SHARP: A Short-Word Hierarchical Accelerator for Robust and Practical Fully Homomorphic Encryption

Jongmin Kim
Seoul National University
Seoul, Republic of Korea
jongmin.kim@snu.ac.kr

Sangpyo Kim
Seoul National University
Seoul, Republic of Korea
vnb987@snu.ac.kr

Jaewan Choi
Seoul National University
Seoul, Republic of Korea
cjw9202@snu.ac.kr

Jaiyoung Park
Seoul National University
Seoul, Republic of Korea
jeff1273@snu.ac.kr

Donghwan Kim
Seoul National University
Seoul, Republic of Korea
eastflame@snu.ac.kr

Jung Ho Ahn
Seoul National University
Seoul, Republic of Korea
gajh@snu.ac.kr

ABSTRACT

Fully homomorphic encryption (FHE) is an emerging cryptographic technology that guarantees the privacy of sensitive user data by enabling direct computations on encrypted data. Despite the security benefits of this approach, FHE is associated with prohibitively high levels of computational and memory overhead, preventing its widespread use in real-world services. Numerous domain-specific hardware designs have been proposed to address this issue, but most of them use excessive amounts of chip area and power, leaving room for further improvements in terms of practicality.

We propose SHARP, a robust and practical accelerator for FHE. We analyze the implications of various hardware design choices on the functionality, performance, and efficiency of FHE. We conduct a multifaceted analysis of the impacts of the machine word length choice on the FHE acceleration, which, despite its importance with regard to hardware efficiency, has yet to be explored due to its complex correlation with various FHE parameters. A relatively short word length of 36 bits is discovered to be a robust and efficient solution for FHE accelerators. We devise an efficient hierarchical SHARP microarchitecture with a novel data organization and specialized functional units and substantially reduce the on-chip memory capacity requirement through architectural and software enhancements. This study demonstrates that SHARP delivers superior performance over prior FHE accelerators with a distinctly smaller chip area and lower power budget.

CCS CONCEPTS

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

KEYWORDS

fully homomorphic encryption, accelerator, word length, hierarchical architecture

ISCA '23, June 17–21, 2023, Orlando, FL, USA

ACM ISBN 979-8-4007-0095-8/23/06.
https://doi.org/10.1145/3579371.3589053

ACM Reference Format:
Jongmin Kim, Sangpyo Kim, Jaewan Choi, Jaiyoung Park, Donghwan Kim, and Jung Ho Ahn. 2023. SHARP: A Short-Word Hierarchical Accelerator for Robust and Practical Fully Homomorphic Encryption. In *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA '23), June 17–21, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3579371.3589053

1 INTRODUCTION

As cloud-based everything-as-a-service (XaaS) is becoming prevalent [38], the scope of user data shared with cloud servers is rapidly broadening to even include sensitive private data, such as financial or medical data. Despite the growing concern for data privacy, conventional security measures are failing to cover all attack surfaces, including side channels [73, 82].

Homomorphic encryption (HE) fundamentally blocks the possibility of data exposure by ensuring that data remains encrypted outside of user devices. By allowing direct computations on encrypted data by the server, HE enables users to offload useful computations without sacrificing privacy. This unique characteristic has inspired numerous privacy-preserving applications of HE. Specifically, CKKS [28], an HE scheme supporting the encryption of real (or complex) vectors, has gained popularity with use cases in private machine learning (ML) workloads [17, 48, 68, 75, 78, 89].

There are two methods of applying CKKS to workloads, differing in terms of how they treat errors that accumulate on encrypted data, or *ciphertext*, over *HE operations (HE ops). Leveled HE (LHE)* only allows a limited number of HE ops, requiring extra user-side computations and communication to mitigate errors and thus to continue operating [9]. In contrast, *fully homomorphic encryption (FHE)* features a *bootstrapping* op that can be performed by the server to refresh errors. Despite the benefits of high versatility and a simplified workflow without intermediate user intervention, FHE causes extremely high server-side computation and memory overhead that prevents it from being adopted in real-world services.

To mitigate the overhead, various hardware solutions for FHE CKKS have been proposed, ranging from the use of general-purpose CPU/GPU systems [18, 42, 62, 63] to custom-made accelerators in FPGA [1, 101] and ASIC [67, 70, 95]. In particular, ASIC solutions show performance improvements exceeding two orders of magnitude over state-of-the-art GPU implementations and thus represent a key enabler of future FHE adoption.

However, prior ASIC solutions rely on excessive amounts of chip area and power to store and process the massive working sets of FHE CKKS, which can reach hundreds of MBs. As HE is an emerging application, greater commercial demand for HE may be needed before aggressive prior proposals become practical. Therefore, to deliver high performance at a greatly reduced hardware cost, we explore avenues of opportunity that prior accelerators did not fully investigate.

As a first step, we analyze the possibility and various trade-offs when reducing the machine word length of an FHE CKKS accelerator, which can be met with immediate hardware efficiency benefits of superlinear reductions in logic area and power. Numerous domain-specific architectures, ones targeting ML [91] in particular, adopt short word lengths, even as short as a single bit [7, 8]. Short-word architectures are possible in ML due to the robustness of ML to low-precision arithmetic and specialized quantization techniques [56, 81]. In contrast, how the functionality of FHE CKKS workloads are affected by shorter word lengths is not well understood in the community, let alone performance and hardware efficiency. For the first time, we analyze the various implications of different word lengths on FHE CKKS acceleration to derive a compelling solution.

Based on a detailed analysis, we propose **SHARP**, a **S**hort-Word **H**ierarchical **A**ccelerator for **R**obust and **P**ractical FHE. We observe that an HE accelerator using a short word length is more susceptible to memory and communication bandwidth bottlenecks. Therefore, we devise a hierarchical structure that forces the majority of global data communication to instead be conducted locally, significantly reducing the requirement for on-chip data communication for the most costly function in HE, the number-theoretic transform (NTT). We propose a novel data ordering method along with functional units (FUs) that support the hierarchy and changes that arise when using a short word length. Moreover, we substantially reduce the on-chip memory capacity to a more practical level and devise architectural and software optimizations targeting the reduced capacity.

Our concerted efforts enable SHARP to provide 1.57× to 11.5× higher performance on average with a 1.25× to 2.34× smaller chip area and 1.15× to 1.68× less power consumption on average compared to state-of-the-art FHE CKKS accelerators when performing representative workloads.

Overall, this paper makes the following key contributions:

- We analyze the impact of the word length choice on the functionality, performance, and efficiency of an FHE CKKS accelerator in detail and derive a solution that supports representative FHE workloads robustly and efficiently.
- We propose the SHARP microarchitecture, which minimizes the memory and communication bandwidth usage through a novel hierarchical organization and functional units specialized for the organization.
- We devise architectural and algorithmic optimizations that enable the use of significantly smaller on-chip memory while preventing frequent off-chip memory access.
- Throughout the paper, we make 12 observations (highlighted in **boldface**) regarding the implications of various hardware design decisions on FHE CKKS, which can be used as guidelines for future accelerator research.

Table 1: CKKS notations, parameters, and key operations.

Notation	Description
$\mathbf{m}$	Message, a vector of n complex (real) numbers.
P, $\mathsf{P}_{\mathbf{m}}$	Polynomial or plaintext, $\mathsf{P}_{\mathbf{m}}$ corresponds to a message $\mathbf{m}$.
$[[\mathbf{m}]]$	Ciphertext, corresponds to a message $\mathbf{m}$.
n	Length of a vector message, $n \leq N/2$.
N	Degree, # (number) of coefficients in a polynomial.
Δ	Scale multiplied to the message during encryption.
$\mathcal{R}_Q$	Cyclotomic polynomial ring, $\mathbb{Z}_Q/(X^N+1)$.
Q	Initial polynomial modulus of $\mathcal{R}_Q$.
P	Auxiliary modulus used with Q in $\mathcal{R}_{PQ}$.
q_i	Small RNS prime composing $Q = \prod_{i=0}^{L-1} q_i$.
p_i	Small RNS prime composing $P = \prod_{i=0}^{K-1} p_i$.
L	Maximum level, # of q_i's composing Q.
K	# of p_i's composing P.
ℓ	(Current) level, # of q_i's currently left in the modulus.
L_{eff}	Effective level, # of rescaling possible between bootstrapping.
dnum	Decomposition number, $\mathsf{dnum} = \lceil L/K \rceil \geq 1$.
HAdd	$\text{HAdd}([[\mathbf{m}]], [[\mathbf{m}']]) \rightarrow [[\mathbf{m}+\mathbf{m}']]$.
PMult	$\text{PMult}([[\mathbf{m}]], \mathsf{P}_{\mathbf{m}'}) \rightarrow [[\mathbf{m}\cdot\mathbf{m}']]$.
HMult	$\text{HMult}([[\mathbf{m}]], [[\mathbf{m}']], \mathbf{evk}_{\text{mult}}) \rightarrow [[\mathbf{m}\cdot\mathbf{m}']]$.
HRot	$\text{HRot}([[\mathbf{m}]], r, \mathbf{evk}_{\text{rot}}^{(r)}) \rightarrow [[\mathbf{m} \ll r]]$, r: rotation amount.
evk	Evaluation key. HMult uses the same single $\mathbf{evk}_{\text{mult}}$ and HRot uses a different $\mathbf{evk}_{\text{rot}}^{(r)}$ for each rotation amount r.

2 BACKGROUND

We describe the construction and basic operations (ops) of CKKS. Notations and terminologies are based on [36, 49, 67]. Table 1 summarizes the key notations and ops.

2.1 CKKS Encryption and HE Operations

State-of-the-art HE schemes [21, 22, 30, 39, 41] including CKKS [28], are based on the RLWE [83] problem, which is known to be secure against cryptographic attacks, including even those that utilize quantum computers. In CKKS, a user initially *packs n* number of real or complex numbers into a vector *message* ($\mathbf{m}$). $\mathbf{m}$ is transformed into a degree-$(N-1)$ ($n \leq N/2$) integer polynomial ($\mathsf{P}_{\mathbf{m}}$), also referred to as *plaintext*. The transformation multiplies $\mathbf{m}$ with a large *scale* Δ and performs rounding ($\lfloor \cdot \rceil$) to obtain integers with small rounding errors. $\mathsf{P}_{\mathbf{m}}$ is encrypted into a *ciphertext* ($[[\mathbf{m}]]$) using Eq. 1 with a random large-coefficient polynomial $\mathsf{A}_{\mathbf{m}}$, a *secret polynomial* S, and a small-coefficient *error polynomial* E. RLWE ensures the hardness of extracting $\mathsf{P}_{\mathbf{m}}$, thus $\mathbf{m}$, from knowing only $[[\mathbf{m}]]$, which is a pair of polynomials in the cyclotomic polynomial ring $\mathcal{R}_Q$ having extremely large integer coefficients (e.g., $\simeq 2^{1200}$).

$$\begin{aligned} &\mathbf{m} = (m_0, m_1, \cdots, m_{n-1}) \in \mathbb{C}^n \\ &\lfloor \Delta \cdot \mathbf{m} \rceil \mapsto \mathsf{P}_{\mathbf{m}} = a_0 + a_1 X + \cdots + a_{N-1} X^{N-1} \\ &[[\mathbf{m}]] = (\mathsf{B}_{\mathbf{m}}, \mathsf{A}_{\mathbf{m}}) = (\mathsf{A}_{\mathbf{m}} * \mathsf{S} + \mathsf{P}_{\mathbf{m}} + \mathsf{E}, \mathsf{A}_{\mathbf{m}}) \in \mathcal{R}_Q^2 \end{aligned} \tag{1}$$

$[[\mathbf{m}]]$ can be sent to a server to offload computation on $\mathbf{m}$. When offloading the calculation of $f(\mathbf{m})$, for example, the server performs a *homomorphic evaluation* of f on $[[\mathbf{m}]]$. This requires a series of *primitive HE ops* to be performed on $[[\mathbf{m}]]$, which include evaluating

an addition or multiplication (mult) op between $[[\mathbf{m}]]$ and another ciphertext (HAdd, HMult), a plaintext (PAdd, PMult), or a constant (CAdd, CMult), cyclic rotation of $[[\mathbf{m}]]$ (HRot), and more. Primitive HE ops can be combined to construct more complex HE ops.

2.2 Primary Functions of CKKS

Primitive HE ops are further subdivided into *primary (polynomial) functions*, including *number-theoretic transform (NTT)*, *base conversion (BConv)*, *automorphism*, and other element-wise functions (add, mult, sub, and more).

Residue number system (RNS): To efficiently handle large integers, CKKS employs the *residue number system (RNS)* [14, 27]. In $\mathcal{R}_Q$, all integer ops are modular ops with respect to the *polynomial modulus* Q. We can set Q to be the product of *RNS primes* $q_0, \cdots, q_{L-1}$, each small enough to fit in a machine word, and decompose each coefficient of a polynomial using Eq. 2. Then, expensive large-integer ops can be replaced by a set of L parallel modulo-q_i ops.

$$\text{RNS: } a \mapsto (a \bmod q_0, a \bmod q_1, \cdots, a \bmod q_{L-1}) \tag{2}$$

RNS transforms a degree-$(N-1)$ polynomial into an $L \times N$ matrix, where each row, referred to as a *limb* of the polynomial, corresponds to a unique prime q_i. Each limb can be regarded as a separate polynomial in $\mathcal{R}_{q_i}$.

Number-theoretic transform (NTT): In $\mathcal{R}_Q$, polynomial mult is equivalent to a (negacyclic) convolution between two vectors of coefficients, having $O(N^2)$ complexity when naïvely calculated. NTT, an integer-version Fourier transform, can be applied to each vector, converting the convolution into a simple element-wise mult and reducing the overall complexity to $O(N \log N)$ by applying FFT algorithms [32, 44]. Inverse NTT (INTT) should be performed to obtain the final result, but polynomials are usually kept in their NTT-applied versions, referred to as the *evaluation representation*, as more element-wise functions can be evaluated in this representation. However, some functions require the polynomial to be restored to its original form, referred to as the *coefficient representation*, by performing INTT. When used along with RNS, (I)NTT is applied to each limb of the $L \times N$ matrix.

Base conversion (BConv) and evaluation keys (evks): When two polynomials have different polynomial moduli, BConv is performed to match the modulus. This is done when we must multiply a polynomial $\mathsf{P} \in \mathcal{R}_Q$ with an *evaluation key*, $\mathbf{evk} \in \mathcal{R}_{PQ}^{2\times \mathtt{dnum}}$, for HMult and HRot ops, which include the same subroutine, called *key-switching* [49]. HMult uses the same single **evk** ($\mathbf{evk}_{\text{mult}}$) and HRot uses a separate **evk** ($\mathbf{evk}_{\text{rot}}^{(r)}$) for each *rotation amount* r. It is required for the **evk**s to use an *auxiliary modulus* $P = \prod_{i=0}^{K-1} p_i$ along with $Q = \prod_{i=0}^{L-1} q_i$. Each p_i RNS prime is larger than $\max(q_i)$. With RNS, an **evk** is expressed as $\mathtt{dnum} = \lceil L/K \rceil$ (*decomposition number*) pairs of $(L+K) \times N$ matrices.

BConv mostly involves computing matrix-matrix mult [67] between $\mathsf{P} \in \mathcal{R}_Q$, an $L \times N$ matrix, and a predefined $K \times L$ matrix referred to as *base table*, producing $\mathsf{P}' \in \mathcal{R}_P$, a $K \times N$ matrix. P' is concatenated with P to form an *extended polynomial* $\mathsf{P}'' \in \mathcal{R}_{PQ}$, an $(L+K) \times N$ matrix. A similar conversion from $\mathcal{R}_{PQ}$ to $\mathcal{R}_Q$ is also done with BConv. BConv requires the input to be in the coefficient representation before computation. Therefore, the sequence of INTT → BConv → NTT is frequently observed.

Automorphism: For a degree-$(N-1)$ polynomial in the coefficient representation, automorphism maps the i-th coefficient to the $(i \cdot 5^r \bmod N)$-th coefficient's position when performing HRot by rotation amount r. This can also be performed with a similar mapping in the evaluation representation.

2.3 Bootstrapping and Security

Rescaling: The scale of the output ciphertext becomes Δ^2 when performing a multiplicative HE op (HMult, PMult, or CMult) between two operands of scale Δ. An HE op called *rescaling* reduces the scale back to Δ through the approximate division of the ciphertext by Δ. When used with RNS, we select RNS primes (q_i's) close to Δ and instead divide by the last RNS prime. During this process, the last limb of the polynomial corresponding to q_{L-1} is discarded and the modulus is reduced from Q to Q/q_{L-1}. Rescaling is repeated for each multiplicative HE op using primes $q_{L-2}, q_{L-3}, \cdots$.

Level: Due to rescaling, the number of q_i RNS primes used in a polynomial changes in an HE workload. We define the number of q_i RNS primes left in the modulus of a ciphertext as the ciphertext's *(current) level* (ℓ). A ciphertext at level ℓ is a pair of $\ell \times N$ matrices.

Bootstrapping: After multiple HE ops, the modulus becomes too small to perform more rescaling ops. *Bootstrapping* restores the modulus to allow more HE ops. However, bootstrapping is an expensive process, composed of multiple HE ops consuming many levels and dissipating Q_{boot} modulus. Therefore, the restored modulus is much smaller than Q and only L_{eff} ($\ll L$) number of rescaling ops are possible between a consecutive pair of bootstrapping ops, which we refer to as the *effective level*. A higher L_{eff} reduces the frequency of bootstrapping. We adopt the state-of-the-art bootstrapping implementation in prior work [20, 40, 67].

Security: The security of an HE scheme is mainly determined by two factors: N and PQ [34]. A high N and a low PQ are necessary for security. FHE requires a high Q exceeding Q_{boot}, which harms the security. Therefore, a high N ($2^{15} \sim 2^{17}$) is required to offset this. For a certain security target, fixing N also fixes PQ. For example, $N = 2^{16}$ and $\log PQ = 1,555$ form a pair [19, 40] offering 128 bits of security,[1] which is a standard security target [6, 11] we abide by throughout this paper. When PQ is fixed due to the security constraints, dnum (Section 2.2) creates a trade-off among L_{eff}, the working set size, and the required computation. Increasing dnum results in a higher L_{eff} (higher L and Q), but also increases the **evk** size and computational complexity.

2.4 Prior FHE CKKS Accelerators

Due to the large parameter values FHE must support, FHE has a large working set size that can reach hundreds of MBs even for a single basic HE op, while also requiring much more computation. The high computation and memory overhead makes FHE computation over 10,000× slower than unencrypted computation on conventional computing systems [63].

To mitigate the overhead, abundant prior research has attempted hardware acceleration of HE [1, 18, 63, 71, 72, 85, 90, 92, 93, 99]. F1 [94] is the first ASIC work supporting CKKS, but its performance is greatly degraded in FHE circumstances. Recently, three ASIC accelerators fully supporting FHE CKKS have been proposed:

[1] n-bit security: the best-known attack will take $\geq 2^n$ ops to break the scheme.

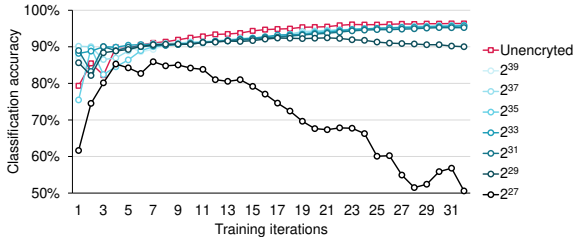


Figure 1: Accuracy of a logistic regression binary classifier over 32 training iterations using HELR [48] with the MNIST dataset (1,024 images per batch) while varying the scales as in Table 2. The unencrypted training (FP64) result is also shown. The accuracy values are averaged over five trials.

CraterLake [95], BTS [70], and ARK [67]. These ASIC proposals achieve higher performance by more than two orders of magnitude compared to the state-of-the-art GPU implementation [62]. However, they are highly costly with massive chip areas ranging from 373mm^2 to 472mm^2 and power consumption from 163W to 320W.

3 WORD LENGTH FOR ROBUST AND PRACTICAL FHE

For improved practicality, first we explore the possibility of using a short word length for a compact hardware design.

3.1 Impact of the Word Length on Workload Functionality

Previous studies lack consideration of the negative effect of the short word length on the precision of CKKS workloads, which is affected by the random error polynomial included in the encryption process (Section 2.1) and the inherent errors in HE ops. Also, how the degree of precision affects the functionality of CKKS workloads is not well understood in the community. A number of prior studies have attempted to enhance the precision of CKKS [12, 19, 20, 64, 66, 77, 79], but most have focused on enhancing only the precision of bootstrapping, the noisiest HE op in CKKS.

The word length of an HE accelerator must be long enough to contain the q_i or p_i RNS prime, the size of which is determined by the scale (Δ). An RNS prime must be close to Δ for rescaling while also satisfying Eq. 3 for (I)NTT.

$$\text{Requisite for (I)NTT on } \mathbb{Z}_{q_i}\text{: } q_i = 1 \bmod 2N \tag{3}$$

CraterLake uses a 28-bit word length. For a word length shorter than 28 bits, it is difficult to find primes that are close to each other [66] and that also satisfy Eq. 3. Other accelerators use more conventional word lengths such as 32 bits (F1) or 64 bits (BTS, ARK).

The size of Δ must be sufficient to guarantee the precision of the workload. When using a small Δ, *error explosions* occur, in which completely useless output values are produced. Although the numbers initially encrypted into a CKKS ciphertext are regarded to be numerically similar to fixed-point numbers [28] in the range of $[-1, 1]$, as we perform HE ops, random errors behave more unstably than rounding errors in fixed-point numbers [45]. Also, although CKKS itself is tolerant against overflows with the ability to contain numbers up to nearly $Q/\Delta \gg 1$, we observe that when the numbers are out of the initial $[-1, 1]$ range, the result becomes highly instable.[2] Many workloads include the approximation of non-linear functions with polynomial functions to enable an evaluation with HE ops [52, 76]. This polynomial approximation is highly accurate when the input values are around zero, but its accuracy severely degrades as the magnitude of the values grows.

Table 2: Precision* of a freshly encrypted ciphertext (Fresh prec.), precision after bootstrapping (Boot prec.), and application functionality depending on the scale choices.

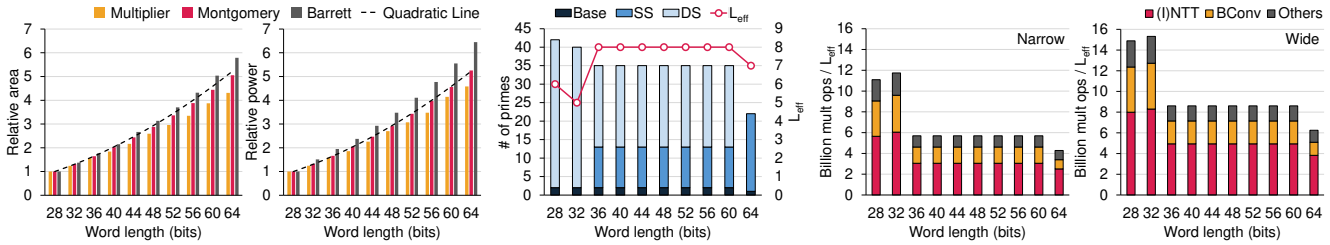
Normal scale (SS)	2^{27}	2^{29}	2^{31}	2^{33}	2^{35}	2^{37}	2^{39}
Fresh prec. (bits)	14.19	16.32	18.44	20.34	22.39	24.43	26.43
Boot scale (DS)	2^{55}	2^{59}	2^{60}	2^{62}	2^{62}	2^{64}	2^{64}
Boot prec. (bits)	13.37	14.86	17.28	19.29	21.86	23.78	25.50
HELR† acc. after 32 iters (Fig. 1)	50.58%	90.01%	95.24%	95.76%	95.88%	95.82%	95.82%
ResNet-20‡ acc. (3,000 samples)	10.37%	9.97%	10.87%	89.53%	91.90%	91.73%	91.77%
Sorting max err.* (16,384 elements)	5.2e+75	4.4e-4	1.4e-4	2.9e-5	8.0e-6	4.4e-6	3.8e-6

* Precision is measured as $\log_2 \epsilon^{-1}$, where ϵ is the maximum magnitude of error (max err.) in the decrypted result averaged over 5 trials.
† Unencrypted model (FP64) shows 96.37% accuracy for 1,984 test images.
‡ Unencrypted model (FP32) shows 92.18% accuracy for 10,000 images.

Bootstrapping requires a higher Δ value than normal HE ops because 1) the former involves a sequence of normal HE ops, accumulating a high degree of error in the process, and 2) the polynomial approximation necessary for bootstrapping introduces additional errors. State-of-the-art implementations [20, 40, 79] use different scales for bootstrapping and normal HE ops to gain similar precision; *bootstrapping scales*[3] are set to around $2^{50} \sim 2^{60}$ and *normal scales* to around $2^{25} \sim 2^{45}$. A 64-bit machine can easily find primes having similar values to each of the varying scales. However, machines with shorter word lengths should handle high scales (e.g., 2^{55}) by rescaling with two primes having values close to $\sqrt{\Delta}$ ($2^{27.5}$), referred to as *double-prime scaling (DS)*, contrary to the conventional *single-prime scaling (SS)*.

We observe that, ① **to use SS for normal scales while guaranteeing robust execution of FHE CKKS workloads, at least a 36-bit word length is required.** We created parameter sets[4] using $N = 2^{16}$, dnum $= 3$, and $\log PQ \leq 1,555$ with different normal scales while setting the bootstrapping precision close to the precision of a freshly encrypted ciphertext by adjusting the bootstrapping scale (see Table 2). We implemented representative FHE workloads (see Section 6.1) on Lattigo [40]. Fig. 1 and Table 2 show that the workloads work for much lower normal scales than the original implementations [48, 52, 75] using normal scales in the

[2] We secure bootstrapping precision outside the range by applying [12].
[3] Although even higher bootstrapping scales may be desirable for precision [29], prior FHE research usually does not employ scales higher than 2^{60} to be able to contain each number in a 64-bit word.
[4] [20] suggested the use of a high Hamming weight (h) for secret polynomials, which allows one to increase $\log PQ$ to around 1,790 without sacrificing the security or bootstrapping precision. However, we observed that this method damages the overall precision and degrades the ResNet-20 accuracy to 56%, even for the 2^{39} normal scale. Thus, we used the bootstrapping implementation in [20], but set h to 192.



(a) ALU area and power scaling (b) Prime setting and L_{eff} (c) Word-length operation count

Figure 2: (a) Area (left) and power (right) scaling of the major ALUs in HE, including general multiplier, Montgomery modular multiplier, and Barrett modular multiplier, (b) the number of q_i RNS primes dedicated to the base (never rescaled), SS, and DS, along with L_{eff}, and (c) the number of word-length ops, translated in terms of mult ops divided by L_{eff}, for the narrow (left) and wide (right) workloads. The results are shown for various settings with word lengths ranging from 28 to 64 bits.

range $2^{40} \sim 2^{46}$, partially due to the new techniques for high precision [19, 66] we applied. Also, we removed the excessively high margin for precision in the original implementations; e.g., in ResNet-20 [75], they divide the ciphertext by 1,000 to adjust the number range, which loses 10 bits of precision, whereas we find that division by only 10 is acceptable. HELR starts to work from $\Delta = 2^{29}$, and its accuracy does not improve after $\Delta = 2^{35}$. Sorting shows an even lower magnitude of error than the original implementation (max err. = 3.2e-4) from $\Delta = 2^{31}$. Encrypted ResNet-20 inference requires a higher $\Delta = 2^{33}$ to work, and its accuracy does not improve after $\Delta = 2^{35}$.

For smaller normal scales, error explosions were observed for all of the workloads (e.g., max err. = 5.2e+75 in sorting). This is well shown in Fig. 1, where the weight values start from 0, become larger over the iterations, and eventually leave the stable range and quickly degrade the classification accuracy when $\Delta = 2^{27}$. This suggests that machines with a word length shorter than 36 bits should use DS even for normal scales to support all of the workloads without functionality degradation.

Although it cannot be guaranteed that the normal scale of 2^{35} can provide enough precision for all existing and future FHE workloads, as the FHE workloads we used for the experiment are some of the most complex FHE workloads currently available, the results suggest that 2^{35} should be able to support a vast subset robustly, including a majority of currently available and future FHE workloads, especially FHE ML workloads, with over 22 bits of precision.

3.2 Implication of the Word Length on FHE CKKS Acceleration

We conducted an analysis of how word lengths affect the performance and efficiency of FHE hardware acceleration. We prepared 128-bit secure ($N = 2^{16}$, dnum = 3, and $\log PQ \leq 1,555$) settings with word lengths that varied from 28 to 64 bits. We refer to each setting using the k-bit word length as Set_k. Based on Section 3.1, we set minimum normal/bootstrapping scales to $2^{35}/2^{62}$. For each Set_k, if the scale exceeds what can be expressed in a word, DS is utilized. We relieved the conditions in favor of Set_{28}, which cannot support the 2^{62} scale with DS, by setting its bootstrapping scale to 2^{55}. Instead, we used a slightly more complex bootstrapping algorithm for Set_{28}, which requires 1.05× more computation, to maintain a similar degree of bootstrapping precision. As shown in Fig. 2(b), Set_{28} and Set_{32} always use DS. Set_{36} through Set_{60} utilize the same set of primes, where 11 out of $L = 35$ primes are used for SS. Set_{64} always uses SS.

Hardware cost: ② The hardware cost of the major ALUs in HE shows a quadratic relationship with the word length. We synthesized the major ALUs used in HE for each word length setting in RTL, i.e., general multipliers, Montgomery modular multipliers [84], and Barrett modular multipliers [15]. The results are shown in Fig. 2(a). Set_{64} ALUs occupy 5.01× more area and dissipate 5.37× more power in geometric means compared to those of Set_{28}, close to a quadratic scaling of 5.22×, which is partially due to the use of more complex logic to satisfy the timing constraints associated with the longer-word implementations.

Max level (L) and L_{eff}: ③ Although using a short word length with DS allows the use of more q_i RNS primes (higher L) for a fixed Q, doing so may not directly lead to a higher L_{eff}, as the inability to use SS can degrade L_{eff}. Due to the lack of small primes satisfying Eq. 3, Set_{28} ($L_{\text{eff}} = 6$) and Set_{32} ($L_{\text{eff}} = 5$) cannot set normal scales lower than 2^{47} using DS. Using excessively high normal scales exhausts the polynomial modulus (Q) more quickly by rescaling and thus degrades L_{eff}. Settings with word lengths longer than 36 bits, associated with a 2^{35} normal scale, have higher L_{eff} values. However, Set_{64} ($L_{\text{eff}} = 7$), which always uses SS, has a lower L_{eff} than the others ($L_{\text{eff}} = 8$) because Set_{64} uses higher p_i primes to satisfy the $p_i > \max(q_i)$ condition. Therefore, Set_{64} uses a higher P value and consequently a lower Q value, resulting in a slightly lower L_{eff}.

The following implications of the word length on HE differ greatly from those in other domains, such as ML, where a reduced word length leads to a proportionally reduced working set size without an increase in the operational count.

Working set: ④ The working set size of HE does not change much with different word lengths with most of the bits in a word being utilized. The size of the working set is determined by the sizes of the ciphertext and **evk**. Compared to Set_{28}, the size of an **evk** only increases by 1.08× and 1.22× for Set_{36} and Set_{64}, respectively, representing a very modest increase considering that the ratio of word length is 1 : 1.29 : 2.29. In addition, the size of a ciphertext shows modest corresponding increases of 1.07× and

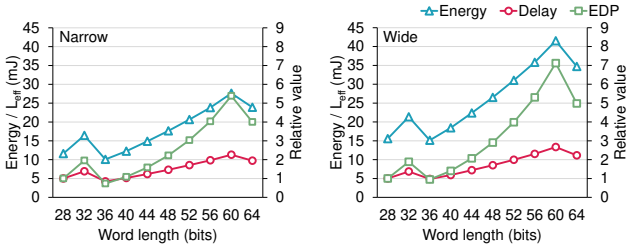


Figure 3: Energy and relative delay, each divided by L_{eff}, and relative energy-delay product (EDP) of each word length setting for the narrow (left) and wide (right) workloads.

1.20×. The size of the working set is mainly determined by the HE parameters, such as N, dnum, and $\log PQ$, and an increased word length is significant only when bits in a word are not fully utilized when using primes with under-fitting sizes, as in Set_{40} through Set_{60} (e.g., an **evk** of Set_{60} is 1.80× larger than that of Set_{28}).

Operational count: ⑤ Short word length settings require much more word-length operations due to DS. We counted the number of major integer ops (mult, Montgomery and Barrett modular reductions) and calculated the weighted sum while considering the relative logic area cost of each op to integer mult. Similar to prior studies [70, 95], we used synthetic workloads covering a wide range of applications to analyze the operational cost: a *narrow workload* performing a single HMult for every L_{eff} after bootstrapping and a *wide workload* performing 30. The total operational counts are divided by L_{eff} to obtain per-level counts because real workloads have a fixed level consumption. The results in Fig. 2(c) show sharp increases in the number of word-length ops when DS is applied more. Increased L due to DS results in much more word-length ops because the primary functions of HE have $O(L)$ (e.g., (I)NTT) to $O(L^2)$ (e.g., BConv) complexity. High per-level counts of Set_{28} and Set_{32} are partially due to the degraded L_{eff} (③). Set_{28} respectively requires 1.95× and 2.59× more word-length ops than Set_{36} and Set_{64} for the narrow workload, and 1.73× and 2.38× for the wide one.

3.3 36-bit Word Length for Robust and Practical FHE

The factors discussed in Section 3.2 have conflicting impacts on the performance and efficiency of an FHE CKKS accelerator. Therefore, we put ② ~ ⑤ together, and derived the energy and delay values, each divided by L_{eff}, and the energy-delay product (EDP) for each word length setting. We combined Fig. 2(a) and Fig. 2(c) by assuming that the same chip area is filled with the synthesized ALUs in Fig. 2(a) for each setting.

Fig. 3 shows that, ⑥ **for the three aspects of energy, delay, and EDP, Set_{36} shows the best efficiency for both narrow and wide workloads.** Set_{40} through Set_{60}, which use the same set of primes as Set_{36} do not provide any benefit due to the unnecessarily long word length usage compared to Set_{36}. Compared to conventional Set_{64}, Set_{36} requires 2.37× (resp., 2.29×) less energy and 2.31× (2.29×) less delay, showing 5.47× (5.26×) lower EDP, for the narrow (wide) workload. Also, compared to Set_{28}, Set_{36} requires 1.15× (resp., 1.03×) less energy and 1.19× (1.03×) less delay, resulting in a 1.37× (1.06×) lower EDP for the narrow (wide) workload.

Although targeting different scales may produce different numbers, we conclude that the 36-bit word length functions as a compelling trade-off point in terms of robustness and efficiency. Set_{36} supports various workloads by mixing SS and DS and improves the practicality of an FHE accelerator compared to the prior word length choices of 64 bits (BTS, ARK), 32 bits (F1), and 28 bits (CraterLake). It saves significant amounts of chip area and power compared to the conventional 64-bit SS setting, while achieving better performance and efficiency compared to the 28-bit and 32-bit DS settings due to the higher L_{eff} and lower L values.

4 SHARP MICROARCHITECTURE

4.1 Overview

Based on Section 3, we design SHARP, an FHE CKKS accelerator that uses a 36-bit word length. Our design goal is to reduce area and power consumption for the practical execution of FHE workloads with performance comparable to those in prior studies, such as CraterLake, BTS, and ARK. Using a shorter word length can help to reduce the area and power consumption of computational logic (②), but cannot reduce the working set size and data load substantially (④). Therefore, with the same computational logic area, ⑦ **using a shorter word length intensifies the on-chip bandwidth bottlenecks in data communication and memory access**, which is already a major challenge to FHE CKKS acceleration [67, 95]. ⑦ is especially true for recent technology nodes (e.g., 7nm) as wires have not scaled as well as logic gates, rendering bandwidth extremely expensive compared to logic [35, 51, 60].

The vector architecture adopted in F1, CraterLake, and ARK has shown fruitful results with regard to ⑦. Therefore, we also adopt the vector architecture and set ARK, the state-of-the-art work, as the starting point for our design; SHARP adopts the global configuration of ARK — four *clusters* each having $\sqrt{N} = 256$ lanes, lane-wise NoC, and the same global data distribution policy between the clusters. We add the pseudo random number generator (PRNG) for **evk**s proposed in CraterLake to the baseline architecture, which reduces the storage and memory bandwidth used for **evk**s by half.

SHARP is distinguished from previous vector architectures in terms of its novel *hierarchical architecture* tailored to (I)NTT. Our architecture divides each cluster into $M = \sqrt[4]{N} = 16$ *lane groups*,[5] each consisting of $M = 16$ adjacent lanes. We change the data organization method inside each cluster, replacing the long-distance data exchanges required in prior vector architectures for (I)NTT with much closer exchanges between adjacent lanes within each lane group. (I)NTT accounts for more than half of the computation in FHE (see Fig. 2(c)) and has a complex data exchange pattern which is challenging to handle in vector architectures. Therefore, how the architecture supports (I)NTT determines the quality of an HE accelerator, in the same way as matrix-matrix-mult FUs [26, 61] do for an ML accelerator. We describe in detail how SHARP efficiently handles (I)NTT with its hierarchical architecture and *ten-step NTT unit (NTTU)* in Section 4.2.

[5] CraterLake also uses this terminology, but we refer to CraterLake's lane group as cluster, as its functionality is similar to the cluster as defined in other architectures.

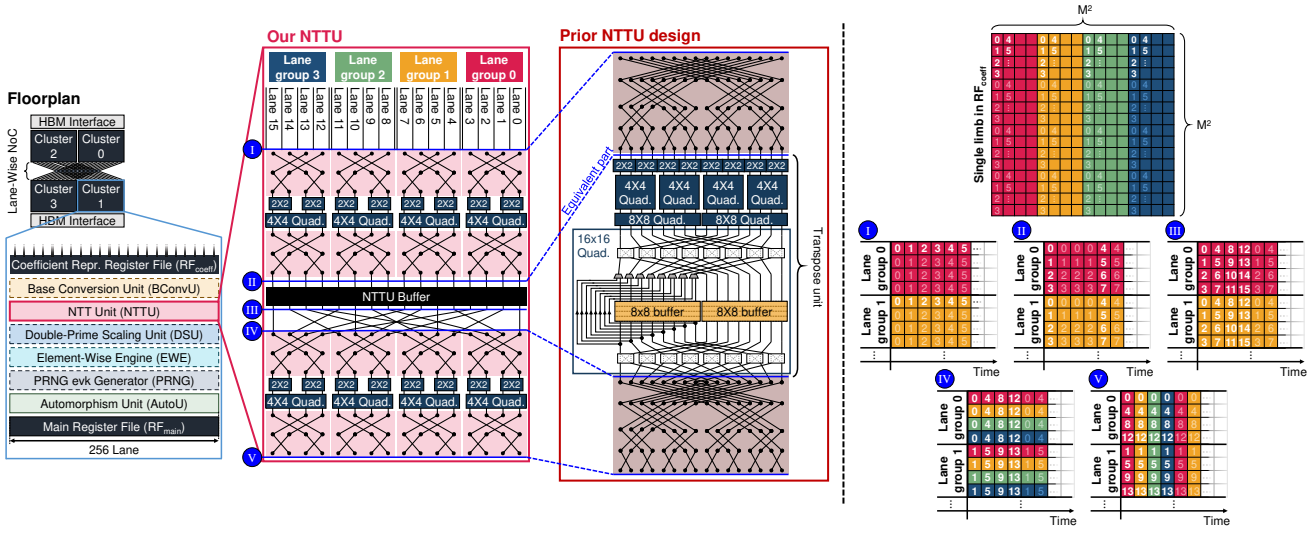


Figure 4: Organization of SHARP. SHARP has 4 vector clusters connected by lane-wise NoC, which is the configuration proposed in ARK. FUs depicted with dotted borders are fully parallelizable across the $\sqrt{N} = 256$ lanes. SHARP adds a hierarchy of $M = 16$ lane groups, each composed of M lanes, inside a cluster (diagrams are simplified to $M = 4$). A comparison of our NTTU design to the prior NTTU design in F1/CraterLake/ARK is shown. CraterLake does not include the transpose unit with multiple quadrant swap units (Quad.), but instead requires a global transpose network with high bandwidth. How data elements are distributed inside our NTTU during NTT is shown in I through V. Each color represents the lane group the data element belongs to at the start of NTT, and each number the cycle in which the data element is initially fed into the NTTU from RF$_{coeff}$.

In addition, to better support the hierarchical structure and the use of a 36-bit word length, SHARP introduces novel memory organization and FUs, including an *element-wise engine (EWE)* and a *double-prime scaling unit (DSU)*, and enhances prior designs of the *BConv unit (BConvU)* and *automorphism unit (AutoU)*. The resulting floorplan of SHARP is shown in Fig. 4.

4.2 Hierarchical Architecture for (I)NTT

Prior vector NTTU: F1 proposes a deeply-pipelined vector NTTU which effectively saves the limited on-chip memory bandwidth. Its NTTU applies a well-known four-step 2D-FFT [13] to (I)NTT and pipelines the entire process of N-point (I)NTT by the sequence of $\sqrt{N}$-point *butterfly* $\rightarrow$ *transpose* $\rightarrow$ *twisting* $\rightarrow$ another $\sqrt{N}$-point butterfly. For each butterfly step, a *butterfly unit* composed of $\frac{\sqrt{N}}{2} \log \sqrt{N}$ Montgomery modular multipliers is placed to multiply data elements with *twiddle factors*, which are connected in a butterfly network. Twisting allows each multiplier to reuse the same twiddle factor throughout (I)NTT of a limb, by instead multiplying each element with its required offset value, called a *twisting factor*. However, twisting factors change for each data element. A *transpose unit* performs an all-to-all data exchange of the data elements in a limb distributed across the lanes, which is required for 2D-FFT-like execution. It consists of multiple stages of quadrant swaps, which conditionally swap data between the upper half and lower half of the lanes. ARK further proposes *on-the-fly twisting (OF-Twist)*, which generates twisting factors at runtime, using the property that twisting factors form a geometric sequence (e.g., $1, \zeta, \zeta^2, \cdots$). OF-Twist substantially saves the on-chip memory space by storing only the common ratio (ζ) per lane, using it to generate all twisting factors used for (I)NTT of a limb.

Although F1's NTTU is effective for a small number of lanes, recent FHE accelerators, CraterLake and ARK, expand it to span 256 lanes, inducing severe data communication costs for exchanging data across lanes that are hundreds of lanes apart. Significant area and power overhead arises due to the semi-global wires inside the clusters (see the butterfly and transpose units in Fig. 4). CraterLake attempts to mitigate the overhead with its global transpose network, which replaces the transpose unit in NTTU with a fixed global NoC. However, the global transpose approach requires excessive global communication between all of its 2,048 lanes for (I)NTT and automorphism, which is even more critical; CraterLake requires a much higher global NoC bandwidth than our baseline, ARK (see Table 4).

Ten-step NTTU: To reduce the high communication overhead in (I)NTT considering ⑦, we devise a *ten-step NTTU* along with a corresponding hierarchical structure. Our NTTU regards an input limb as an $M^2 \times M^2$ matrix for $M = \sqrt[4]{N}$. Initially, the first M^2-point four-step NTT *phase* (I $\rightarrow$ II in Fig. 4) is performed. Each lane group receives a column of the matrix over M cycles from RF$_{coeff}$, performing NTT over the column. As each four-step NTT is performed within a lane group, long semi-global connections are unnecessary. The *intra-lane-group* transpose unit converts M-strided access across the lanes (I) into non-strided access (II) for a column. These results are stored in an *NTTU buffer* and are read from the buffer in a different order (II $\rightarrow$ III) after all of the columns arrive. An *inter-lane-group transpose* is performed through a direct

cluster-wide wire connection (III → IV), which is the only semi-global connection required in our NTTU design. Finally, the second four-step NTT phase (IV → V) is performed. Each lane group performs an M^2-point NTT over a row of the matrix and stores the results into $\mathrm{RF}_{\mathrm{main}}$. INTT is performed in the reverse order.

Bit-reversed row access and double OF-Twist: We identify a method by which to apply OF-Twist in our ten-step NTTU. In the first four-step NTT phase, M^2 twisting factors at a lane form the following sequence (simplified to $M = 4$):

$$\text{Phase 1 } (M=4)\text{: } 1, \zeta, \zeta^2, \zeta^3, 1, \zeta, \zeta^2, \zeta^3, 1, \zeta, \zeta^2, \zeta^3, 1, \zeta, \zeta^2, \zeta^3$$

The sequence can be broken down into M geometric sequences with the same common ratio (ζ), making OF-Twist easily applicable.

The second four-step NTT phase is more complicated. We discover that accessing the M rows assigned to a lane group in a *bit-reversed row access order* enables the use of OF-Twist in the second phase. For example, lane group 0, which is assigned rows 0, 4, 8, and 12 when $M = 4$, must access the rows in the order of $0 \rightarrow 8 \rightarrow 4 \rightarrow 12$. Then, we can make the twisting factors form the following sequence:

$$\text{Phase 2 } (M=4)\text{: } 1, \zeta, \zeta^2, \zeta^3, 1, \zeta^3, \zeta^6, \zeta^9, 1, \zeta^5, \zeta^{10}, \zeta^{15}, 1, \zeta^7, \zeta^{14}, \zeta^{21}$$

The sequence can be broken down into M geometric sequences whose common ratios ($\zeta, \zeta^3, \zeta^5, \zeta^7$) also form a geometric sequence. We devise a specialized *double OF-Twist unit* to support this pattern, which receives the first common ratio (ζ), and the common ratio of common ratios (ζ^2), and generates the entire sequence on-the-fly.

Also, twiddle factors for butterfly units change every M cycle in the second phase. Nevertheless, bit-reversed row access also causes the twiddle factors to form a geometric sequence. Therefore, we place *(single) OF-Twiddle units*, similar to OF-Twist units, inside the butterfly units of the second phase.

To simplify data access, we use the bit-reversed row access order as the default data ordering inside $\mathrm{RF}_{\mathrm{main}}$, based on the observation that ⑧ **data ordering inside a limb does not affect primary functions other than (I)NTT and automorphism as long as all limbs follow the same data ordering.** Henceforth, *sequential access* to $\mathrm{RF}_{\mathrm{main}}$ means accessing the data in this order.

Efficiency: Our hierarchical architecture is a solution scalable to cluster designs with many (256) lanes utilized in prior vector FHE accelerators, reducing the horizontal bisection bandwidth of an NTTU by six-fold (see Table 4) and requiring 9.17× shorter wiring for horizontal connections compared to that required for ARK. The area and power benefits are detailed in Section 6.5.

4.3 Register File (RF) Organization and Automorphism

SHARP has two register files (RFs): $\mathrm{RF}_{\mathrm{main}}$ that stores polynomials in the evaluation representation and $\mathrm{RF}_{\mathrm{coeff}}$ that stores polynomials in the coefficient representation. Each RF is heavily banked, with multiple banks per lane group, and each bank is spliced into six sub-banks, similar to the RF structure of NVIDIA GPUs [10]. A sub-bank is 96-bit wide, meaning that six can provide 576 bits of data, one 36-bit word for each lane every cycle. Banks are interleaved to serve multiple (4R4W for each of $\mathrm{RF}_{\mathrm{main}}$ and $\mathrm{RF}_{\mathrm{coeff}}$) requests from a lane every cycle [87]. Bank conflicts can be avoided because we always sequentially access data over the entirety of reading or writing to a limb, which lasts $M^2 = 256$ cycles. We capitalize on the sequential access to simplify the control and addressing by placing small lane-group-wise counters to keep track of the address LSB instead of sending cluster-wide address signals every cycle.

One exception to sequential access is automorphism, where input data elements are shuffled and cause the output to violate the sequential access. However, due to the special property of automorphism [67, 70], if we read one data element from each lane every cycle for automorphism, they will all map to different lanes and are thus free from contention. Also, the destination lanes of M data elements from the same lane group (i.e., lane group A) are always in another identical lane group (lane group B). Using this property, we allow *lane-group-wise addressing* exclusively for storing the output of an automorphism unit (AutoU). The basic structure of AutoU follows that of ARK, but our AutoU also shuffles the destination address calculation results and delivers to the $\mathrm{RF}_{\mathrm{main}}$ of each lane group. Also, a small output buffer is included per lane group to reorder write requests to sequentially access the memory banks. AutoU outputs always evenly access memory banks in the finest granularity, i.e., evenly for every (# of banks) cycle.

Table 3: Instructions supported by the element-wise engine (EWE). The operands are in actual port order. NI: NTTU input, PI: PRNG input, CI: constant input, NO: NTTU output.

Instr.	$\mathrm{RF}_{\mathrm{main}}$ in	NI	PI	CI	$\mathrm{RF}_{\mathrm{main}}$ out	NO	Usage
Tensor	A, A′, B, B′	-	-	-	$\mathsf{D}_0 = \mathsf{BB}'$ $\mathsf{D}_1 = \mathsf{AB}' + \mathsf{A}'\mathsf{B}$ $\mathsf{D}_2 = \mathsf{AA}'$	D_2	HMult
AccQ	$\mathsf{D}_0, \mathsf{D}_1, \mathsf{B}_k, \mathsf{D}_2$	-	A_k	c	$\mathsf{E}_0 = \mathsf{D}_2\mathsf{B}_k + c\mathsf{D}_0$ $\mathsf{E}_1 = \mathsf{D}_2\mathsf{A}_k + c\mathsf{D}_1$	-	HMult, HRot
AccP	$\mathsf{D}_0, \mathsf{D}_1, \mathsf{B}_k$, -	D_2	A_k	-	$\mathsf{E}_0 = \mathsf{D}_2\mathsf{B}_k + \mathsf{D}_0$ $\mathsf{E}_1 = \mathsf{D}_2\mathsf{A}_k + \mathsf{D}_1$	-	HMult, HRot
ModD	B, -, -, -	B′	-	c	$\mathsf{D}_0 = c\mathsf{B} - c\mathsf{B}'$	-	HMult, HRot, Rescale
MAD	B′, A′, B, A	P	-	c	$\mathsf{D}_0 = \mathsf{PB} + c\mathsf{B}'$ $\mathsf{D}_1 = \mathsf{PA} + c\mathsf{A}'$	-	HAdd, PMult, PAdd, CMult, CAdd

4.4 Element-Wise Engine (EWE)

Primary functions other than (I)NTT, BConv, and automorphism are all element-wise functions. Although they only account for 14% of the computation on average (see Fig. 5(a)), they have very low arithmetic intensity (ops/byte), inducing high RF pressure. Prior accelerators allow data forwarding between FUs to reduce the RF pressure, though forwarding increases the hardware cost by requiring FU-side buffers to manage different arrival times of the operands while also increasing the compiler-side burden associated with controlling the transport [53].

Instead, we design a versatile *element-wise engine (EWE)* that supports five instructions. ⑨ **The five instructions in Table 3 are all that are needed to cover all compound element-wise computation patterns found in CKKS.** Input operands are optional to make each instruction serve multiple purposes. For example, by replacing P and c with 1, we can perform two parallel additions with a MAD instruction.

An EWE is composed of four Barrett modular multipliers, two adders, input and output buffers for interleaved bank access and synchronization, and a control unit. EWE serves as a centralized structure that replaces buffering and synchronization at each FU while also minimizing the hardware/software complexity.

4.5 Supporting the 36-Bit Word Length

Base conversion unit (BConvU): We extend the one-dimensional (1×6 per lane) systolic array BConvU of ARK to two-dimensional ($H \times W$ per lane) BConvU. For every cycle, our BConvU receives H data elements in the same limb of a polynomial and emits the H matrix multiplication results between the base table and the input polynomial (Section 2.2). Although increasing H results in higher performance, it also requires a higher RF_{coeff} bandwidth.

Higher BConv throughput is required for a short word length. Revisiting Fig. 2(c), while 20% of calculation on average is for BConv in Set_{64}, 27% in Set_{36} and 30% in Set_{28} are for BConv. The complexity of BConv increases with $\alpha = \lceil L/dnum \rceil$, where short-word settings have high L values. The lower dnum usage in SHARP makes α even higher compared to that in ARK. Also, the portion of BConv in the computation fluctuates with the level, between 21% of (I)NTT to 60% (see Fig. 5(a)). We design our BConvU to minimize the (I)NTT stall time in the frequent computation pattern of INTT → BConv → NTT (Section 2.2) for all levels; i.e., it is designed for a level with relatively high BConv complexity. We found that $H = 2$ and $W = 8$ is sufficient for the worst case, confirming that the workload performance saturates after the point through a simulation. Therefore, we use a 2×8 BConvU in SHARP.

Double-prime scaling unit (DSU): To support DS, double-word-length accumulation is required for rescaling and bootstrapping. We design a *double-prime scaling unit (DSU)* for this. In each lane, a DSU reads two data elements from RF_{coeff}, evaluates Eq. 4 for accumulation, and feeds the result to the NTTU.

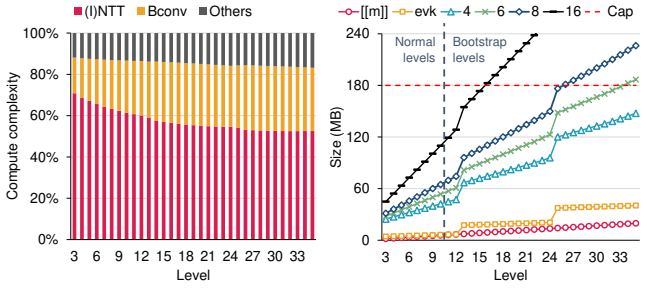
$$\text{For } a \in \mathbb{Z}_{q_i}, b \in \mathbb{Z}_{q_{i+1}}, \text{ compute } (a \cdot q_{i+1} + b \cdot q_i) \bmod q_j \qquad (4)$$

A DSU is also used during PMult to support the on-the-fly limb extension (OF-Limb) proposed in ARK, which generates the plaintexts ($\text{P}_{\mathbf{m}}$) used in PMult ops at runtime just with a single limb. Two input limbs are required for OF-Limb when using DS, where a DSU supports the runtime plaintext extension using them.

5 REDUCING ON-CHIP MEMORY CAPACITY

CraterLake (256+26MB), BTS (512+22MB), and ARK (512+76MB) all rely on a massive on-chip memory capacity to reuse data and mitigate the off-chip memory bandwidth bottleneck. An excessively large chip area, reaching even 262mm^2 at 7nm (ARK), is deployed to accommodate the on-chip memory. In SHARP, we reduce the on-chip memory capacity to 180+18MB, occupying 87.3mm^2 of chip area. With a reduced on-chip memory capacity, we need stricter on-chip memory management to handle the working set.

Fig. 5 shows how the computational complexity of HMult and the working set size change for different levels (ℓ). Until recently, the loading time of **evk**s from the off-chip memory was thought to be the factor dominating the FHE CKKS accelerator performance, with BTS even arguing that no more computational power is required than just enough to finish a single HE op (HMult or HRot) within



(a) Computational complexity (b) Working set size

Figure 5: (a) Computational complexity breakdown of HMult and (b) working set size on various levels. The working set sizes are shown for cases each requiring an evk and 4, 6, 8, or 16 ciphertexts. The size of a ciphertext ([[m]]), the size of an evk, and the RF_{main} capacity (Cap) are also shown. PRNG evk generation is assumed. $N = 2^{16}$, dnum $= 3$, $L = 35$, and $K = 12$.

the loading time of an **evk**. However, the minimum key-switching technique [46, 67] changes the situation by greatly enhancing the reusability of **evk**s; ⑩ **the working set size is now more sensitive to the number of temporary ciphertexts than to evks.** Fig. 5(b) reflects this change by showing the working set size for different numbers of temporary ciphertexts. A ciphertext is sized 19.7MB, and an **evk** 79.3MB (40.3MB when using PRNG), at the max level of our parameters.

HE ops at high levels, which are mostly used for bootstrapping, are much more memory- and compute-heavy than HE ops at lower levels. To support the reduced memory capacity, extra effort is required to ensure that the working set size does not exceed the capacity, as doing so will incur frequent off-chip memory accesses for temporary data. Such a situation rarely arises at low levels (see Fig. 5(b)); thus, ⑪ **we have to look out for the working set size only at high levels — that is, only while bootstrapping.** We apply several software optimizations to reduce the working set size while bootstrapping, and introduce some examples.

Memory-capacity-aware BSGS fine-tuning: We adjust the number of temporary ciphertexts required in the baby-step giant-step subroutine (BSGS) [46], which takes most of the bootstrapping time [67]. In BSGS, we can adjust the bs and gs values, where bs · gs $= D$ for a fixed D value (e.g., $D = 64$).[6]. The computation of BSGS has $O(\text{bs} + \text{gs})$ complexity, which can be minimized by setting bs $=$ gs $= \sqrt{D}$ (e.g., bs $=$ gs $= 8$). Instead, ⑫ **if we can hold** (bs + 1) **ciphertexts on-chip, we can reuse them for** gs **times during BSGS.** Thus, we set bs according to the number of ciphertexts the on-chip memory can accommodate, even when doing so results in higher computational complexity. For example, based on Fig. 5(b), we can choose bs $= 4$ for BSGS at the highest level and bs $=$ gs $= \sqrt{D}$ for BSGS at lower levels.

Operation fusion: We apply operation fusion extensively to eliminate temporary ciphertexts. In BSGS, for example, the process

[6]Another optimization in [24] allows using smaller D values at the cost of more levels consumed during bootstrapping (i.e. a lower L_{eff}). We also leverage [24], but with our BSGS fine-tuning, we can instead choose a relatively high D value to sacrifice less levels during bootstrapping

Table 4: Resources utilized in FHE accelerators. BW: bandwidth, Cap: capacity, TP: throughput, w/c: words per cycle.

Resource	CraterLake	ARK	SHARP
Word length	28-bit	64-bit	36-bit
Core frequency	1GHz	1GHz	1GHz
# of lanes	2,048	1,024	1,024
Off-chip memory BW	1TB/s	1TB/s	1TB/s
On-chip memory Cap	256+26MB	512+76MB	180+18MB
On-chip memory BW	84TB/s (excl. 26MB buffer)	20TB/s + 72TB/s	36TB/s + 36TB/s
Global NoC BW	8,192 w/c	1,024 w/c	1,024 w/c
# of NTTUs (total TP)	16 (4,096 w/c)	4 (1,024 w/c)	4 (1,024 w/c)
Horizontal bisection BW of an NTTU	256 w/c	768 w/c	128 w/c
BConvU configuration	60 parallel MAC units / lane	1×6 systolic array / lane	2×8 systolic array / lane
Element-wise op TP	5 mult & 5 add / lane	2 MAD / lane	4 mult & 2 add / lane (EWE)
Area (7nm-scaled)	472.3mm^2 (222.7mm^2)	418.3mm^2	178.8mm^2

of accumulating the results of PMult ops into one ciphertext frequently occurs. Instead of storing the temporary results of PMult ops separately, we utilize EWEs to fuse PMult and HAdd into one PMAD (plaintext multiply-add) op, reducing the working set by getting rid of temporary ciphertexts.

Data scheduling: To reduce the on-chip memory usage further, we aggressively reduce the working set size required inside RF_{main} for a single HE op (HMult/HRot in particular) to a size just large enough to hold two extended polynomials in $\mathcal{R}_{PQ}$ (Section 2.2). This is enabled by our data scheduling inside an HE op that aggressively prioritizes the removal of temporary values.

For more global-level data scheduling, SHARP adopts the greedy data scheduling policy of CraterLake. Due to the static nature of FHE workloads, the compiler can precisely predict the data usage order. Therefore, we can manage the on-chip memory with an optimal data replacement algorithm [16].

6 EVALUATION

6.1 Implementation and Experimental Setup

We synthesized the major logic units in RTL using the ASAP7 7.5-track 7nm predictive process design kit (PDK) [31]. We modified FinCACTI [96], a cache modeling tool, to reflect the published information in the 7nm logic and SRAM technologies [23, 55, 59, 60, 86, 97, 100] properly and used it to evaluate the long wiring and SRAM components. Two HBM stacks are utilized, each providing 500GB/s of bandwidth [57, 58], for which we estimated the area and power based on prior work [60, 88]. FUs and networks run fully pipelined at 1GHz. All SRAM components are single-ported and run double-pumped at 2GHz [43] providing 1R1W per cycle. Table 4 summarizes the resources utilized in SHARP. SHARP is 178.8mm^2 in size, with 66% for RF and HBM PHY (see Fig. 6(b)).

We built a cycle-level simulator for SHARP to evaluate performance, which receives FHE CKKS applications expressed as a sequence of HE ops, converts them to a data dependence graph of the primary functions, and assigns the primary functions to FUs. Data scheduling is done as discussed in Section 5.

We measured the runtime of bootstrapping and three representative FHE CKKS workloads using the simulator. Then, we compared SHARP against prior ASIC accelerators using their reported performance and power consumption values when targeting 128-bit security. For fairness, we excluded the cost of the PCIe Interface in BTS and assumed the same area for HBM PHY for all accelerators. We estimated the area and power consumption of CraterLake (14/12nm) at 7nm with the optimistic 14nm to 7nm area (2.8×) and power (2.2×) scaling reported in [86], denoting this as CLake+. We used FinCACTI to evaluate the SRAM components of CLake+. The workloads we used are as follows:

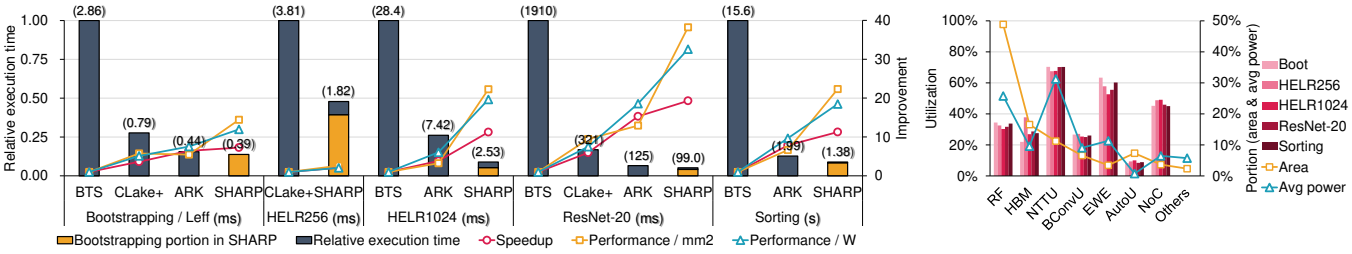
- **Bootstrapping:** We divided the bootstrapping runtime by L_{eff}. For CLake+, we assumed $L_{\text{eff}} = 8$, which is identical to that in SHARP and ARK, due to a lack of information.
- **HELR:** HELR [48] is an ML workload training a binary classification model using logistic regression. A 196-element weight is trained with 14×14 MNIST [37] images for classifying the numbers 3 and 8. We used a batch size of 256 (HELR256) or 1,024 (HELR1024) images. We trained the model for 32 iterations, where an iteration is a gradient update step with a single batch, and report the average execution time per iteration.
- **ResNet-20:** We performed a CNN inference with a $32 \times 32 \times 3$ CIFAR-10 [74] image using the FHE CKKS implementation of the ResNet-20 [50] model in [75].
- **Sorting:** We performed a two-way bitonic sorting on an array with 2^{14} numbers using a method in [52].

In SHARP, we used $2^{35}/2^{62}$ normal/bootstrapping scales in Table 2 for the workloads, except for sorting where we used $2^{31}/2^{60}$, based on the analysis in Section 3.1. DS is used for bootstrapping, and SS is used for the other ops.

6.2 Performance and Efficiency

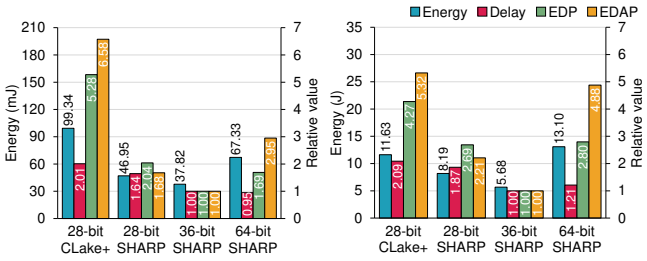
The 36-bit word length, the hierarchical design, and the reduced on-chip memory space together result in substantial reductions in the chip area and power. SHARP is 1.98×, 1.25×, and 2.34× smaller in terms of the chip area than BTS, CLake+, and ARK, respectively. Also, SHARP dissipates < 98W of power for the workloads, resulting in corresponding power reductions of 1.68× (vs. BTS), 1.15× (vs. CLake+), and 1.30× (ARK) in geometric mean (gmean).

Even with such a restricted area and power budget, SHARP shows superior performance against prior accelerators, as shown in Fig. 6(a). All of the workloads have a high portion of bootstrapping in their execution times varying from 59% to 95%, signifying the importance of accelerating bootstrapping in FHE. SHARP shows 11.5×, 2.39×, and 1.57× higher performance in gmean compared to BTS, CLake+, and ARK, respectively. Putting it all together, we compare the performance per area and performance per power to evaluate the efficiency of SHARP. SHARP shows increased improvements in gmean at 22.9×, 2.98×, and 3.67× in terms of the performance per chip area, and 19.4×, 2.75×, and 2.04× in terms of the performance per watt.



(a) Performance comparison with prior ASIC FHE accelerators (b) Component utilization & area, power portion

Figure 6: (a) Performance comparison between SHARP and prior accelerators: BTS, CLake+, and ARK. We measured power and execution time of bootstrapping (divided by L_{eff}), HELR256, HELR1024, ResNet-20, and sorting. (b) Utilization of components of SHARP, and their portions in the total area (178.8mm^2) and average power consumption (94.7W) of the workloads.



(a) Bootstrapping / L_{eff} (b) HELR256

Figure 7: Energy, delay, EDP, and EDAP of CLake+, 28-bit SHARP$_{28}$, 36-bit SHARP$_{36}$, and 64-bit SHARP$_{64}$ for (a) bootstrapping (energy and delay divided by L_{eff}) and (b) HELR256.

6.3 Utilization of Hardware Components

Fig. 6(b) shows the utilization of the hardware components, revealing that SHARP is mostly compute-bound. NTTUs, which participate in most of the HE ops, are highly active with 69% utilization on average. BConvUs are designed for the worst case and show a lower 26% utilization on average. The FUs required for particular HE ops (e.g., AutoU for HRot) are less busy. Due to the improved memory management, the memory components, whose bandwidth we set similar to those in prior work (see Table 4), show low utilization rates even with the reduced on-chip memory capacity, demonstrating the potential for further bandwidth reductions for efficiency.

6.4 Word Length Choice

We can reaffirm the observations in Section 3.3 that a 36-bit word length is an efficient choice for FHE accelerators. We designed a 28-bit version of SHARP with 168MB RF$_{\text{main}}$ (SHARP$_{28}$) and a 64-bit version with 200MB RF$_{\text{main}}$ (SHARP$_{64}$) considering ④ and compared them with the baseline SHARP$_{36}$ and CLake+, as shown in Fig. 7. SHARP$_{28}$ always uses DS by using a lower 2^{55} bootstrapping scale as in Section 3.2. SHARP$_{36}$ and SHARP$_{64}$ are respectively 1.22× and 2.12× larger than SHARP$_{28}$. Although greater amounts of chip area and power are required for SHARP$_{36}$ compared to SHARP$_{28}$, SHARP$_{36}$ reduces the delay by 1.64-1.87× and the energy by 1.24-1.44×, resulting in a 2.04-2.69× lower energy-delay product (EDP) and 1.68-2.21× lower energy-delay-area product (EDAP) [80]. SHARP$_{64}$ has a delay similar (0.95-1.21×) to that of SHARP$_{36}$, but requires much more chip area and power, showing 1.69-2.80× higher EDP and 2.95-4.88× higher EDAP values compared to those of SHARP$_{36}$.

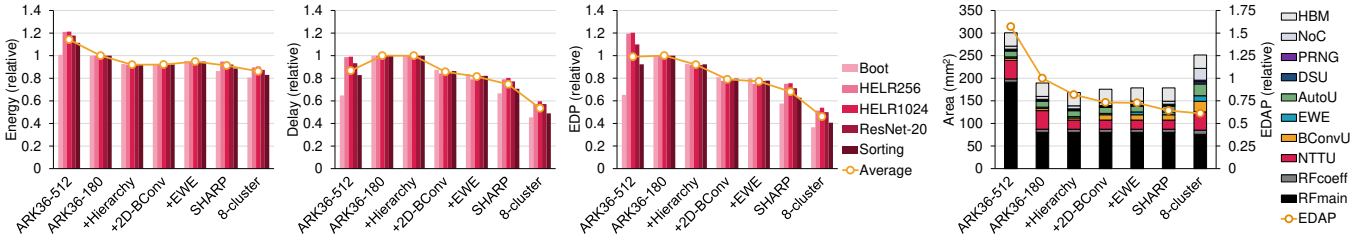
Although SHARP is not highly optimized for a 28-bit word length, SHARP$_{28}$ shows superior performance and efficiency compared to CLake+, which targets the 28-bit word length. SHARP$_{28}$ uses far fewer computing resources than CLake+ (see Table 4) and is 1.51× smaller (147.0mm^2) than CLake+. Nevertheless, our hardware and software co-optimization enables SHARP$_{28}$ to show 1.59-2.58× lower EDP and 2.41-3.92× lower EDAP values compared to those of CLake+.

6.5 Sensitivity Study

To analyze how SHARP shows superior performance with less area and a lower power budget, we created 36-bit versions of ARK, which are improved with CraterLake's PRNG, our DSU, and our data scheduling methods, under our evaluation settings (ARK$_{36}$-512/ARK$_{36}$-180 with 512/180MB RF$_{\text{main}}$) and incrementally applied additional hardware/software features of SHARP. Fig. 8 shows the results.

Our hierarchical NTTU architecture reduces the area and power consumption of an NTTU by 2.04× and 1.29×, respectively, which leads to 1.09× lower energy on average and a 1.12× smaller chip area than ARK$_{36}$-180. Other features are more effective for improving the delay; two-dimensional BConvU, EWE, and BSGS fine-tuning lead to gradual improvements (decreases) in the average delay by 1.17×, 1.05×, and 1.09×. Overall, the hardware/software features add up to 1.47× (resp., 1.45×) lower EDP and 1.56× (2.44×) lower EDAP values compared to those of ARK$_{36}$-180 (ARK$_{36}$-512).

We also created a larger (251.5mm^2) version of SHARP by deploying eight clusters (8-cluster in Fig. 8). Eight-clustered SHARP partially resolves the compute-bound issue (Section 6.3) and performs 1.40× better with a 41% increased area, achieving 1.48× lower EDP and 1.05× lower EDAP than the original SHARP, which makes it a high-performance alternative. Compared to prior accelerators, eight-clustered SHARP has a similar power budget and performs 16.2× (vs. BTS), 3.34× (CLake+), and 2.19× (ARK) faster in gmean of the workloads.



(a) Energy (b) Delay (c) EDP (d) Area and EDAP

Figure 8: How (a) energy, (b) delay, (c) EDP, (d) area and EDAP change when incrementally applying hierarchical NTTU architecture (+Hierarchy), 2×8 BConvU (+2D-BConv), EWE (+EWE), and BSGS fine-tuning (SHARP) to the baseline 36-bit ARK improved with CraterLake's PRNG, our DSU, and our data scheduling methods. 36-bit ARK with 512MB (resp., 180MB) RF_{main} is denoted ARK_{36}-512 (ARK_{36}-180). Eight-clustered version of SHARP is also shown (8-cluster).

7 RELATED WORK

CPU/GPU acceleration of HE: Many CPU HE libraries [2, 25, 33, 40, 47] are openly available for use. Meanwhile, [18, 63] use CPU SIMD instructions to accelerate HE. However, the CPU performance for HE is limited by its relatively low computing power. The GPU emerges as a viable alternative, with prior studies [3–5, 42, 62, 63, 69] attempting to utilize the massive parallelism and memory bandwidth of GPUs. [4, 5] accelerate the BFV scheme by applying BFV-specific optimizations and discrete Galois transform (DGT) instead of NTT. [62] was the first to support FHE CKKS on a GPU. [62] significantly reduces the off-chip memory access by applying kernel fusion across primary functions. [42] proposed the use of tensor cores in recent NVIDIA GPUs for NTT acceleration, breaking down each of its 32-bit word into 8-bit integers to feed into tensor cores. However, the limited on-chip memory capacity of GPUs still incurs frequent off-chip memory accesses, making FHE workloads memory-bound. Furthermore, as GPUs include hardware units with no use for HE ops, such as floating-point units, they are not the most efficient hardware for accelerating HE ops.

FPGA/ASIC acceleration of HE: Most FPGA-based HE accelerators [54, 71, 72, 92, 93, 98, 103] do not support FHE. [71, 72] propose dedicated functional units for primitive functions only, and [54, 92, 93, 98, 103] target non-bootstrappable parameters; i.e., they only support leveled HE (LHE). FAB [1] is one of the first FPGA-based accelerators to support FHE CKKS, which optimizes the key-switching subroutine to maximize the reuse of ciphertexts given the limited on-chip memory capacity. Poseidon [101] also supports FHE CKKS and deploys dedicated compute cores, such as a high-radix NTT core. FAB and Poseidon offer performance comparable to those of GPUs and ASIC accelerators [70, 94] for some FHE workloads, but bootstrapping on FPGA accelerators is still orders of magnitude slower than more recent ASIC proposals. Meanwhile, Cheetah [90] proposes an ASIC accelerator for privacy-preserving ML based on the Gazelle framework [65], which combines multi-party computation (MPC) [102] with LHE. However, MPC+LHE systems essentially require frequent server-client communication, resulting in expensive network communication overhead. Starting with F1 [94], ASIC accelerators supporting FHE [67, 70, 95] have shown impressive performance improvements, but this comes at the cost of massive chip area and power consumption (see Section 2.4).

8 CONCLUSION

In this paper, we have proposed SHARP, an accelerator for robust and practical fully homomorphic encryption (FHE). We analyzed how the machine word length influences the functionality, performance, and hardware efficiency when executing FHE workloads, determining that the 36-bit word length is a robust and efficient choice. Reducing the memory and communication bandwidth usage is the key challenge for a short-word accelerator. Therefore, we designed a 36-bit SHARP architecture to have a hierarchical structure that ensures that most of the communication occurs locally. Also, we substantially reduced the on-chip memory capacity for improved practicality. The hierarchy and reduced on-chip memory capacity are assisted by our microarchitectural and software enhancements, which enable SHARP to maintain high performance with a much tighter area and power budget. SHARP is highly efficient compared to state-of-the-art ASIC FHE accelerators, showing 2.98× to 22.9× enhanced performance per area, and 2.04× to 19.4× higher performance per watt.

ACKNOWLEDGMENTS

The authors thank the shepherd and anonymous reviewers for their constructive comments. This work was supported in part by Samsung Electronics Co., Ltd. (IO201207-07812-01) and by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) [NO.2021-0-01343, Artificial Intelligence Graduate School Program (Seoul National University)]. The EDA tool was supported by the IC Design Education Center (IDEC), Korea. Jongmin Kim and Donghwan Kim are with the Interdisciplinary Program in Artificial Intelligence, Seoul National University (SNU). Sangpyo Kim, Jaewan Choi, and Jaiyoung Park are with the Department of Intelligence and Information, SNU. Jung Ho Ahn, the corresponding author, is with the Department of Intelligence and Information, the Interdisciplinary Program in Artificial Intelligence, and ICT (Institute of Computer Technology), SNU, Seoul, South Korea.

REFERENCES

[1] Rashmi Agrawal, Leo de Castro, Guowei Yang, Chiraag Juvekar, Rabia Yazicigil, Anantha Chandrakasan, Vinod Vaikuntanathan, and Ajay Joshi. 2023. FAB: An FPGA-based Accelerator for Bootstrappable Fully Homomorphic Encryption. In *HPCA*. 882–895. https://doi.org/10.1109/HPCA56546.2023.10070953

[2] Ahmad Al Badawi, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Ian Quah, Yuriy Polyakov, Saraswathy R.V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. 2022. OpenFHE: Open-Source Fully Homomorphic Encryption Library. In *Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. 53–63. https://doi.org/10.1145/3560827.3563379

[3] Ahmad Al Badawi, Louie Hoang, Chan Fook Mun, Kim Laine, and Khin Mi Mi Aung. 2020. PrivFT: Private and Fast Text Classification with Homomorphic Encryption. *IEEE Access* 8 (2020), 226544–226556. https://doi.org/10.1109/ACCESS.2020.3045465

[4] Ahmad Al Badawi, Yuriy Polyakov, Khin Mi Mi Aung, Bharadwaj Veeravalli, and Kurt Rohloff. 2019. Implementation and Performance Evaluation of RNS Variants of the BFV Homomorphic Encryption Scheme. *IEEE Transactions on Emerging Topics in Computing* 9, 2 (2019), 941–956. https://doi.org/10.1109/TETC.2019.2902799

[5] Ahmad Al Badawi, Bharadwaj Veeravalli, Chan Fook Mun, and Khin Mi Mi Aung. 2018. High-Performance FV Somewhat Homomorphic Encryption on GPUs: An Implementation Using CUDA. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2018, 2 (2018), 143–163. https://doi.org/10.13154/tches.v2018.i2.70-95

[6] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. 2021. Homomorphic Encryption Standard. In *Protecting Privacy through Homomorphic Encryption*. Springer, 31–62. https://doi.org/10.1007/978-3-030-77287-1_2

[7] Renzo Andri, Lukas Cavigelli, Davide Rossi, and Luca Benini. 2018. YodaNN: An Architecture for Ultralow Power Binary-Weight CNN Acceleration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 1 (2018), 48–60. https://doi.org/10.1109/TCAD.2017.2682138

[8] Renzo Andri, Lukas Cavigelli, Davide Rossi, and Luca Benini. 2019. Hyperdrive: A Multi-Chip Systolically Scalable Binary-Weight CNN Inference Engine. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9, 2 (2019), 309–322. https://doi.org/10.1109/JETCAS.2019.2905654

[9] Frederik Armknecht, Colin Boyd, Christopher Carr, Kristian Gjøsteen, Angela Jäschke, Christian A Reuter, and Martin Strand. 2015. A Guide to Fully Homomorphic Encryption. *IACR Cryptology ePrint Archive* 1192 (2015).

[10] Hodjat Asghari Esfeden, Farzad Khorasani, Hyeran Jeon, Daniel Wong, and Nael Abu-Ghazaleh. 2019. CORF: Coalescing Operand Register File for GPUs. In *ASPLOS*. 701–714. https://doi.org/10.1145/3297858.3304026

[11] Jean-Philippe Aumasson. 2019. Too Much Crypto. *IACR Cryptology ePrint Archive* 1492 (2019).

[12] Youngjin Bae, Jung Hee Cheon, Wonhee Cho, Jaehyung Kim, and Taekyung Kim. 2022. META-BTS: Bootstrapping Precision Beyond the Limit. In *ACM Conference on Computer and Communications Security*. 223–234. https://doi.org/10.1145/3548606.3560696

[13] David H. Bailey. 1989. FFTs in External or Hierarchical Memory. In *ACM/IEEE Conference on Supercomputing*. 234–242. https://doi.org/10.1145/76263.76288

[14] Jean-Claude Bajard, Julien Eynard, M. Anwar Hasan, and Vincent Zucca. 2016. A Full RNS Variant of FV Like Somewhat Homomorphic Encryption Schemes. In *Selected Areas in Cryptography*. 423–442. https://doi.org/10.1007/978-3-319-69453-5_23

[15] Paul Barrett. 1986. Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor. In *Annual International Conference on the Theory and Application of Cryptographic Techniques*. 311–323. https://doi.org/10.5555/36664.36688

[16] Laszlo A. Belady. 1966. A Study of Replacement Algorithms for a Virtual-Storage Computer. *IBM Systems Journal* 5, 2 (1966), 78–101. https://doi.org/10.1147/sj.52.0078

[17] Fabian Boemer, Anamaria Costache, Rosario Cammarota, and Casimir Wierzynski. 2019. NGraph-HE2: A High-Throughput Framework for Neural Network Inference on Encrypted Data. In *Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. 45–56. https://doi.org/10.1145/3338469.3358944

[18] Fabian Boemer, Sejun Kim, Gelila Seifu, Fillipe D. M. de Souza, and Vinodh Gopal. 2021. Intel HEXL: Accelerating Homomorphic Encryption with Intel AVX512-IFMA52. In *Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. 57–62. https://doi.org/10.1145/3474366.3486926

[19] Jean-Philippe Bossuat, Christian Mouchet, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. 2021. Efficient Bootstrapping for Approximate Homomorphic Encryption with Non-sparse Keys. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 587–617. https://doi.org/10.1007/978-3-030-77870-5_21

[20] Jean-Philippe Bossuat, Juan Troncoso-Pastoriza, and Jean-Pierre Hubaux. 2022. Bootstrapping for Approximate Homomorphic Encryption with Negligible Failure-Probability by Using Sparse-Secret Encapsulation. In *Applied Cryptography and Network Security*. 521–541. https://doi.org/10.1007/978-3-031-09234-3_26

[21] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2014. (Leveled) Fully Homomorphic Encryption without Bootstrapping. *ACM Transactions on Computing Theory* 6, 3 (2014), 1–36. https://doi.org/10.1145/2633600

[22] Zvika Brakerski and Vinod Vaikuntanathan. 2014. Efficient Fully Homomorphic Encryption from (Standard) LWE. *SIAM J. Comput.* 43, 2 (2014), 831–871. https://doi.org/10.1137/120868669

[23] Jonathan Chang, Yen-Huei Chen, Wei-Min Chan, Sahil Preet Singh, Hank Cheng, Hidehiro Fujiwara, Jih-Yu Lin, Kao-Cheng Lin, John Hung, Robin Lee, Hung-Jen Liao, Jhon-Jhy Liaw, Quincy Li, Chih-Yung Lin, Mu-Chi Chiang, and Shien-Yang Wu. 2017. A 7nm 256Mb SRAM in High-K Metal-Gate FinFET Technology with Write-Assist Circuitry for Low-VMIN Applications. In *IEEE International Solid-State Circuits Conference*. 206–207. https://doi.org/10.1109/ISSCC.2017.7870333

[24] Hao Chen, Ilaria Chillotti, and Yongsoo Song. 2019. Improved Bootstrapping for Approximate Homomorphic Encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 34–54. https://doi.org/10.1007/978-3-030-17656-3_2

[25] Hao Chen, Kim Laine, and Rachel Player. 2017. Simple Encrypted Arithmetic Library - SEAL v2.1. In *Financial Cryptography and Data Security*. 3–18. https://doi.org/10.1007/978-3-319-70278-0_1

[26] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In *ISCA*. 367–379. https://doi.org/10.1109/ISCA.2016.40

[27] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. 2018. A Full RNS Variant of Approximate Homomorphic Encryption. In *Selected Areas in Cryptography*. 347–368. https://doi.org/10.1007/978-3-030-10970-7_16

[28] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *International Conference on the Theory and Applications of Cryptology and Information Security*. 409–437. https://doi.org/10.1007/978-3-319-70694-8_15

[29] Jung Hee Cheon, Yongha Son, and Donggeon Yhee. 2022. Practical FHE Parameters against Lattice Attacks. *Journal of the Korean Mathematical Society* 59, 1 (2022), 35–51. https://doi.org/10.4134/JKMS.j200650

[30] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2020. TFHE: Fast Fully Homomorphic Encryption Over the Torus. *Journal of Cryptology* 33, 1 (2020), 34–91. https://doi.org/10.1007/s00145-019-09319-x

[31] Lawrence T Clark, Vinay Vashishtha, Lucian Shifren, Aditya Gujja, Saurabh Sinha, Brian Cline, Chandarasekaran Ramamurthy, and Greg Yeric. 2016. ASAP7: A 7-nm FinFET Predictive Process Design Kit. *Microelectronics Journal* 53 (2016), 105–115. https://doi.org/10.1016/j.mejo.2016.04.006

[32] James W. Cooley and John W. Tukey. 1965. An Algorithm for the Machine Calculation of Complex Fourier Series. *Math. Comp.* 19, 90 (1965), 297–301. https://doi.org/10.1090/s0025-5718-1965-0178586-1

[33] CryptoLab Inc. 2018. HEAAN v2.1. https://github.com/snucrypto/HEAAN

[34] Benjamin R. Curtis and Rachel Player. 2019. On the Feasibility and Impact of Standardising Sparse-secret LWE Parameter Sets for Homomorphic Encryption. In *Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. 1–10. https://doi.org/10.1145/3338469.3358940

[35] William J. Dally, Francois Labonte, Abhishek Das, Patrick Hanrahan, Jung Ho Ahn, Jayanth Gummaraju, Mattan Erez, Nuwan Jayasena, Ian Buck, Timothy J. Knight, and Ujval J. Kapasi. 2003. Merrimac: Supercomputing with Streams. In *SC*. https://doi.org/10.1145/1048935.1050187

[36] Leo de Castro, Rashmi Agrawal, Rabia Tugce Yazicigil, Anantha P. Chandrakasan, Vinod Vaikuntanathan, Chiraag Juvekar, and Ajay Joshi. 2021. Does Fully Homomorphic Encryption Need Compute Acceleration? *arXiv preprint arXiv:2112.06396* (2021). https://doi.org/10.48550/arXiv.2112.06396

[37] Li Deng. 2012. The MNIST Database of Handwritten Digit Images for Machine Learning Research. *IEEE Signal Processing Magazine* 29, 6 (2012), 141–142. https://doi.org/10.1109/MSP.2012.2211477

[38] Yucong Duan, Guohua Fu, Nianjun Zhou, Xiaobing Sun, Nanjangud C. Narendra, and Bo Hu. 2015. Everything as a Service (XaaS) on the Cloud: Origins, Current and Future Trends. In *IEEE International Conference on Cloud Computing*. 621–628. https://doi.org/10.1109/CLOUD.2015.88

[39] Léo Ducas and Daniele Micciancio. 2015. FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 617–640. https://doi.org/10.1007/978-3-662-46800-5_24

[40] EPFL-LDS and Tune Insight SA. 2022. Lattigo v4. https://github.com/tuneinsight/lattigo

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

[42] Shengyu Fan, Zhiwei Wang, Weizhi Xu, Rui Hou, Dan Meng, and Mingzhe Zhang. 2023. TensorFHE: Achieving Practical Computation on Encrypted Data Using GPGPU. In *HPCA*. 922–934. https://doi.org/10.1109/HPCA56546.2023.10071017

[43] Hidehiro Fujiwara, Yi-Hsin Nien, Chih-Yu Lin, Hsien-Yu Pan, Hao-Wen Hsu, Shin-Rung Wu, Yao-Yi Liu, Yen-Huei Chen, Hung-Jen Liao, and Jonathan Chang. 2021. A 5nm 5.7GHz@1.0V and 1.3GHz@0.5V 4kb Standard-Cell-Based Two-Port Register File with a 16T Bitcell with No Half-Selection Issue. In *IEEE International Conference on Solid-State Circuits Conference*. 1–3. https://doi.org/

10.1109/ISSCC42613.2021.9366000

[44] W. Morven Gentleman and Gordon Sande. 1966. Fast Fourier Transforms: For Fun and Profit. In *AFIPS Fall Joint Computer Conference*. 563–578. https://doi.org/10.1145/1464291.1464352

[45] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep Learning with Limited Numerical Precision. In *International Conference on Machine Learning*. 1737–1746.

[46] Shai Halevi and Victor Shoup. 2018. Faster Homomorphic Linear Transformations in HElib. In *Annual International Cryptology Conference*. 93–120. https://doi.org/10.1007/978-3-319-96884-1_4

[47] Shai Halevi and Victor Shoup. 2020. Design and implementation of HElib: a homomorphic encryption library. *IACR Cryptology ePrint Archive* 1481 (2020).

[48] Kyoohyung Han, Seungwan Hong, Jung Hee Cheon, and Daejun Park. 2019. Logistic Regression on Homomorphic Encrypted Data at Scale. In *AAAI Conference on Artificial Intelligence*. 9466–9471. https://doi.org/10.1609/aaai.v33i01.33019466

[49] Kyoohyung Han and Dohyeong Ki. 2020. Better Bootstrapping for Approximate Homomorphic Encryption. In *Cryptographers' Track at the RSA Conference*. 364–390. https://doi.org/10.1007/978-3-030-40186-3_16

[50] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*. 770–778. https://doi.org/10.1109/cvpr.2016.90

[51] Ron Ho, Kenneth Mai, and Mark Horowitz. 2001. The Future of Wires. *Proc. IEEE* 89, 4 (2001), 490–504. https://doi.org/10.1109/5.920580

[52] Seungwan Hong, Seunghong Kim, Jiheon Choi, Younho Lee, and Jung Hee Cheon. 2021. Efficient Sorting of Homomorphic Encrypted Data With k-Way Sorting Network. *IEEE Transactions on Information Forensics and Security* 16 (2021), 4389–4404. https://doi.org/10.1109/TIFS.2021.3106167

[53] Jan Hoogerbrugge and Henk Corporaal. 1994. Transport-Triggering vs. Operation-Triggering. In *International Conference on Compiler Construction*. 435–449. https://doi.org/10.1007/3-540-57877-3_29

[54] Xiao Hu, Minghao Li, Jing Tian, and Zhongfeng Wang. 2022. Efficient Homomorphic Convolution Designs on FPGA for Secure Inference. *IEEE Transactions on VLSI Systems* 30 (2022), 1691–1704. https://doi.org/10.1109/TVLSI.2022.3197895

[55] IEEE. 2018. *International Roadmap for Devices and Systems: 2018.* Technical Report. https://irds.ieee.org/editions/2018/

[56] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In *IEEE Conference on Computer Vision and Pattern Recognition*. 2704–2713. https://doi.org/10.1109/cvpr.2018.00286

[57] JEDEC. 2021. *High Bandwidth Memory (HBM) DRAM.* Technical Report JESD235D.

[58] JEDEC. 2022. *High Bandwidth Memory DRAM (HBM3).* Technical Report JESD238.

[59] W.C. Jeong, S. Maeda, H.J. Lee, K.W. Lee, T.J. Lee, D.W. Park, B.S. Kim, J.H. Do, T. Fukai, D.J. Kwon, K.J. Nam, W.J. Rim, M.S. Jang, H.T. Kim, Y.W. Lee, J.S. Park, E.C. Lee, D.W. Ha, C.H. Park, H.J. Cho, S.M. Jung, and H.K. Kang. 2018. True 7nm Platform Technology featuring Smallest FinFET and Smallest SRAM cell by EUV, Special Constructs and 3rd Generation Single Diffusion Break. In *IEEE Symposium on VLSI Technology*. 59–60. https://doi.org/10.1109/VLSIT.2018.8510682

[60] Norman P. Jouppi, Doe Hyun Yoon, Matthew Ashcraft, Mark Gottscho, Thomas B. Jablin, George Kurian, James Laudon, Sheng Li, Peter C. Ma, Xiaoyu Ma, Thomas Norrie, Nishant Patil, Sushma Prasad, Cliff Young, Zongwei Zhou, and David A. Patterson. 2021. Ten Lessons From Three Generations Shaped Google's TPUv4i: Industrial Product. In *ISCA*. 1–14. https://doi.org/10.1109/ISCA52012.2021.00010

[61] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *ISCA*. 1–12. https://doi.org/10.1145/3079856.3080246

[62] Wonkyung Jung, Sangpyo Kim, Jung Ho Ahn, Jung Hee Cheon, and Younho Lee. 2021. Over 100x Faster Bootstrapping in Fully Homomorphic Encryption through Memory-centric Optimization with GPUs. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021, 4 (2021), 114–148. https://doi.org/10.46586/tches.v2021.i4.114-148

[63] Wonkyung Jung, Eojin Lee, Sangpyo Kim, Jongmin Kim, Namhoon Kim, Keewoo Lee, Chohong Min, Jung Hee Cheon, and Jung Ho Ahn. 2021. Accelerating Fully Homomorphic Encryption Through Architecture-Centric Analysis and Optimization. *IEEE Access* 9 (2021), 98772–98789. https://doi.org/10.1109/ACCESS.2021.3096189

[64] Charanjit S. Jutla and Nathan Manohar. 2022. Sine Series Approximation of the Mod Function for Bootstrapping of Approximate HE. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 491–520. https://doi.org/10.1007/978-3-031-06944-4_17

[65] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. {GAZELLE}: A Low Latency Framework for Secure Neural Network Inference. In *USENIX Security Symposium*. 1651–1669.

[66] Andrey Kim, Antonis Papadimitriou, and Yuriy Polyakov. 2022. Approximate Homomorphic Encryption with Reduced Approximation Error. In *Cryptographers' Track at the RSA Conference*. 120–144. https://doi.org/10.1007/978-3-030-95312-6_6

[67] Jongmin Kim, Gwangho Lee, Sangpyo Kim, Gina Sohn, Minsoo Rhu, John Kim, and Jung Ho Ahn. 2022. ARK: Fully Homomorphic Encryption Accelerator with Runtime Data Generation and Inter-Operation Key Reuse. In *MICRO*. 1237–1254. https://doi.org/10.1109/MICRO56248.2022.00086

[68] Miran Kim, Xiaoqian Jiang, Kristin Lauter, Elkhan Ismayilzada, and Shayan Shams. 2022. Secure human action recognition by encrypted neural network inference. *Nature communications* 13, 1 (2022), 1–13. https://doi.org/10.1038/s41467-022-32168-5

[69] Sangpyo Kim, Wonkyung Jung, Jaiyoung Park, and Jung Ho Ahn. 2020. Accelerating Number Theoretic Transformations for Bootstrappable Homomorphic Encryption on GPUs. In *IEEE International Symposium on Workload Characterization*. 264–275. https://doi.org/10.1109/IISWC50251.2020.00033

[70] Sangpyo Kim, Jongmin Kim, Michael Jaemin Kim, Wonkyung Jung, John Kim, Minsoo Rhu, and Jung Ho Ahn. 2022. BTS: An Accelerator for Bootstrappable Fully Homomorphic Encryption. In *ISCA*. 711–725. https://doi.org/10.1145/3470496.3527415

[71] Sunwoong Kim, Keewoo Lee, Wonhee Cho, Jung Hee Cheon, and Rob A. Rutenbar. 2019. FPGA-based Accelerators of Fully Pipelined Modular Multipliers for Homomorphic Encryption. In *International Conference on ReConFigurable Computing and FPGAs*. 1–8. https://doi.org/10.1109/ReConFig48160.2019.8994793

[72] Sunwoong Kim, Keewoo Lee, Wonhee Cho, Yujin Nam, Jung Hee Cheon, and Rob A. Rutenbar. 2020. Hardware Architecture of a Number Theoretic Transform for a Bootstrappable RNS-based Homomorphic Encryption Scheme. In *IEEE International Symposium on Field-Programmable Custom Computing Machines*. 56–64. https://doi.org/10.1109/FCCM48280.2020.00017

[73] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2020. Spectre Attacks: Exploiting Speculative Execution. *Commun. ACM* 63, 7 (2020), 93–101. https://doi.org/10.1145/3399742

[74] Alex Krizhevsky and Geoffrey Hinton. 2009. *Learning Multiple Layers of Features from Tiny Images.* Technical Report. University of Toronto.

[75] Eunsang Lee, Joon-Woo Lee, Junghyun Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and Woosuk Choi. 2022. Low-Complexity Deep Convolutional Neural Networks on Fully Homomorphic Encryption Using Multiplexed Parallel Convolutions. In *International Conference on Machine Learning*. 12403–12422.

[76] Junghyun Lee, Eunsang Lee, Joon-Woo Lee, Yongjune Kim, Young-Sik Kim, and Jong-Seon No. 2021. Precise Approximation of Convolutional Neural Networks for Homomorphically Encrypted Data. *arXiv preprint arXiv:2105.10879* (2021).

[77] Joon-Woo Lee, Eunsang Lee, Yongwoo Lee, Young-Sik Kim, and Jong-Seon No. 2021. High-Precision Bootstrapping of RNS-CKKS Homomorphic Encryption Using Optimal Minimax Polynomial Approximation and Inverse Sine Function. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 618–647. https://doi.org/10.1007/978-3-030-77870-5_22

[78] Joon-Woo Lee, Hyungchul Kang, Yongwoo Lee, Woosuk Choi, Jieun Eom, Maxim Deryabin, Eunsang Lee, Junghyun Lee, Donghoon Yoo, Young-Sik Kim, and Jong-Seon No. 2022. Privacy-Preserving Machine Learning With Fully Homomorphic Encryption for Deep Neural Network. *IEEE Access* 10 (2022), 30039–30054. https://doi.org/10.1109/ACCESS.2022.3159694

[79] Yongwoo Lee, Joon-Woo Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and HyungChul Kang. 2022. High-Precision Bootstrapping for Approximate Homomorphic Encryption by Error Variance Minimization. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 551–580. https://doi.org/10.1007/978-3-031-06944-4_19

[80] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. 2009. McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures. In *MICRO*. 469–480. https://doi.org/10.1145/1669112.1669172

[81] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. 2016. Fixed Point Quantization of Deep Convolutional Networks. In *International Conference on Machine Learning*. 2849–2858.

[82] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, Mike Hamburg, and Raoul Strackx. 2020. Meltdown: Reading Kernel Memory from User Space. *Commun. ACM* 63, 6 (2020), 46–56. https://doi.org/10.1145/3357033

[83] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2010. On Ideal Lattices and Learning with Errors over Rings. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 1–35. https://doi.org/10.1007/978-3-642-13190-5_1

[84] Peter L. Montgomery. 1985. Modular Multiplication without Trial Division. *Math. Comp.* 44, 170 (1985), 519–521. https://doi.org/10.1090/S0025-5718-1985-0777282-X

[85] Toufique Morshed, Md Momin Al Aziz, and Noman Mohammed. 2020. CPU and GPU Accelerated Fully Homomorphic Encryption. In *IEEE International Symposium on Hardware Oriented Security and Trust*. 142–153. https://doi.org/10.1109/HOST45689.2020.9300288

[86] 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. Dilliway, K. Donegan, M. Eller, Y. Fan, Q. Fang, A. Gassaria, R. Gauthier, S. Ghosh, G. Gifford, T. Gordon, M. Gribelyuk, G. Han, J.H. 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, T. Kim, W. Kim, R. Krishnan, B. Krishnan, K. Anil, M. Kumar, J. Lee, R. Lee, J. Lemon, S.L. Liew, P. Lindo, M. Lingalugari, M. Lipinski, P. Liu, J. Liu, S. Lucarini, W. Ma, E. Maciejewski, S. Madisetti, 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.C. 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.B. 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. 2017. A 7nm CMOS Technology Platform for Mobile and High Performance Compute Application. In *IEEE International Electron Devices Meeting*. 29.5.1–29.5.4. https://doi.org/10.1109/IEDM.2017.8268476

[87] Wilfried Oed and Otto Lange. 1985. On the Effective Bandwidth of Interleaved Memories in Vector Processor Systems. *IEEE Trans. Comput.* C-34, 10 (1985), 949–957. https://doi.org/10.1109/TC.1985.6312199

[88] Mike O'Connor, Niladrish Chatterjee, Donghyuk Lee, John Wilson, Aditya Agrawal, Stephen W Keckler, and William J Dally. 2017. Fine-Grained DRAM: Energy-Efficient DRAM for Extreme Bandwidth Systems. In *MICRO*. 41–54. https://doi.org/10.1145/3123939.3124545

[89] Jestine Paul, Meenatchi Sundaram Muthu Selva Annamalai, William Ming, Ahmad Al Badawi, Bharadwaj Veeravalli, and Khin Mi Mi Aung. 2021. Privacy-Preserving Collective Learning With Homomorphic Encryption. *IEEE Access* 9 (2021), 132084–132096. https://doi.org/10.1109/ACCESS.2021.3114581

[90] Brandon Reagen, Woo-Seok Choi, Yeongil Ko, Vincent T. Lee, Hsien-Hsin S. Lee, Gu-Yeon Wei, and David Brooks. 2021. Cheetah: Optimizing and Accelerating Homomorphic Encryption for Private Inference. In *HPCA*. 26–39. https://doi.org/10.1109/HPCA51647.2021.00013

[91] Albert Reuther, Peter Michaleas, Michael Jones, Vijay Gadepally, Siddharth Samsi, and Jeremy Kepner. 2020. Survey of Machine Learning Accelerators. In *IEEE High Performance Extreme Computing Conference*. 1–12. https://doi.org/10.1109/HPEC43674.2020.9286149

[92] M. Sadegh Riazi, Kim Laine, Blake Pelton, and Wei Dai. 2020. HEAX: An Architecture for Computing on Encrypted Data. In *ASPLOS*. 1295–1309. https://doi.org/10.1145/3373376.3378523

[93] Sujoy Sinha Roy, Furkan Turan, Kimmo Järvinen, Frederik Vercauteren, and Ingrid Verbauwhede. 2019. FPGA-Based High-Performance Parallel Architecture for Homomorphic Computing on Encrypted Data. In *HPCA*. 387–398. https://doi.org/10.1109/HPCA.2019.00052

[94] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Srinivas Devadas, Ronald Dreslinski, Christopher Peikert, and Daniel Sanchez. 2021. F1: A Fast and Programmable Accelerator for Fully Homomorphic Encryption. In *MICRO*. 238–252. https://doi.org/10.1145/3466752.3480070

[95] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Nathan Manohar, Nicholas Genise, Srinivas Devadas, Karim Eldefrawy, Chris Peikert, and Daniel Sanchez. 2022. CraterLake: A Hardware Accelerator for Efficient Unbounded Computation on Encrypted Data. In *ISCA*. 173–187. https://doi.org/10.1145/3470496.3527393

[96] Alireza Shafaei, Yanzhi Wang, Xue Lin, and Massoud Pedram. 2014. FinCACTI: Architectural Analysis and Modeling of Caches with Deeply-Scaled FinFET Devices. In *IEEE Computer Society Annual Symposium on VLSI*. 290–295. https://doi.org/10.1109/ISVLSI.2014.94

[97] Taejoong Song, Jonghoon Jung, Woojin Rim, Hoonki Kim, Yongho Kim, Changnam Park, Jeongho Do, Sunghyun Park, Sungwee Cho, Hyuntaek Jung, Bongjae Kwon, Hyun-Su Choi, Jaeseung Choi, and Jong Shik Yoon. 2018. A 7nm FinFET SRAM Using EUV Lithography with Dual Write-Driver-Assist Circuitry for Low-Voltage Applications. In *IEEE International Solid-State Circuits Conference*. 198–200. https://doi.org/10.1109/ISSCC.2018.8310252

[98] Furkan Turan, Sujoy Sinha Roy, and Ingrid Verbauwhede. 2020. HEAWS: An Accelerator for Homomorphic Encryption on the Amazon AWS FPGA. *IEEE Trans. Comput.* 69, 8 (2020), 1185–1196. https://doi.org/10.1109/TC.2020.2988765

[99] Vernam Group. 2019. CUDA-Accelerated Fully Homomorphic Encryption Library. https://github.com/vernamlab/cuFHE

[100] Shien-Yang Wu, C.Y. Lin, M.C. Chiang, J.J. Liaw, J.Y. Cheng, S.H. Yang, C.H. Tsai, P.N. Chen, T. Miyashita, C.H. Chang, V.S. Chang, K.H. Pan, J.H. Chen, Y.S. Mor, K.T. Lai, C.S. Liang, H.F. Chen, S.Y. Chang, C.J. Lin, C.H. Hsieh, R.F. Tsui, C.H. Yao, C.C. Chen, R. Chen, C.H. Lee, H.J. Lin, C.W. Chang, K.W. Chen, M.H. Tsai, K.S. Chen, Y. Ku, and S.M. Jang. 2016. A 7nm CMOS Platform Technology Featuring 4th Generation FinFET Transistors with a 0.027um2 High Density 6-T SRAM cell for Mobile SoC Applications. In *IEEE International Electron Devices Meeting*. 2.6.1–2.6.4. https://doi.org/10.1109/IEDM.2016.7838333

[101] Yinghao Yang, Huaizhi Zhang, Shengyu Fan, Hang Lu, Mingzhe Zhang, and Xiaowei Li. 2023. Poseidon: Practical Homomorphic Encryption Accelerator. In *HPCA*. 870–881. https://doi.org/10.1109/HPCA56546.2023.10070984

[102] Andrew Chi-Chih Yao. 1986. How to Generate and Exchange Secrets. In *IEEE Symposium on Foundations of Computer Science*. 162–167. https://doi.org/10.1109/SFCS.1986.25

[103] Yilan Zhu, Xinyao Wang, Lei Ju, and Shanqing Guo. 2023. FxHENN: FPGA-based Acceleration Framework for Homomorphic Encrypted CNN Inference. In *HPCA*. 896–907. https://doi.org/10.1109/HPCA56546.2023.10071133