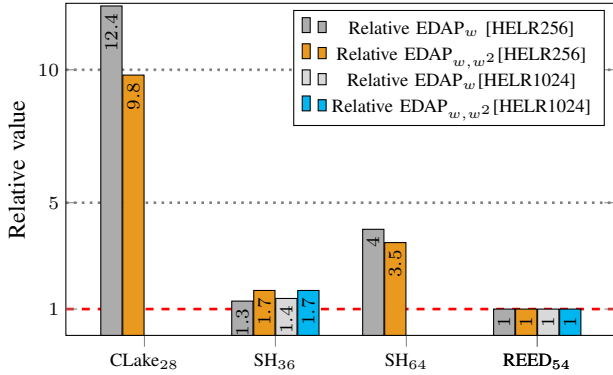


Fig. 13. Relative metrics comparison for the HELR [25] application with batch sizes 256 and 1024. Under these metrics, the lower the value, the better.

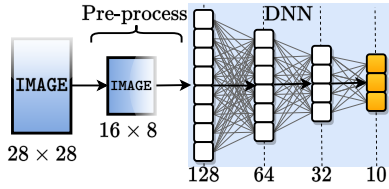


Fig. 14. A DNN for MNIST [35] with two hidden and one output layers.

2) *Logistic Regression*: It is a supervised machine learning model that reports the probability of an event using the logistic function, evaluated using function approximations in a homomorphic context. Their accuracy depends on the degree of approximation function expansion and precision. Existing works, such as [32], [57], utilize the HELR [25] application to benchmark encrypted training on MNIST [35] data, with varying batch sizes (256, 1024). In Fig. 13, we illustrate the superior performance of REED 2.5D compared to these works.

We also evaluate logistic regression on the iDASH2017 cancer dataset (similar to [31]) employed to predict cancer probability. Utilizing the same expansion as [25], we achieve a training accuracy of 62% in just a single iteration. The competition winner [31] reports a slightly higher accuracy of 62.36%. This dataset comprises 18 features per input, with batch sizes of 1422 and 1579 used for training and inference.

3) *Deep Neural Network*: The DNN serves as a powerful tool for Deep Learning, leveraging multiple network layers. In our study, we employ a DNN for the MNIST dataset [35], as illustrated in Fig. 14. We pack four pre-processed images per batch to prevent overflow during the 128×64 (2^{15}) matrix multiplication. DNN training requires 12,500 batches. Thus, all the existing works [32]–[34], [57] not providing computation-communication parallelism will suffer as their on-chip memory is insufficient. The DNN is trained for ≈ 7000 (≈ 5.8 Bootstrappings per iteration) iterations and achieves 95.2% accuracy in 29 days using OpenFHE [2]. REED 2.5D could finish this in only 7.7 minutes. This is where our computation-communication parallelism shines, as a huge amount of ciphertexts are required for such an application. None of the works in literature offers this and are bound to suffer for any memory-intensive application.

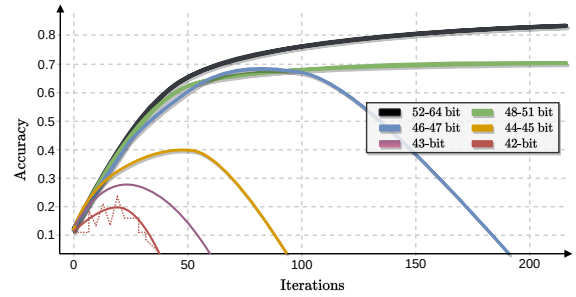


Fig. 15. Accuracy plot of different word sizes for the DNN. The lines are smoothened and the red dotted zig-zag line resembles the original form.

A. Precision-loss experimental study

Another facet of privacy-preserving computation is precision loss. Since the server cannot see the intermediate or final results, the best it can do is to ensure that the parameters it operates on support higher precision. To validate our parameter sets, we ran experiments for the DNN training. In Fig. 15, we can see how quickly the training accuracy drops as the word size is reduced. Thus, precision plays a vital role in providing privacy-preserving computation on the cloud. Our choice of 54-bit word size strikes the perfect balance between precision and performance. Works offering a smaller word-size [19], [32], [57] require in-depth study to mitigate the accuracy loss due to low-precision. Although we cannot prove that our parameters are the best, we ensure they can support most applications with a high precision guarantee. Our analysis will encourage readers to seek straightforward privacy-preserving solutions with maximal application coverage.

VI. CONCLUSION

FHE has garnered considerable interest due to its privacy-preserving computation capability. However, the major obstacle preventing its widespread deployment lies in its substantial computational overhead. Consequently, numerous efforts have been dedicated to accelerating fully homomorphic encryption in hardware; however, many of these attempts tend to focus excessively on acceleration at the expense of practicality. In this regard, our proposed accelerator design, REED, effectively addresses this limitation and achieves remarkable acceleration. Our approach utilizes a scalable design methodology that can be easily extended to larger configurations while also adapting to constrained environments.

We implement this methodology using a chiplet-based technique, which enables scalability. The experimental results highlight both the acceleration achieved and the practical implementation aspects of REED. Notably, our design is modular, paving the way for intriguing future prospects such as formal verification. Additionally, we plan to extend benchmarking to encompass larger network training scenarios to further demonstrate the utility of our parameters. Overall, the advancements presented in this work hold the promise of advancing privacy-preserving computations and promoting the wider adoption of fully homomorphic encryption.

ACKNOWLEDGEMENT

This work was supported in part by Samsung Electronics co. Ltd., Samsung Advanced Institute of Technology and the State Government of Styria, Austria – Department Zukunftsfonds Steiermark. We also extend our gratitude to Ian Khodachenko for his assistance in conducting the application benchmarking process.

REFERENCES

- [1] R. Agrawal, L. de Castro, G. Yang, C. Juvekar, R. T. Yazicigil, A. P. Chandrakasan, V. Vaikuntanathan, and A. Joshi, "FAB: an fpga-based accelerator for bootstrappable fully homomorphic encryption," in *IEEE International Symposium on High-Performance Computer Architecture, HPCA 2023, Montreal, QC, Canada, February 25 - March 1, 2023*. IEEE, 2023, pp. 882–895. [Online]. Available: <https://doi.org/10.1109/HPCA56546.2023.10070953>
- [2] A. Al Badawi, J. Bates, F. Bergamaschi, D. B. Cousins, S. Erabelli, N. Genise, S. Halevi, H. Hunt, A. Kim, Y. Lee, Z. Liu, D. Micciancio, I. Quah, Y. Polyakov, S. R.V., K. Rohloff, J. Saylor, D. Suponitsky, M. Triplett, V. Vaikuntanathan, and V. Zucca, "OpenFHE: Open-Source Fully Homomorphic Encryption Library," in *Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, ser. WAHC'22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 53–63. [Online]. Available: <https://doi.org/10.1145/3560827.3563379>
- [3] M. R. Albrecht, R. Player, and S. Scott, "On the concrete hardness of learning with errors," *J. Math. Cryptol.*, vol. 9, no. 3, pp. 169–203, 2015. [Online]. Available: <http://www.degruyter.com/view/j/jmc.2015.9.issue-3/jmc-2015-0016/jmc-2015-0016.xml>
- [4] A. A. Badawi, L. Hoang, C. F. Mun, K. Laine, and K. M. M. Aung, "Privt: Private and fast text classification with homomorphic encryption," *IEEE Access*, vol. 8, p. 226544–226556, 2020. [Online]. Available: <https://doi.org/10.1109/ACCESS.2020.3045465>
- [5] J. Bossuat, C. Mouchet, J. R. Troncoso-Pastoriza, and J. Hubaux, "Efficient Bootstrapping for Approximate Homomorphic Encryption with Non-sparse Keys," in *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I*, ser. Lecture Notes in Computer Science, A. Canteaut and F. Standaert, Eds., vol. 12696. Springer, 2021, pp. 587–617. [Online]. Available: https://doi.org/10.1007/978-3-030-77870-5_21
- [6] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "Fully homomorphic encryption without bootstrapping," *Electron. Colloquium Comput. Complex.*, p. 111, 2011. [Online]. Available: <https://eccc.weizmann.ac.il/report/2011/111>
- [7] Cadence, "3D-IC Technology," *Tech article*, 2023. [Online]. Available: https://www.cadence.com/en_US/home/explore/what-is-3dic.html
- [8] K. Chae, J. Park, J. Song, B. Koo, J. Oh, S. Yi, W. Lee, D. Kim, T. Yeo, K. Kang, S. Park, E. Kim, S. Jung, S. Park, S. Park, M. Noh, H. Rhew, and J. Shin, "A 4nm 1.15TB/s HBM3 Interface with Resistor-Tuned Offset-Calibration and In-Situ Margin-Detection," in *IEEE International Solid-State Circuits Conference, ISSCC 2023, San Francisco, CA, USA, February 19-23, 2023*. IEEE, 2023, pp. 406–407. [Online]. Available: <https://doi.org/10.1109/ISSCC42615.2023.10067736>
- [9] H. Chen, I. Chillotti, and Y. Song, "Improved Bootstrapping for Approximate Homomorphic Encryption," in *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II*, ser. Lecture Notes in Computer Science, Y. Ishai and V. Rijmen, Eds., vol. 11477. Springer, 2019, pp. 34–54. [Online]. Available: https://doi.org/10.1007/978-3-030-17656-3_2
- [10] 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*, ser. Lecture Notes in Computer Science, J. B. Nielsen and V. Rijmen, Eds., vol. 10820. Springer, 2018, pp. 360–384. [Online]. Available: https://doi.org/10.1007/978-3-319-78381-9_14
- [11] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A full RNS variant of approximate homomorphic encryption," in *Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers*, ser. Lecture Notes in Computer Science, C. Cid and M. J. J. Jr., Eds., vol. 11349. Springer, 2018, pp. 347–368. [Online]. Available: https://doi.org/10.1007/978-3-030-10970-7_16
- [12] J. H. Cheon, A. Kim, M. Kim, and Y. S. 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*, ser. Lecture Notes in Computer Science, T. Takagi and T. Peyrin, Eds., vol. 10624. Springer, 2017, pp. 409–437. [Online]. Available: https://doi.org/10.1007/978-3-319-70694-8_15
- [13] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: fast fully homomorphic encryption over the torus," *Journal of Cryptology*, vol. 33, no. 1, pp. 34–91, 2020.
- [14] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "ASAP7: A 7-nm finFET predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S002626921630026X>
- [15] A. C. Elster and T. A. Haugdahl, "Nvidia Hopper GPU and Grace CPU Highlights," *Computing in Science & Engineering*, vol. 24, no. 2, pp. 95–100, 2022.
- [16] EPFL-LDS and T. I. SA, "Lattigo v3.0.1," Feb 2022. [Online]. Available: <https://github.com/tuneinsight/lattigo>
- [17] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *IACR Cryptol. ePrint Arch.*, p. 144, 2012. [Online]. Available: <http://eprint.iacr.org/2012/144>
- [18] S. Fan, Z. Wang, W. Xu, R. Hou, D. Meng, and M. Zhang, "Tensorfhe: Achieving practical computation on encrypted data using GPGPU," in *IEEE International Symposium on High-Performance Computer Architecture, HPCA 2023, Montreal, QC, Canada, February 25 - March 1, 2023*. IEEE, 2023, pp. 922–934. [Online]. Available: <https://doi.org/10.1109/HPCA56546.2023.10071017>
- [19] 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)," 2021.
- [20] H. L. Garner, "The Residue Number System," *IRE Trans. Electron. Comput.*, vol. 8, no. 2, pp. 140–147, 1959. [Online]. Available: <https://doi.org/10.1109/TEC.1959.5219515>
- [21] R. Geelen, M. Van Beirendonck, H. V. L. Pereira, B. Huffman, T. McAuley, B. Selfridge, D. Wagner, G. Dimou, I. Verbauwhede, F. Vercauteren, and D. W. Archer, "BASALISC: Flexible Asynchronous Hardware Accelerator for Fully Homomorphic Encryption," 2022. [Online]. Available: <https://arxiv.org/abs/2205.14017>
- [22] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University, USA, 2009. [Online]. Available: <https://searchworks.stanford.edu/view/8493082>
- [23] J. L. Gonzalez, "Heterogeneous Integration of Chiplets Using Socketed Platforms, Off-Chip Flexible Interconnects, and Self-Alignment Technologies," Ph.D. dissertation, Georgia Institute of Technology, Atlanta, GA, USA, 2021. [Online]. Available: <https://hdl.handle.net/1853/64749>
- [24] A. Graening, S. Pal, and P. Gupta, "Chiplets: How Small is too Small?," *ACM/IEEE Design Automation Conference (DAC)*, 2023.
- [25] K. Han, S. Hong, J. H. Cheon, and D. Park, "Logistic regression on homomorphic encrypted data at scale," in *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 2019, pp. 9466–9471. [Online]. Available: <https://doi.org/10.1609/aaai.v33i01.33019466>
- [26] IBM, "IBM Cost of a Data Breach 2022 – Highlights for Cloud Security Professionals," *Technical Report*, 2020.
- [27] JEDEC, "High Bandwidth Memory DRAM (HBM3)," *Tech. Rep. JESD238*, 2022.
- [28] N. E. Jerger, "Presentation on Architecting Chiplet-Based Systems," 2020. [Online]. Available: <http://www.diti.umict.it/users/mpalesi/nocarc/nocarc20/pdf/Nocarc2020-EnrightJerger.pdf>

- [29] 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*, vol. 2021, no. 4, p. 114–148, Aug. 2021. [Online]. Available: <https://tches.iacr.org/index.php/TCHES/article/view/9062>
- [30] W. Jung, E. Lee, S. Kim, J. Kim, N. Kim, K. Lee, C. Min, J. H. Cheon, and J. H. Ahn, "Accelerating fully homomorphic encryption through architecture-centric analysis and optimization," *IEEE Access*, vol. 9, pp. 98772–98789, 2021. [Online]. Available: <https://doi.org/10.1109/ACCESS.2021.3096189>
- [31] A. Kim, Y. Song, M. Kim, K. Lee, and J. Cheon, "Logistic regression model training based on the approximate homomorphic encryption," *BMC Medical Genomics*, vol. 11, 10 2018.
- [32] 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, ISCA 2023, Orlando, FL, USA, June 17-21, 2023*, Y. Solihin and M. A. Heinrich, Eds. ACM, 2023, pp. 18:1–18:15. [Online]. Available: <https://doi.org/10.1145/3579371.3589053>
- [33] J. Kim, G. Lee, S. Kim, G. Sohn, J. Kim, M. Rhu, and J. H. Ahn, "ARK: Fully Homomorphic Encryption Accelerator with Runtime Data Generation and Inter-Operation Key Reuse," 2022. [Online]. Available: <https://arxiv.org/abs/2205.00922>
- [34] S. Kim, J. Kim, M. J. Kim, W. Jung, J. Kim, M. Rhu, and J. H. Ahn, "BTS: An Accelerator for Bootstrappable Fully Homomorphic Encryption," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 711–725. [Online]. Available: <https://doi.org/10.1145/3470496.3527415>
- [35] Y. LeCun and C. Cortes, "MNIST handwritten digit database," <http://yann.lecun.com/exdb/mnist/>, 2010. [Online]. Available: <http://yann.lecun.com/exdb/mnist/>
- [36] D. Lee, K. S. Lee, Y. Lee, K. W. Kim, J. Kang, J. Lee, and J. H. Chun, "Design considerations of HBM stacked DRAM and the memory architecture extension," in *2015 IEEE Custom Integrated Circuits Conference, CICC 2015, San Jose, CA, USA, September 28-30, 2015*. IEEE, 2015, pp. 1–8. [Online]. Available: <https://doi.org/10.1109/CICC.2015.7338357>
- [37] J. A. Lewis, Z. L. M. Smith, and E. Lostrì, "The Hidden Costs of Cybercrime," *Technical Report*, 2020.
- [38] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-42 2009), December 12-16, 2009, New York, New York, USA*, D. H. Albonesi, M. Martonosi, D. I. August, and J. F. Martínez, Eds. ACM, 2009, pp. 469–480. [Online]. Available: <https://doi.org/10.1145/1669112.1669172>
- [39] X. Ma, Y. Wang, Y. Wang, X. Cai, and Y. Han, "Survey on chipllets: interface, interconnect and integration methodology," *CCF Trans. High Perform. Comput.*, vol. 4, no. 1, pp. 43–52, 2022. [Online]. Available: <https://doi.org/10.1007/s42514-022-00093-0>
- [40] A. C. Mert, Aikata, S. Kwon, Y. Shin, D. Yoo, Y. Lee, and S. S. Roy, "Medha: Microcoded Hardware Accelerator for computing on Encrypted data," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2023, no. 1, pp. 463–500, 2023. [Online]. Available: <https://doi.org/10.46586/tches.v2023.i1.463-500>
- [41] A. C. Mert, E. Öztürk, and E. Savaş, "Design and implementation of a fast and scalable ntt-based polynomial multiplier architecture," in *2019 22nd Euromicro Conference on Digital System Design (DSD)*. IEEE, 2019, pp. 253–260.
- [42] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of computation*, vol. 44, no. 170, pp. 519–521, 1985.
- [43] S. Morgan, "McAfee Vastly Underestimates The Cost Of Cybercrime," *Cybersecurity Report*, 2020.
- [44] B. Murdock, "What Designers Need to Know About HBM3," *Synopsys*, Accessed on July 11, 2023. [Online]. Available: <https://www.synopsys.com/designware-ip/technical-bulletin/hbm3-ip-dwtb.html>
- [45] MUSE Semiconductor, "TSMC UNIVERSITY FINFET PROGRAM," <https://www.musesemi.com/university-finfet-program>. Accessed July 27th 2023.
- [46] M. Nabeel, D. Soni, M. Ashraf, M. A. Gebremichael, H. Gamil, E. Chielle, R. Karri, M. Sanduleanu, and M. Maniatakos, "CoFHEE: A Co-processor for Fully Homomorphic Encryption Execution," 2022. [Online]. Available: <https://arxiv.org/abs/2204.08742>
- [47] S. Narasimha, B. Jagannathan, A. Ogino, D. Jaeger, B. Greene, C. Sheraw, K. Zhao, B. Haran, U. Kwon, A. K. M. Mahalingam, B. Kannan, B. Morganfeld, J. Dechene, C. Radens, A. Tessier, A. Hassan, H. Narisetty, I. Ahsan, M. Aminpur, C. An, M. Aquilino, A. Arya, R. Augur, N. Baliga, R. Bhelkar, G. Biery, A. Blauberg, N. Borjemscaia, A. Bryant, L. Cao, V. Chauhan, M. Chen, L. Cheng, J. Choo, C. Christiansen, T. Chu, B. Cohen, R. Coleman, D. Conklin, S. Crown, A. da Silva, D. Dechene, G. Derderian, S. Deshpande, G. Dillway, K. Donegan, M. Eller, Y. Fan, Q. Fang, A. Gassaria, R. Gauthier, S. Ghosh, G. Gifford, T. Gordon, M. Gribelyuk, G. Han, J. Han, K. Han, M. Hasan, J. Higman, J. Holt, L. Hu, L. Huang, C. Huang, T. Hung, Y. Jin, J. Johnson, S. Johnson, V. Joshi, M. Joshi, P. Justison, S. Kalaga, R. Kim, W. Kim, R. Krishnan, B. Krishnan, K. Anil, M. Kumar, J. Lee, R. Lee, J. Lemon, S. Liew, P. Lindo, M. Lingalugari, M. Lipinski, P. Liu, J. Liu, S. Lucarini, W. Ma, E. Maciejewski, S. Madiseti, A. Malinowski, J. Mehta, C. Meng, S. Mitra, C. Montgomery, H. Nayfeh, T. Nigam, G. Northrop, K. Onishi, C. Ordonio, M. Ozbek, R. Pal, S. Parihar, O. Patterson, E. Ramanathan, I. Ramirez, R. Ranjan, J. Sarad, V. Sardesai, S. Saudari, C. Schiller, B. Senapati, C. Serrau, N. Shah, T. Shen, H. Sheng, J. Shepard, Y. Shi, M. Silvestre, D. Singh, Z. Song, J. Sporre, P. Srinivasan, Z. Sun, A. Sutton, R. Sweeney, K. Tabakman, M. Tan, X. Wang, E. Woodard, G. Xu, D. Xu, T. Xuan, Y. Yan, J. Yang, K. Yeap, M. Yu, A. Zainuddin, J. Zeng, K. Zhang, M. Zhao, Y. Zhong, R. Carter, C.-H. Lin, S. Grunow, C. Child, M. Lagus, R. Fox, E. Kaste, G. Gomba, S. Samavedam, P. Agnello, and D. K. Sohn, "A 7nm cmos technology platform for mobile and high performance compute application," in *2017 IEEE International Electron Devices Meeting (IEDM)*, 2017, pp. 29.5.1–29.5.4.
- [48] M. Park, J. Lee, K. Cho, J. H. Park, J. Moon, S. Lee, T. Kim, S. Oh, S. Choi, Y. Choi, H. S. Cho, T. Yun, Y. J. Koo, J. Lee, B. K. Yoon, Y. J. Park, S. Oh, C. K. Lee, S. Lee, H. Kim, Y. Ju, S. Lim, K. Y. Lee, S. Lee, W. S. We, S. Kim, S. M. Yang, K. Lee, I. Kim, Y. Jeon, J. Park, J. C. Yun, S. Kim, D. Lee, S. Oh, J. Shin, Y. Lee, J. Jang, and J. Cho, "A 192-Gb 12-High 896-GB/s HBM3 DRAM With a TSV Auto-Calibration Scheme and Machine-Learning-Based Layout Optimization," *IEEE J. Solid State Circuits*, vol. 58, no. 1, pp. 256–269, 2023. [Online]. Available: <https://doi.org/10.1109/JSSC.2022.3193354>
- [49] G. Paulin, F. Zaruba, S. Mach, M. Eggimann, M. Cavalcante, P. Scheffler, Y. Zhang, T. Fischer, N. Wistoff, L. Bertaccini, T. Benz, L. Colagrande, A. D. Mauro, A. Kurth, S. Riedel, N. Huetter, G. Ottavi, Z. Jiang, B. Muheim, F. K. Gurkaynak, D. Rossi, and L. Benini, "Occamy: A 432-core, Multi-TFLOPs RISC-V-Based 2.5D Chiplet System for Ultra-Efficient (Mini-)Floating-Point Computation," *PULP Platform, ETH Zurich*, 2023. [Online]. Available: <https://pulp-platform.org/occamy/>
- [50] B. Reagen, W. Choi, Y. Ko, V. Lee, G.-Y. Wei, H.-H. S. Lee, and D. Brooks, "Cheetah: Optimizing and Accelerating Homomorphic Encryption for Private Inference," 2020. [Online]. Available: <https://arxiv.org/abs/2006.00505>
- [51] D. Reis, M. T. Niemier, and X. S. Hu, "A computing-in-memory engine for searching on homomorphically encrypted data," *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, vol. 5, no. 2, pp. 123–131, 2019.
- [52] D. Reis, J. Takeshita, T. Jung, M. Niemier, and X. S. Hu, "Computing-in-memory for performance and energy-efficient homomorphic encryption," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 11, p. 2300–2313, Nov 2020. [Online]. Available: <http://dx.doi.org/10.1109/TVLSI.2020.3017595>
- [53] M. S. Riazzi, K. Laine, B. Pelton, and W. Dai, "HEAX: an architecture for computing on encrypted data," in *ASPLOS '20: Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, March 16-20, 2020*, J. R. Larus, L. Ceze, and K. Strauss, Eds. ACM, 2020, pp. 1295–1309. [Online]. Available: <https://doi.org/10.1145/3373376.3378523>
- [54] R. L. Rivest, L. Adleman, and M. L. Dertouzos, "On data banks and privacy homomorphisms," *Foundations of Secure Computation, Academia Press*, pp. 169–179, 1978.
- [55] S. S. Roy, K. Järvinen, F. Vercauteren, V. Dimitrov, and I. Verbauwhede, "Modular hardware architecture for somewhat homomorphic function

- evaluation,” in *Cryptographic Hardware and Embedded Systems - CHES*, 2015.
- [56] S. S. Roy, K. Järvinen, J. Vliegen, F. Vercauteren, and I. Verbauwhede, “HEPCloud: An FPGA-based multicore processor for FV somewhat homomorphic function evaluation,” *IEEE Transactions on Computers*, 2018.
- [57] N. Samardzic, A. Feldmann, A. Krastev, N. Manohar, N. Genise, S. Devadas, K. Eldefrawy, C. Peikert, and D. Sanchez, “CraterLake: A hardware accelerator for efficient unbounded computation on encrypted data,” in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 173–187. [Online]. Available: <https://doi.org/10.1145/3470496.3527393>
- [58] M. Scott, “A note on the implementation of the number theoretic transform,” in *Cryptography and Coding - 16th IMA International Conference, IMACC 2017, Oxford, UK, December 12-14, 2017, Proceedings*. Springer, 2017, pp. 247–258.
- [59] M. Scott, “Linear regression - insurance dataset,” 2020. [Online]. Available: <https://www.kaggle.com/code/kianwee/linear-regression-insurance-dataset>
- [60] “Microsoft SEAL (release 3.7),” <https://github.com/Microsoft/SEAL>, Sep. 2021, microsoft Research, Redmond, WA.
- [61] S. Sinha Roy, F. Turan, K. Jarvinen, F. Vercauteren, and I. Verbauwhede, “Fpga-based high-performance parallel architecture for homomorphic computing on encrypted data,” in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 387–398.
- [62] D. Sprenkels, “The Kyber/Dilithium NTT,” Accessed on July 21, 2023, <https://dsprenkels.com/ntt.html>.
- [63] Y. Sun, Y. Yuan, Z. Yu, R. Kuper, I. Jeong, R. Wang, and N. S. Kim, “Demystifying CXL memory with genuine cxl-ready systems and devices,” *CoRR*, vol. abs/2303.15375, 2023. [Online]. Available: <https://doi.org/10.48550/arXiv.2303.15375>
- [64] J. Takeshita, D. Reis, T. Gong, M. Niemier, X. S. Hu, and T. Jung, “Algorithmic acceleration of b/fv-like somewhat homomorphic encryption for compute-enabled ram,” Cryptology ePrint Archive, Report 2020/1223, 2020, <https://ia.cr/2020/1223>.
- [65] T. Thorolfsson, K. Gonsalves, and P. D. Franzon, “Design automation for a 3DIC FFT processor for synthetic aperture radar: a case study,” in *Proceedings of the 46th Design Automation Conference, DAC 2009, San Francisco, CA, USA, July 26-31, 2009*. ACM, 2009, pp. 51–56. [Online]. Available: <https://doi.org/10.1145/1629911.1629928>
- [66] P. Vivet, E. Guthmuller, Y. Thonnart, G. Pillonnet, G. Moritz, I. Miro-Panades, C. F. Tortolero, J. Durupt, C. Bernard, D. Varreau, J. J. H. Pontes, S. Thuries, D. Coriat, M. Harrand, D. Dutoit, D. Lattard, L. Arnaud, J. Charbonnier, P. Coudrain, A. Garnier, F. Berger, A. Gueugnot, A. Greiner, Q. L. Meunier, A. Farcy, A. Arriordaz, S. Cheramy, and F. Clermidy, “2.3 A 220gops 96-core processor with 6 chiplets 3d-stacked on an active interposer offering 0.6ns/mm latency, 3tb/s/mm² inter-chiplet interconnects and 156mw/mm² @ 82%-peak-efficiency DC-DC converters,” in *2020 IEEE International Solid-State Circuits Conference, ISSCC 2020, San Francisco, CA, USA, February 16-20, 2020*. IEEE, 2020, pp. 46–48. [Online]. Available: <https://doi.org/10.1109/ISSCC19947.2020.9062927>
- [67] W. Wang and X. Huang, “Fpga implementation of a large-number multiplier for fully homomorphic encryption,” in *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2013, pp. 2589–2592.
- [68] W. Wang, X. Huang, N. Emmart, and C. Weems, “Vlsi design of a large-number multiplier for fully homomorphic encryption,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 9, pp. 1879–1887, 2014.
- [69] G. Xin, Y. Zhao, and J. Han, “A multi-layer parallel hardware architecture for homomorphic computation in machine learning,” in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.
- [70] 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)*, 2023, pp. 870–881.
- [71] Z. Ye, R. C. C. Cheung, and K. Huang, “Pipentt: A pipelined number theoretic transform architecture,” *IEEE Trans. Circuits Syst. II Express Briefs*, vol. 69, no. 10, pp. 4068–4072, 2022. [Online]. Available: <https://doi.org/10.1109/TCSII.2022.3184703>
- [72] J. Yin, Z. Lin, O. Kayiran, M. Poremba, M. S. B. Altaf, N. D. E. Jerger, and G. H. Loh, “Modular Routing Design for Chiplet-Based Systems,” in *45th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2018, Los Angeles, CA, USA, June 1-6, 2018*, M. Annavaram, T. M. Pinkston, and B. Falsafi, Eds. IEEE Computer Society, 2018, pp. 726–738. [Online]. Available: <https://doi.org/10.1109/ISCA.2018.00066>
- [73] F. Zaruba, F. Schuiki, and L. Benini, “Manticore: A 4096-Core RISC-V Chiplet Architecture for Ultraefficient Floating-Point Computing,” *IEEE Micro*, vol. 41, no. 2, pp. 36–42, 2021.
- [74] Y. Zhang, S. Wang, X. Zhang, J. Dong, X. Mao, F. Long, C. Wang, D. Zhou, M. Gao, and G. Sun, “Pipezk: Accelerating zero-knowledge proof with a pipelined architecture,” in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 416–428.