Poseidon-NDP: Practical Fully Homomorphic Encryption Accelerator Based on Near Data Processing Architecture

Yinghao Yang, Hang Lu, and Xiaowei Li, *Senior Member, IEEE*

***Abstract*—With the development of the important solution for privacy computing—fully homomorphic encryption (FHE), the explosion of data size, and computing intensity in FHE applications brings enormous challenges to the hardware design. In this article, we propose a novel co-design scheme for FHE acceleration named "Poseidon-NDP," which focuses on improving the efficiency of the hardware resource and the bandwidth. Specifically, we investigate the special implications of the hardware imposed by the FHE applications. It empirically shows that the FHE performance is suffered from both the intractable data movement and the computation bottleneck. Besides, we also introduce the opportunity and the challenges of accelerating FHE on near data processing (NDP) architecture. Based on such analysis, we propose an optimized technique called "NTT-fusion" to simplify the FHE operator and reduce its hardware overhead. Then, we design the accelerator based on the simplified operator to achieve maximized data and computation parallelism with limited hardware resources. Additionally, we evaluate Poseidon-NDP with 4 domain-specific FHE applications on the SmartSSD, which is a practical NDP device. The empirical studies show that the efficient co-design enables Poseidon-NDP vastly superior to the state-of-the-art FHE acceleration techniques: 1) up to 217×/84× speedup over CPU and high-performance GPUs for the number theoretic transform; 2) up to 3.7×/29× higher-speedup/energy delay product (EDP) over the SOTA FPGA accelerator for the FHE applications; and 3) up to 4.9× higher-bandwidth utilization over CPU due to the NDP-based architecture.**

***Index Terms*—FPGA accelerator, fully homomorphic encryption (FHE), near data processing (NDP), privacy computing.**

I. Introduction

As THE need for privacy protection grows, the technology of privacy computing is significant for scenarios involving sensitive data, i.e., personal credit records, medical history, financial records, and so on. Fully homomorphic encryption (FHE), as one of the mainstream privacy-preserving schemes, can guarantee data security without the trusted third party and compute on the encrypted data. Fig. 1 shows the concept of FHE, the client encrypts their data locally using the secret key and sent to the server providing the computing power. After computing, the server returns the encrypted result and the client decrypts it to obtain the actual value. There is no private data leakage in the whole computing process.

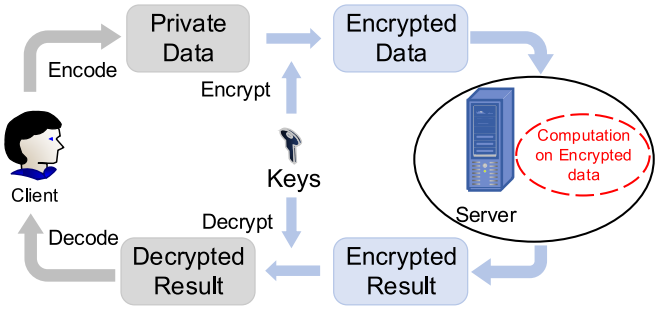

Fig. 1. General concept of FHE.

With the rapid development of FHE in recent years, there are several FHE schemes are proposed by researchers. The first FHE scheme is brought forward by Gentry in 2009 [22], which proposed a generic method for constructing FHE. It supports any number of homomorphic additions and multiplications with higher-computational overhead. To improve Gentry's work, several FHE schemes subsequently emerged, such as BGV [8], BFV [21], TFHE [15], and CKKS [14]. BGV and BFV aim to accurately evaluate arithmetic circuits, while CKKS is dedicated to approximate calculations. Unlike the BFV, BGV, and CKKS, TFHE is an implementation based on the boolean circuit and specializes in encrypted bitwise operations. Therefore, different FHE schemes are available for different scenarios.

Although a collection of FHE algorithms have been proposed, it is still difficult to apply them in practical scenarios. For the proposed FHE solutions, there is a big execution efficiency gap between plaintext and ciphertext computation. The main reason is that FHE will hugely expand the data size and increase the computation complexity. As will be proved in Section II, the length of the ciphertext can be expanded tens of thousands of times relative to the plaintext, and the runtime of the ciphertext increases to even thousands of times relative to the plaintext.

Manuscript received 23 December 2022; revised 21 April 2023 and 15 June 2023; accepted 20 June 2023. Date of publication 4 July 2023; date of current version 22 November 2023. This work was supported in part by the National Natural Science Foundation of China under Grant 62172387, and in part by the Youth Innovation Promotion Association of Chinese Academy of Sciences (CAS) under Grant 2021098. This article was recommended by Associate Editor J. Rajendran. *(Corresponding author: Hang Lu.)*

Yinghao Yang is with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100045, China, and also with the University of Chinese Academy of Sciences, Beijing 100190, China.

Hang Lu and Xiaowei Li are with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100045, China, also with the University of Chinese Academy of Sciences, Beijing 100190, China, and also with the Shanghai Innovation Center for Processor Technologies, Beijing 100190, China (e-mail: luhang@ict.ac.cn).

Digital Object Identifier 10.1109/TCAD.2023.3292211

Landscape of Prior Work: In order to alleviate the formidable burden imposed on the existing CPU platforms, recent works devoted to designing specialized FHE accelerator architectures to offload the FHE computations from CPU, which can be categorized into two main aspects:

1) accelerating a particular FHE scheme but cannot support other FHE schemes well (i.e., HEAX [39] for CKKS, Hepcloud [40], [41] for BFV);
2) accelerating a series of key operators in FHE, such as number theoretic transform (NTT), inverse NTT (INTT), modular multiplication (ModMult), and modular addition (ModAdd), and thus supports a wide range of FHE schemes [10], [11], [19], [49].

Prior works have theoretically achieved the speedup from tens to even hundreds of times compared with CPU. However, the memory access bandwidth is always ignored or impractically assumed constant in these works. For example, HEAX [39] reports 76× speedup to GPU for NTT while ignoring the actual bandwidth requirement that could exceed 450 GB/s in theory. Kim et al. [33], [34] achieved the 118× speedup excluding the slowdown caused by the bandwidth limitation. Other works focus on accelerating the FHE key operators in hardware but achieve limited performance improvement. For example, Roy et al. [40], [41] designed the BFV accelerator based on the Barrett Reduction algorithm [24] and achieved 1.3× speedup over CPU. Cao et al. [9] applied the algorithm in [24] to accelerate ModMult on large numbers but it is not suitable for the widely used residual number system (RNS)-based FHE [7], [13], [23]. Doröz et al. [20] aimed at the fast decryption/encryption operation and achieved 20% acceleration compared with the software decryption. In addition, some works also resort to GPUs for the FHE acceleration [5], [6], [29], [47], [48]. Multiple FHE schemes are tested on different GPU platforms to explore the most matched GPU architecture in these works. There are also some ASIC-based FHE accelerator works [30], [32], [43], [44]. They achieve a remarkable acceleration ratio for the overall FHE applications. However, they rely on expensive large-capacity on-chip SRAM buffers, even up to 512 MB. Some open-source software libraries [16], [37], [46] are also developed for accelerating FHE on GPUs. However, compared with the FPGA platform, GPU exhibits relatively higher-power consumption.

In this article, we propose a practical FHE accelerator - Poseidon-NDP, which is based on the near data processing (NDP) architecture. Poseidon-NDP fully tackles the ciphertext flood spawned by the plaintext encryption, and sufficiently utilizes the abundant bandwidth brought by NDP to accommodate the frequent and large-volume data movement. As the key operator, the computation pattern of NTT follows the uniform and recursive "multiply-and-accumulation," providing the opportunity to fuse multiple NTT operations together to alleviate the computation burden from the software side. The contributions of this article are listed as follows.

1) We analyze the FHE acceleration challenges and the possible opportunities. Such analysis proves that the software/hardware co-design is imperative for the FHE acceleration.
2) From the software perspective, we propose an algorithm optimization scheme called NTT-fusion to reduce the computation intensity of FHE. It collaborates with the accelerator hardware to maximize the overall performance on the power-constrained NDP platform.
3) We propose an NDP-based homomorphic encryption accelerator—Poseidon-NDP. It focus on the common operators of the FHE schemes and leverages the broad bandwidth naturally provided by NDP to tackle the ciphertext flood and support different FHE applications.
4) We use the commercial NDP platform—SmartSSD to evaluate Poseidon-NDP. Although the programmable logic in SmartSSD is less powerful, according to our evaluations, Poseidon-NDP still outperforms CPU, GPU and the state-of-the-art FPGA-based accelerator by 72× ∼ 158×, 35× ∼ 87×, and 2.5× ∼ 3.7×, respectively.

II. Background and Motivation

A. Homomorphic Encryption Preliminaries

As mentioned in the previous section, the classic FHE algorithms include CKKS, BGV, BFV, and TFHE. These schemes are all based on the ring-based learning with errors (RLWE) problem [34]—an augmented version of learning with errors (LWE) by introducing the "polynomial rings." The hardness of RLWE can be regarded as a traditional NP-hard lattice problem: given randomly generated polynomials $a \in R_q$, where $R_q = Z_q/f(x)$ and $f(x)$ is an irreducible polynomial with degree n; let $s \in R_q$ and $e \in R_q$ abiding by the χ distribution; define $b_i = a_i s + e_i \in R_q$, so solving this problem is to compute s from several (a_i, b_i) groups, which is regarded as NP-hard. The a_i and b_i could be regarded as the plaintext and ciphertext, expressed by polynomials. The s indicates the secret key, which is very difficult to be deduced from the given a_i and b_i.

The key operators of FHE can be summarized as follows.

① *ModAdd*$(ct_{0,0}, ct_{0,1})$*:* For two ciphertext $ct_0 = (ct_{0,0}, ct_{0,1})$ and $ct_1 = (ct_{1,0}, ct_{1,1})$, their addition will return a ciphertext $ct = (ct_{0,0} + ct_{1,0}, ct_{0,1} + ct_{1,1})$, where the operator "+" represents the elementwise addition of the two polynomials. Since FHE is performed on the "polynomial ring," the result of the ciphertext addition equals to the modular addition ("ModAdd" for clarity). For example, $a+b = (a_i+b_i)\ mod\ q$, $(a, b \in R_q)$, where q is the modulus.

② *ModMult* (ct_0, ct_1)*:* Compared with ModAdd, ModMult of two ciphertext polynomials is more complicated. Taking CKKS [42] as an example, computing ModMult (ct_0, ct_1) requires $\widetilde{ct} = (d_0, d_1, d_2) \bmod q = (ct_{0,0} \cdot ct_{1,0}, ct_{0,0} \cdot ct_{1,1} + ct_{0,1} \cdot ct_{1,0}, ct_{0,1} \cdot ct_{1,1}) \bmod q$. The final result is $ct = (d_0, d_1) + p^{-1} \cdot d_2 \cdot rlk)$. *rlk* is the relinearization key represented as $rlk = (b, a) \in \mathbb{R}^2_{pq} = (-a \cdot s + e + p \cdot s^2, a) \bmod p \cdot q$, where s is the secrete key and p is a special integer that relies on the *rlk* setting. The corresponding p^{-1} is the inverse of p in the modulus pq, represented as $p^{-1} \cdot p \equiv 1 (\bmod\ pq)$. From these deductions, we can see that the ciphertext multiplication involves both multiplication and addition of the polynomials. The addition is identical to the elementwise ModAdd

TABLE I
FHE BENCHMARKS WE USE FOR EVALUATION. WE COLLECT FOUR FHE APPLICATIONS RANGING FROM THE ENCRYPTED LOGISTIC REGRESSION TRAIN/TEST, DNN INFERENCE TO THE MATRIXMULT, AND SO ON. THE FHE ALGORITHMS INCLUDE CKKS AND BFV WHICH ARE THE MOST WIDELY USED ALGORITHMS IN PRACTICE

Characteristics Applications	**HE Type**	**Poly Modulus Degree**	**Mult. Depth**	**Coefficient Modulus**	**Input Data Size (Unenc. / Enc.)**	**Expansion Ratio (×)**
Enc. LR (Train)	CKKS	2^{13}	6	206 bits	30.5 KB / **1.33 GB**	51,753.7
Enc. LR (Test)	CKKS	2^{12}	1	100 bits	13.05 KB / **41.76 MB**	3,975.9
MNIST Evaluation	CKKS	2^{13}	6	218 bits	29.9 KB / **546.8 GB**	18,732.4
MatrixMult	BFV	2^{13}	3	160 bits	0.076 KB / **75 MB**	3,296.8

TABLE II
RUNTIME AND BANDWIDTH DEMAND ANALYSIS. WE USE INTEL V-TUNE TOOL [26] TO PROFILE THE ENCRYPTED APPLICATION EXECUTION ON THE OFF-THE-SHELF X86 SERVER PRODUCT. THE CONCRETE SPECS: 10-CORE/20-THREAD, 2.2-GHZ XEON SILVER E4114 CPU×2; DDR4, 2666 MHZ DRAM; AND 4-TB SATA III HARD DRIVE

Applications	Total Memory Accesses (**unenc. / enc.**)	Theoretical Bandwidth Demand (**enc.**)	Runtime (**unenc.**, no memory bound)	Runtime (**enc.**, no memory bound)	Computation Intensity Increase (×)
Enc. LR (Train)	48.63 MB / 538.5 GB	3.37 TB/s	0.145 s	390.6 s	2,693.8
Enc. LR (Test)	26.19 MB / 28.6 GB	1.99 TB/s	13.48 ms	2.44 s	181.4
MNIST Evaluation	19.25 GB / 62,408.5 GB	273.28 GB/s	212.16 s	112,026.16 s	528.0
MatrixMult	0.565 GB / 1,683.52 GB	1.49 TB/s	1.1 s	1,838.5 s	1,111.76

TABLE III
ANALYSIS OF THE BENCHMARK RUNTIME ON DIFFERENT HARDWARE PLATFORMS. WE USE THE DATA DIRECTLY REPORTED IN THE LITERATURE TO ESTIMATE THE PERFORMANCE OF HEAX [39]. OUR METHODOLOGY—POSEIDON-NDP, EMPLOYS THE PRACTICAL NDP-BASED PLATFORM AND THE FULL-SYSTEM PERFORMANCE IS REPORTED. CONCRETE EXPERIMENTAL SETUP IS SPECIFIED IN SECTION V-A

Platform / Application	CPU (Xeon)	GPU (P100) [6]	Stratix10 [39] (300MHz; **Theoretical**)	Stratix10 [39] (300MHz; **Practical**)	SmartSSD (300MHz; NDP, non-Fusion)	SmartSSD **[Poseidon-NDP]** (300MHz; NDP, NTT-Fusion)
Enc. LR (Train)	420 s	201.97 s	8.16 s	12.31 s	10.06 s	5.79 s
Enc. LR (Test)	2.55 s	1.63 s	26.84 ms	113.43 ms	44.61 ms	30.2 ms
MNIST Evaluation	116,815.6 s	58,705.31 s	1,174.48 s	2,446.89 s	1,524.16 s	1,012.17 s
MatrixMult	1,864.6 s	1,025.55 s	16.18 s	18.73 s	18.6 s	11.75 s

operation, and the multiplication is the convolution of the respective coefficient vectors of the two polynomials.

③ *NTT/INTT:* NTT uses the "primitive roots" from a finite ring Z_q to operate on integer-coefficient polynomials. Given two polynomials a and b, the basic principle of NTT-based polynomial multiplication is to transform the "coefficient representation" of the polynomial into the "point-value representation": $a_{\text{ntt}} = \text{NTT}(a)$, $b_{\text{ntt}} = \text{NTT}(b)$, also termed as the NTT domain. The polynomial multiplication and addition are equivalent to the elementwise multiplying or adding of the vectors in the NTT domain: $\text{result}_{\text{ntt}} = a_{\text{ntt}} \odot b_{\text{ntt}}$ or $a_{\text{ntt}} + b_{\text{ntt}}$. Such operation significantly reduces the complexity of the polynomial arithmetic. The result is then converted back to the coefficient representation by INTT: $\text{result}_{\text{coeff}} = \text{INTT}(\text{result}_{\text{ntt}})$. The FHE algorithm ensures decrypting $\text{result}_{\text{coeff}}$ is identical to the plaintext result.

B. Acceleration Challenges

1) Intractable Data Movement: Table I lists the FHE benchmarks used for the analysis of the algorithm features and the evaluations of Poseidon-NDP. The four applications range from the simple matrix multiplication to the more complicated deep learning training/testing, all of which take the encrypted data as input and perform homomorphic arithmetic stipulated by several classic FHE algorithms—CKKS and BFV.

First, despite the large size of the coefficient modulus and the secret key, the encryption increases the size of the "input data" remarkably, because the plaintext must be hidden in a very large-scale ciphertext polynomial to ensure the security and homomorphism. Compared with unencrypted input data, the expansion ratio reaches up to 50 thousand times. The explosion of the input ciphertext inevitably leads to the explosion of the intermediate data during the application execution. Table II clearly shows the significant variation of the architectural metrics before and after the encryption. For example, the "theoretical" bandwidth demand for the ciphertext computation attains up to 3 TB/s; the total memory accesses increase from tens of megabytes to tens of thousands of gigabytes. We can conclude that the FHE benchmark is highly "memory intensive." Unfortunately, previous works either regard the computation acceleration as the first priority [12], [38] or directly ignore the critical impact of the intractable data movement [33], [34], [39]. If taking the memory access into consideration however, the acceleration performance will degrade severely. As evidence, Table III estimates the gap between the "theoretical" and "practical" runtime of HEAX [39]. The data in the literature only reports the computational speed—the theoretical case excluding the data movement between the computational cores and the memory system. If we use the peak bandwidth provided by the DDR4 of its Stratix FPGA to estimate the practical-case performance, it exhibits 4.2× runtime degradation.

2) Computation Bottleneck: It is widely adopted that the application after FHE is also computationally intensive. Also proved by Table II, the increment of the computation intensity

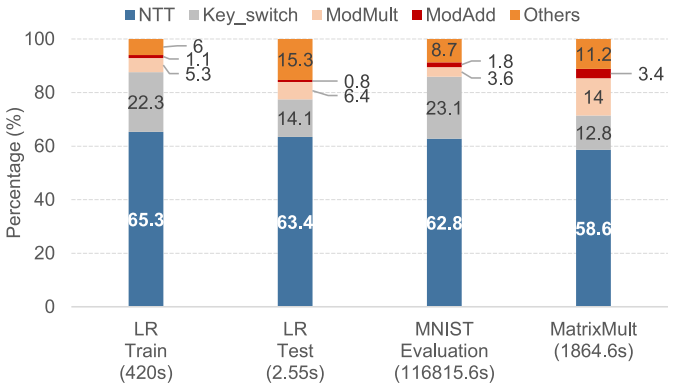

Fig. 2. Time breakdown of the key operators in FHE. NTT occupies the primary portion of the benchmark runtime.

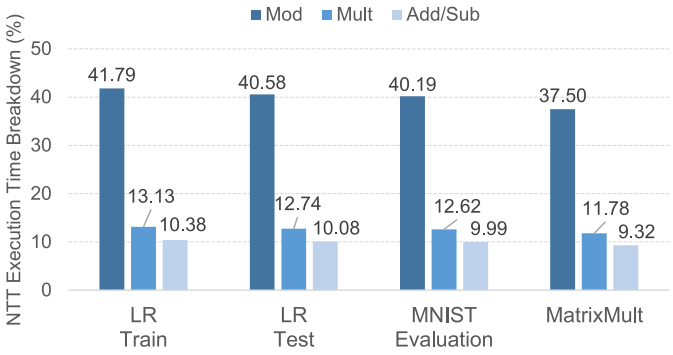

Fig. 3. Intrinsic time breakdown of NTT. Modular arithmetic is the most time-consuming operation globally. Note that each percentage datum is normalized to the total runtime of the benchmark.

attains thousands of times under the assumption of the in-situ memory access (the rightmost column). Two factors induce such a huge magnitude. The first one is the explosion of the ciphertext as mentioned in Table I. The degree of the ciphertext polynomial is generally around 2^{12}–2^{16}. CPU is burdened by the lengthy polynomial arithmetic to accomplish FHE. The second factor stems from plenty of extra special operations in FHE, i.e., the encryption or decryption, which are not involved in the plaintext computation.

Fig. 2 illustrates the time breakdown of the FHE key operators. Although the benchmarks employ different types of FHE schemes, the basic operations are the same: NTT/INTT, ModMult, ModAdd, keyswitch, and other miscellaneous operations. "Keyswitch" includes two main operations: 1) "rescaling," which divides the plaintext by a constant value and 2) "rotation," which circularly rotates the ciphertext. The figure proves that the overwhelming computation bottleneck is NTT/INTT. All the benchmarks exhibit more than 58% fraction of the total runtime. Such NTT domination stems from tackling high-degree polynomials, and the frequent transforming of these polynomials from the coefficient representation to the vectorized representation which is very time consuming.

To make things worse, single NTT/INTT is time-consuming as well. It incurs numerous expensive modular arithmetic, which monopolizes the runtime of each execution as proved by Fig. 3. The fraction of "Mod" exceeds 37% of the total runtime, followed by the integer multiplications (∼12%) and additions (∼9%). We can conclude that the modular arithmetic is the key reason causing the FHE computation bottleneck. However, it also exposes a unique opportunity to accelerate FHE, if we could successfully reduce the expensive modular operations through specific algorithm optimization techniques. That is what Poseidon-NDP targets at the software level.

C. Potential Opportunities

1) Near Data Processing: From the above analysis, the architectural characteristics of FHE are twofold: 1) computation and 2) memory intensive. In Poseidon-NDP, we first seek to leverage novel memory-access-friendly architecture to alleviate the formidable data movement overhead. The affinity to the vast imposition of the ciphertext is possibly beneficial to the performance improvement of the FHE accelerator. In Table III, we evaluate the benchmarks on four types of hardware platforms, including CPU, GPU, non-NDP FPGA, and NDP-based FPGA. Comparing with conventional DRAM memory which suffers the limited bandwidth and power budget, NDP is a proper architectural choice for its targeting the data-centric computation scenario. However, when considering the FPGA board with HBM, the limited HBM memory capacity will become the main bottleneck (i.e., the state-of-the-art Xilinx U280 FPGA board only supports 8 GB for HBM memory). Fortunately, the SmartSSD [50], a Xilinx commercial product, provides the FPGA device with DRAM and neighboring the vast-capacity SSD. Such architecture enables the accelerator to access the large capacity storage with high bandwidth. Therefore, in this article, we employ the SmartSSD to implement our Poseidon-NDP accelerator. It is worth mentioning that our proposed Poseidon-NDP structure does not only focus on the FPGA, it can provide much higher performance with ASIC-based NDP devices.

The runtime result clearly shows an overwhelming advantage of the FPGA-based FHE accelerator. Stratix10 FPGA is used in the prior state-of-the-art work—HEAX [39]. However, this work only focuses on the computation acceleration and impractically assumes the in-situ memory access. In other words, the costly data movement overhead is simply ignored, so the runtime reported can only be regarded as "theoretical." In order to obtain the real-world FHE performance, we deploy the benchmarks on the SmartSSD to run the "practical" full-system experiment and obtain more than 41× improvement (the second column from the right-hand side) relative to the CPU. It proves the NDP architecture actually takes effect to alleviate the intractable data movement overhead.

2) Operator Fusion: Although NDP has great advantages in the FHE acceleration, only leveraging NDP is also suboptimal. The large-capacity SSD equipped in the NDP-based SoC always consumes significant area and power. As the victim, the associate FPGA device has to be compromised to obey the tight thermal design power budget of the SoC. A direct consequence is that the FPGA device cannot be equipped as powerful as the non-NDP platform. As evidence, the FPGA in SmartSSD only belongs to the Xilinx *K* series to balance the power and performance.

This tradeoff makes us reconsider the solution of the computation bottleneck in FHE, that is, leveraging the less powerful FPGA device to also achieve high-performance computing. As proved before in Figs. 2 and 3, the modular arithmetic is the most expensive operation in NTT/INTT and even the whole FHE. If we could reduce its amount in NTT/INTT, the FPGA computation burden might be effectively alleviated as well.

In NTT/INTT, one of the important operands - the "twiddle factor" has the exponential characteristics and satisfies the exponential operation rules, expressed as $w_i \cdot w_j = w_{i+j}$, where w_i, w_j, and w_{i+j} are the twiddle factors. It means the continuous multiplication of the twiddle factors can be finally transformed into only one twiddle factor w_{i+j} as the result. This feature provides an unique optimization opportunity that we could take advantage of to simplify the NTT computation. By combining multiple iterative basic operations into one single operation, the number of the expensive modular operations could be reduced. For example, let $x_1 = (a_1 + a_2 \cdot w_1) \bmod q$ and $x_2 = x_1 \cdot w_2 \bmod q$, then $x_2 = (a_1 \cdot w_2 + a_2 \cdot w_3) \bmod q$. This procedure is called "NTT-fusion" in Poseidon-NDP—the algorithm optimization method we propose to accelerate FHE from the software level. The last column in Table III proves its efficacy on top of NDP—up to 28× runtime reduction, and the results are practical. We will detail this NTT-fusion method in Section III and the collaborative Poseidon-NDP accelerator design in Section IV.

III. Methodology

A. Theorem

The basic computation pattern in NTT is a series of recursive "twiddle, accumulation and modulo (TAM hereafter)" operations. For example, a_1 and a_2 are two coefficients of a high-degree polynomial, and the phase 1 twiddle factor is w_1. TAM means$(a_1 + a_2 \cdot w_1) \bmod q$, where q is the modulus. If we group 8 such input coefficients $(a_1 \sim a_8)$ for the NTT or INTT, it will experience the 3-phase TAM with 24 modulo operations (please see Fig. 6). Although these tedious modulo operations are necessary and very expensive, the good news is that TAM is "recursive." Each recursion will absorb the previous TAM result as the input of this time, so it creates a unique opportunity to fuse multiple TAMs to reduce the twiddling and modulo operations simultaneously. We have the following lemma.

Lemma: Let $X_1 = a_1 + a_2 \cdot w_1$, $X_2 = a_3 + a_4 \cdot w_2$, $X_3 = [X_1 \bmod q + (X_2 \bmod q) \cdot w_3] \bmod q$ where a, w, and q are integers. Then, $X_3 = (X_1 + X_2 \cdot w_3) \bmod q$.

Proof: Given a positive integer q and any integer n, there is an equation: $n = k \cdot q + s$, where k, s are integers and $0 \le s < q$. We have $X_1 = k_1 \cdot q + s_1$ and $X_2 = k_2 \cdot q + s_2$. Thus, substitute X_1 and X_2 into X_3, we have $X_3 = (k_1 \cdot q + s_1) \bmod q + [(k_2 \cdot q + s_2) \bmod q] \cdot \bmod q = (s_1 + s_2 \cdot w_3) \bmod q$. Let $X_3' = (X_1 + X_2 \cdot w_3) \bmod q = [(k_1 \cdot q + s_1 + (k_2 \cdot q + s_2) \cdot w_3] \bmod q = [(k_1 + k_2 \cdot w_3) \cdot q + s_1 + s_2 \cdot w_3] \bmod q = (s_1 + s_2 \cdot w_3) \bmod q$. Thus, we have $X_3 = X_3'$. QED.

Let a_1, a_2, a_3, and a_4 be the 4 NTT inputs, and w_1, w_2 be the twiddle factors. Let A_1, A_2, A_3, and A_4 be the 4 NTT outputs. Then, the 4-input NTT has 2-phase TAMs. Thus, we have the following theorem. ■

Theorem: $A_1 = [a_1 + a_2 \cdot w_1 + (a_3 + a_4 \cdot w_1) \cdot w_1] \bmod q$, $A_2 = [a_1 - a_2 \cdot w_1 + (a_3 - a_4 \cdot w_1) \cdot w_2] \bmod q$, $A_3 = [a_1 + a_2 \cdot w_1 - (a_3 + a_4 \cdot w_1) \cdot w_1] \bmod q$, and $A_4 = [a_1 - a_2 \cdot w_1 - (a_3 - a_4 \cdot w_1) \cdot w_2] \bmod q$.

Proof: Let the output of the first-phase TAM be t_1, t_2, t_3 and t_4, where $t_1 = (a_1 + a_2 \cdot w_1) \bmod q$, $t_2 = (a_1 - a_2 \cdot w_1) \bmod q$, $t_3 = (a_3 + a_4 \cdot w_1) \bmod q$, and $t_4 = (a_3 - a_4 \cdot w_1) \bmod q$. Thus, the 2nd-phase TAM: $A_1 = (t_1 + t_3 \cdot w_1) \bmod q$, $A_2 = (t_2 + t_4 \cdot w_2) \bmod q$, $A_3 = (t_1 - t_3 \cdot w_1) \bmod q$, and $A_4 = (t_2 - t_4 \cdot w_2) \bmod q$. For A_1, substitute t_1 and t_3 into A_1, we have $A_1 = (a_1 + a_2 \cdot w_1) \bmod q + [(a_3 + a_4 \cdot w_1) \bmod q] \cdot w_1 \bmod q$. Referring to the previous lemma, we have $A_1 = [a_1 + a_2 \cdot w_1 + (a_3 + a_4 \cdot w_1) \cdot w_1] \bmod q$. A_2, A_3 and A_4 can be proved in analogy to A_1. QED.

The theorem illustrates that a 2-phase recursive TAMs could be fused into a 1-phase TAM without changing the NTT result. Therefore, we can fuse a 3-phase or even n-phase TAMs together to significantly reduce the modulo operations. For example, A_1 only requires 1 modulo operation after fusion instead of 3 before fusion. ■

Algorithm 1: NTT Fusion Procedure

Input : Polynomial $a \in \mathbb{R}_q$ of degree n-1, nth primitive roots $\omega_n \in \mathbb{Z}_q$ of unity, the degree of fusion FD, $2 \le FD \le \log_2 n$

Output: Polynomial $a_{ntt} = NTT(a) \in \mathbb{R}_q$

```
begin
    for m ← 1 to log2^n / FD do
        Set(ω); ω_m ← ω_n^{n/m};
        for l ← 0 to n/2^FD do
            u^{2^FD} ← a[l · 2^FD, (l + 1) · 2^FD − 1];
            v^{2^FD} ← NTT_{Core_{2^FD}}(u^{2^FD}, ω);
            a[l · 2^FD, (l + 1) · 2^FD − 1] ← v^{2^FD};
            ω ← update(ω);
        end
        reorder(a);
    end
    return a_ntt;
end
```

B. NTT-Fusion—Optimizing NTT Computation for FHE

Based on the above theorem, we formally propose NTT-fusion. It aims to reduce the expensive modulo operations by fusing the recursive TAMs. Depending on the polynomial degree, the TAM phases usually attain 12 (2^{12} coefficients) ~16 (2^{16} coefficients) or more. Fusing more TAMs will result in much less modular operations, but it will also introduce more twiddles and accumulations. The appropriate number of fused TAMs is instantiated as a design parameter in NTT-fusion—the fusion degree (FD) for short.

The concrete NTT-fusion procedure is shown in Algorithm 1. For a polynomial $a \in \mathbb{R}_q$ with degree n,

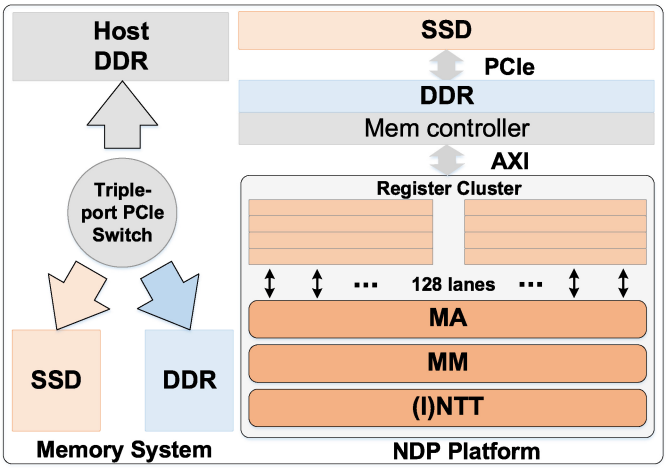

Fig. 4. Poseidon-NDP overall architecture. The NDP platform communicates with the host DDR through a triple-port PCIe switch. Three key operators are implemented as the vectorized computation cores with 128 lanes.

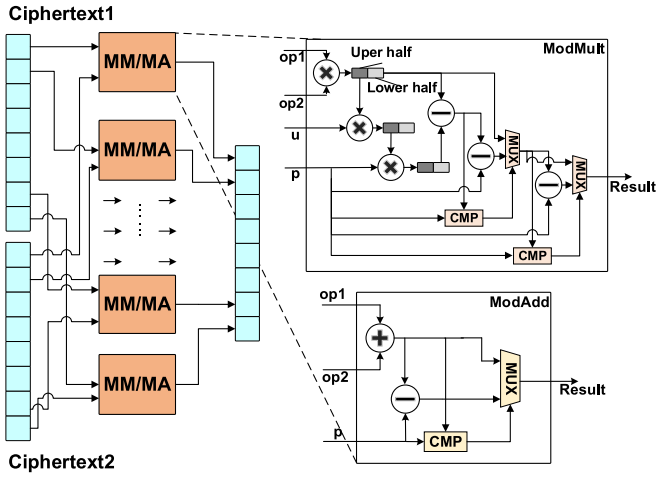

Fig. 5. MA/MM core architecture in Poseidon-NDP. We implement fine-grained decomposition to reduce the resource consumption of ModMult, following the Barrett Reduction algorithm and due to the nature of addition results we implement a subtractor to perform the ModAdd.

conventional NTT requires the $\log_2^n$ phases. After NTT-fusion, the number of phases reduces to $\log_2^n/FD$ (line 2), and each phase involves $(n/2^{FD})$ fused-TAM computations. Each fused TAM is then computed by the "for loop" in line 4∼9 of Algorithm 1. Finally, due to the NTT-specific characteristics, the result of each iteration needs to be reordered for the next iteration (line 10). Compared with the conventional NTT procedure, NTT-fusion reduces the number of iteration loops by fusing TAMs. From the hardware acceleration perspective, each fused-TAM computation is also parallelizable. We could instantiate $(n/2^{FD})$ NTT cores in the accelerator to compute all the fused TAMs in parallel. Note that, the NTT-fusion is based on directly formula derivation, which achieves the same accuracy when comparing to the other approaches (i.e., CPU, GPU, and HEAX).

Comparing to the similar formula derivation in the recent previous works [27], [28], [31], [51], NTT-fusion allows to merge more NTT-operations before the modular, which further reduces the number of modular operations. In the next section, we will detail how our proposed accelerator, namely, Poseidon-NDP, is designed to enforce the NTT-fusion.

IV. FHE Accelerator—Poseidon-NDP

A. Overall Architecture

The overall architecture of Poseidon-NDP is shown in Fig. 4. In addition to the DDR, Poseidon-NDP has its own large-capacity SSD memory, which can avoid frequent data handling between the accelerator and host. The computation cores, including MA(ModAdd), MM(ModMult), and NTT/INTT, are vectorized and communicate with the build-in DDR through the corresponding register cluster.

In Poseidon-NDP, the register buffer, which is directly connected to the computation cores, obtains the input data from the DDR by the AXI interface, and each core (i.e., MA, MM and NTT) parallelly processes the data in a pipelined manner. In our design, each computation core has its own input/output buffer and there is no data conflict in the pipeline. In Poseidon-NDP, the data-level parallelism is set to 128. In theory, thus, the throughput is 128 per clock. When FD is set to 3, for example, each NTT core processes 8 operands per cycle, and all 16 cores enable the 128° of parallelism in NTT calculation. Differing from the NTT core, MA and MM core process two operands in each cycle, so the number of MA/MM cores is more than that of the NTT core. Besides the register cluster, we also use multiple BRAMs to read or write the required data in parallel for each computational core in our FPGA-based evaluations.

Memory System: To fully utilize the NDP characteristics, Poseidon-NDP also leverages the abundant off-FPGA bandwidth provided by NDP to accelerate the data movement. As shown in Fig. 4, the NDP architecture involves a triple-port PCIe switch connecting the upstream PCIe link to DDR4 and the downstream PCIe link to the NVMe SSD controller. It enables transparent NVMe SSD access with minimal add-on latency from the FPGA. In Section V-A, we employ SmartSSD [50] to provide the NDP support for Poseidon-NDP and implement the FHE accelerator in its built-in FPGA device.

B. Computational Cores

1) MA/MM: The ModAdd operation in FHE aims to add two integer vectors under the modulus q. Taking the modulo operation following the addition is the common computation process of ModAdd. However, because the operand is already less than the modulus q, there are only two possible results. As described in (1), if $a+b \geq q$, the result of ModAdd is $a+b-q$; otherwise, $a+b$ is the final result. The hardware circuit logic is shown in Fig. 5, it consists only of the adder, comparator and selector

$$(a+b) \bmod q = \begin{cases} a+b, & a+b < q \\ a+b-q, & a+b \geq q. \end{cases} \quad (1)$$

Similar to ModAdd, ModMult is also a vectorized operation. It is more complex than ModAdd because of the multiplication before the modulo operation. Multiplication will expand the

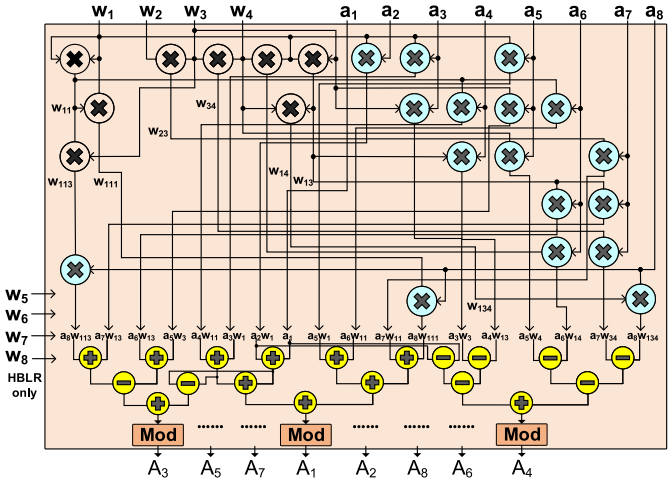

Fig. 6. NTT core architecture in Poseidon-NDP. It has two instances: LBHR and HBLR. LBHR entails all the multipliers in the figure; and HBLR only needs the multipliers marked in blue.

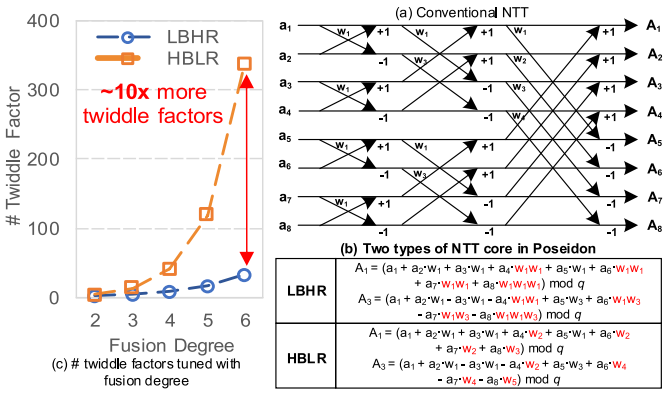

Fig. 7. Iterative fusion calculation rules. The fusion-based NTT algorithm can reduce the size of the twiddle factor after fusion by the exponential characteristic of the twiddle factor that can simplify the calculation.

intermediate result to the magnitude of q^2 and thus cannot be processed in a similar way to ModAdd. The traditional modulo operation requires division and is not hardware-friendly, so we employ Barrett Reduction [24] to finish the computation. The method replaces the original division operation with multiplication by an approximate calculation and eliminates the error by a finite number of comparisons to obtain the exact modulo result at the end. The detailed hardware implementation of ModMult is shown in Fig. 5. The smaller number of multiplier requirements and the simple circuit logic allow the MM core to optimize hardware resource utilization and the efficiency of pipelined execution.

2) NTT/INTT: As is proved in Section II, NTT accounts for the largest proportion of the overall FHE applications execution time. Thus, the computation efficiency of NTT is significant for FHE acceleration. The complex computation and access patterns of the NTT pose challenges for the high-performance NTT acceleration unit design. Fortunately, benefiting from the NTT-fusion, Poseidon-NDP eliminates the large number of vanilla TAMs as in the conventional NTT procedure. On the contrary, it focuses on the efficiency of the "fused-TAM" computation, which distinguishes Poseidon-NDP from the prior FHE accelerators.

Fig. 6 shows the 8-fused NTT core architecture at $FD = 3$, there are eight polynomial coefficients and the associate twiddle factors as the input and eight results as the output. Different from conventional NTT, the Poseidon-NDP NTT core only requires one phase with eight fused TAMs instead of $\log_2^8 = 3$ phases with 24 unfused TAMs (eight for each phase). From the figure, we can see that only one modulo operation is required because of the only one phase after NTT fusion. Therefore, chunking the complete NTT computation and mapping it to the 8-fused NTT core can significantly reduce the number of iterations and modulo operations. We hard-code the circuit in RTL to optimize the 8-fused-TAM computations.

Tradeoffs: According to the NTT-fusion theorem in Section III-A, each output A_i entails the unfixed multiplication of the twiddle factors. For example, A_2 and A_4 incorporates $w_1 \cdot w_2$, while A_1 and A_3 incorporates $w_1 \cdot w_1$. Twiddle factor is essentially the exponential expression of the primitive root, which has the following characteristic: $w_i \cdot w_j = w_{i+j}$. Therefore, the multiplication of two twiddle factors equals twiddling the primitive root on the unit circle (i.e., $w_1 \cdot w_2 = w_3$ and $w_1 \cdot w_1 = w_2$). FD determines the terms of the twiddle-factor multiplication. Usually, a larger FD setting denotes more twiddle factors that will be multiplied together after NTT fusion. Therefore, this operation will inevitably introduce extra computations that could undermine the accelerator performance, so a proper FD setting is critical for balancing the impacts of fusing TAMs and multiplying twiddle factors. Section V-F will evaluate the design space of FD and illustrate that FD = 3 performs the best.

The second tradeoff regards the architectural design of the NTT core in Poseidon-NDP. Since the twiddle factors must be multiplied after fusion when implementing the multiplication also has two conditions: 1) premultiply several twiddle factors before the NTT-core input and 2) multiplying on the spot inside the NTT core. The first condition is conducive to reducing the computation burden on the NTT core but has to increase the storage and data movement for the generated new twiddle factors (i.e., w_{1+1}, w_{1+3}). However, the second condition performs the opposite: less data movement but more computations in the NTT core. This tradeoff triggers two instances of Poseidon-NDP [as shown in Fig. 7(b)].

1) *Low Bandwidth and High Resources (LBHR):* LBHR only takes the original twiddle factors as input (i.e., $w_1 \sim w_4$ in Fig. 6). Multiplying these twiddle factors is performed inside the NTT core. Therefore, more hardware multipliers must be instantiated. For example, in Fig. 6, all of the multipliers are involved. Hence, it must consume more FPGA resources to implement the NTT core. However, the strength of LBHR is the low-bandwidth-and-storage consumption, because it does not generate new twiddle factors.
2) *High Bandwidth and Low Resources (HBLR):* HBLR is the counterpart of LBHR. It takes the newly generated twiddle factors as input (i.e., $w_5 \sim w_8$ in Fig. 6). They are precomputed before the NTT input, so it costs fewer

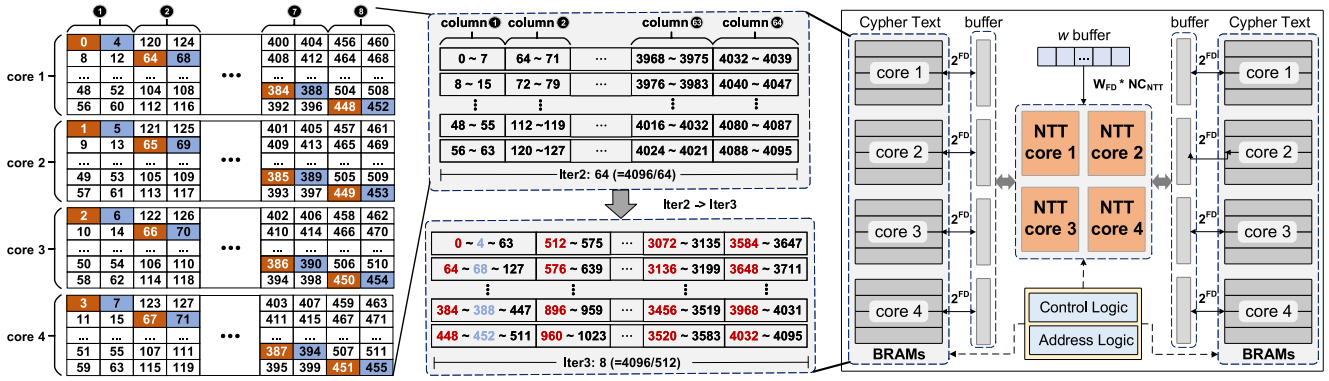

Fig. 8. Poseidon-NDP on-FPGA data access pattern.

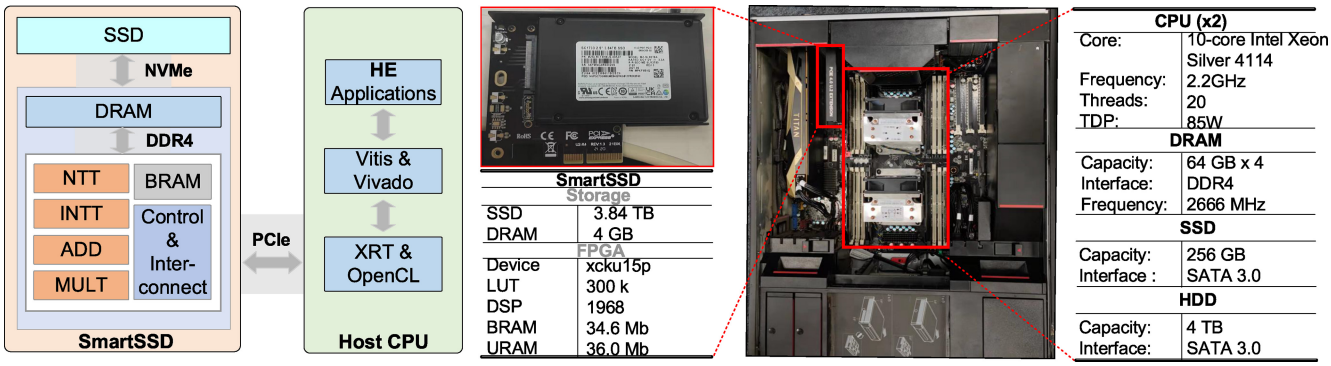

Fig. 9. System overview. Poseidon-NDP is implemented in the built-in FPGA of SmartSSD.

multipliers than LBLR; only the blue ones in the figure are involved. Fewer hardware resources are required to enable more NTT cores in the Poseidon-NDP PE, but it also causes more data movements between PE and DRAM. Fig. 7(c) quantitatively illustrates the difference scaled with FD. In Section V, we will fully evaluate the pros and cons of LBHR and HBLR.

C. Data Access Pattern and Architecture

The method of fusion simplifies the computational complexity of NTT while changing the way of memory access. There are $\log_2^n$ iterations in traditional NTT, where n is the polynomial degree, and the offset between two inputs of a TAM is $2^{\text{iter}-1}$, where iter represents the current number of iterations. In Poseidon-NDP, the iteration number of NTT is reduced effectively. Taking $n = 4096$ and $FD = 3$ as an example, the number of required iterations is reduced from 12 ($\log_2^{4096}$) to 4 ($\log_2^{4096}/FD$). In addition, as multiple iterations are merged into one, the index offset between the input data of the "fused-TAMs" becomes accordingly $2^{(\text{iter}-1)*3}$.

In Poseidon-NDP, the NTT cores are fully parallelized and pipelined. Thus, the NTT cores require sufficient on-chip data bandwidth to achieve the best performance. We allocate the BRAM blocks for each NTT core that matches the amount of input/output data and supports the paralleled read/write. Fig. 8 illustrates the data access pattern of Poseidon-NDP taking 4-cores NTT under $FD = 3$ as the example. For each NTT core, there is an input and output BRAM buffer to cache the intermediate result of each iteration and support ping-pang operation to cover the delay of reading after writing. The left and middle parts of Fig. 8 present an instance of the data path, which takes the 2nd and 3rd iterations as an example. The data offset of iter2 and iter3 is 8 ($2^{(2-1)*3}$) and 64 ($2^{(3-1)*3}$). To enable the intermediate result can directly as the input of the third iteration for the fully pipelined execution, we divide the result of iter2 into multiple columns in groups of 8 (64/8) and map them into the corresponding BRAM blocks of 4 NTT cores. Excepting for the first column, all the other column of data is cyclically shifted by one data position to the adjacent BRAM block based on the previous column, thus distributing the data spaced at 64 intervals into different BRAM blocks (i.e., index 0, 64, 128, 192, 256, 320, 384, and 448 marked in red). With such a storage method, NTT cores can read the required data simultaneously per cycle in the third iteration. The access pattern enables NTT cores directly load the operands without any delay at the beginning of each phase. Poseidon-NDP achieves a promising overall performance by collaborating with the support of NDP-based off-FPGA memory channels.

V. Evaluation

A. Experimental Setup

Platform: Poseidon-NDP is a practical FHE accelerator, so we build a real-world experimental environment based on the x86 CPU system. As shown in Fig. 9, the Poseidon-NDP accelerator is implemented in the SmartSSD plugged into the

TABLE IV
PERFORMANCE COMPARISON BASED ON THE METRIC OF KEY OPERATIONS NUMBER PER SECOND. TAKING CPU AS THE BASELINE

Operation \ Platform	CPU (Xeon)	GPU (P100) [6]	Roy [41]	HEAX [39]	Medha [35]	FAB [4]	Poseidon-NDP (ours)	
							LBHR	**HBLR**
#NTT per second	10,758	27,777	13,699	195,313	130,209	3,125,000	1,171,874 (108.9×)	2,343,749 (217.9×)
#INTT per second	13,268	25,000	11,765	195,313	130,209	3,125,000	1,171,874 (88.3×)	2,343,749 (176.6×)
#ModMult per second	68,978	/	76,336	1,171,875	1,562,500	18,750,000	9,375,000 (135.9×)	9,375,000 (135.9×)
#ModAdd per second	77,051	/	73,529	/	1,562,500	18,750,000	9,375,000 (121.7×)	9,375,000 (127.7×)

PCIe slot of the mainboard. We installed Xilinx off-the-shelf developing toolkit Vivado [2] and Vitis [1] version 2021.1 on the host side. These tools will be working in conjunction with the Xilinx Runtime [3] environment and OpenCL framework to interact with the SmartSSD through PCIe. As an NDP platform, SmartSSD owns a self-contained memory system. The intermediate data of the FHE application are transferred among the SSD, the DRAM and the FPGA device. Only the results are sent back to the host. It eliminates the inefficiency caused by the fuzzy communication between the accelerator and the host storage. Detailed environmental configuration is shown in Fig. 9.

Baseline: In our experiments, the CPU running the Microsoft SEAL library [45] is selected as the baseline. Besides, we also compare Poseidon-NDP with the GPU [6] and four state-of-the-art FHE accelerator designs [4], [35], [39], [41]. Note that, to make a fair comparison, we uniformly use 2^{12} as the polynomial size in the performance comparison of key operations in our experiments.

Benchmark: We use four applications as benchmarks (as shown in Table I) for the evaluation. The `Enc. LR(Train)` and `Enc. LR (Test)` [18] refer the logistic regression models for training and testing-based encrypted heart disease data in respective. Each sample in the dataset owns nine dimensions, while we use 780 samples for the training and 334 samples for the testing. The benchmark is performed for five epochs, and the final accuracy obtained is 0.7. The `MNIST Evaluation` [17] refers the entire inference process of a simple convolutional neural network (CNN) with one convolutional layer and two linear layers on the encrypted official MNIST test dataset, which includes 10000 samples. The convolutional layer includes four kernels with the shape of 7×7, and the strides is 3×3; the number of hidden neurons of two fully connected layers are 64 and 10, respectively; the activation function employed in the model is square activation. Notably, there is no padding operation and batch norm (BN) layer before and after the convolution layer, and both the convolution and full-connected layers have the bias parameter, which is involved in the computation in the plaintext form. The `MatrixMult` [25] refers the multiplication of two encrypted matrices, each matrix is 100 × 100. These benchmarks use batching techniques to boost computation efficiency. In the `LR`, the 9 data of each item are batched into a single ciphertext; in the `MNIST Evaluation`, the feature map matrix of 28×28 is fully expanded into a 1-D vector and encoded for the efficient convolution; similarly, the rows and columns of the matrix in the `MatrixMult` are also batch encrypted. Using batching technique allows the computation of batches of data through a single ciphertext operation, e.g., convolution in CNN and vector inner-product in matrix multiplication, thus significantly improving the efficiency of FHE applications.

B. Acceleration Performance

1) Key Operator Computation: The computation of the key operators in FHE, i.e., NTT /INTT, ModMult, and so on, directly determines the performance of the accelerator. The accelerator architectural design in return determines the performance of the operator computation. In Poseidon-NDP, we instantiate two instances—LBHR and HBLR, and as mentioned in the previous section, the two instances tackle the tradeoff specialized in the NTT-fusion. In this experiment, we compare the pure operator computation performance of Poseidon-NDP with the CPU and GPU baseline. Here, we use the "executed key operation number per second" as the performance metric. The results are listed in Table IV. Poseidon-NDP outperforms CPU by 88× at least for LBHR and up to 217× for HBLR, respectively. The GPU performance lies between the CPU and Poseidon-NDP, nearly ∼50× to ∼100× inferior to LBHR and HBLR.

Besides, we also compare Poseidon-NDP architecture to the state-of-the-art FPGA-based FHE accelerators, i.e., HEAX [39], Medha [35], and FAB [4]. As shown in Table IV, although HEAX achieves up to 17× improvement for the ModMult operation on the CPU, our proposed LBHR and HBLR still perform 8× better than it. Medha has a lower parallelism of 32 and the accelerator runs at 200 MHz, so its speedup of NTT, ModAdd, and ModMult is slightly lower than that of HEAX. Compared to Medha and HEAX, FAB has no algorithmic optimization in the design of the NTT unit but has the advantage of a higher parallelism of 512, so it achieves a much higher speedup than HEAX and Medha. Poseidon-NDP is designed with a highly optimized NTT unit, making its performance on NTT close to that of FAB at a parallelism of only 128. We cannot achieve the same parallelism as FAB because the FPGA resource in our experimental platform is much less than that in FAB. However, at the same parallelism, the performance of Poseidon-NDP will be 3× more than that of FAB.

Discussion: The evaluation has proved the efficacy of the NTT-fusion and the architectural design of Poseidon-NDP. It could execute much more key operators like NTT/INTT per second because the number of modular arithmetic, having been proved as the most critical computation bottleneck, is reduced significantly by the fusion. Such optimization seeks to offload the FHE computation burden from the software level and is proved to be effective. The fine performance of the

TABLE V
OVERALL FHE PERFORMANCE COMPARISON WITH CPU. IN HERE, WE CARRY OUT THE FULL SYSTEM EVALUATION. THE IMPROVEMENT IS CONVEYED BY NOT ONLY THE COMPUTATION OPTIMIZATION BUT ALSO THE NDP-BASED MEMORY ACCESS. NOTE THAT IN THE BENCHMARK OF MNIST EVALUATION, WE USE THE OFFICIAL MNIST TEST DATASET OF 10000 SAMPLES FOR SYSTEM EVALUATION

Platform / Application	CPU (Xeon)	GPU (P100) [6]	HEAX [39]	Medha [35]	FAB [4]	**Poseidon-NDP (Theoretical)**		**Poseidon-NDP (NDP)**	
						LBHR	**HBLR**	**LBHR**	**HBLR**
Enc. LR (Train)	420 s	201.97 s	8.16 s	24.27 s	1.45 s	6.56 s	3.89 s	8.47 s	5.79 s
Enc. LR (Test)	2.55 s	1.63 s	26.84 m	143.33 ms	7.8 ms	19.66 ms	12.48 ms	37.42 ms	30.2 ms
MNIST Evaluation	116,815.6 s	58,705.31 s	1174.48 s	6466.57 s	392.82 s	992.48 s	662.49 s	1332.17 s	1012.17 s
MatrixMult	1,864.6 s	1,025.55 s	16.18 s	105.95 s	6.01 s	13.61 s	9.33 s	16.03 s	11.75 s

key operators also paves the road to a satisfying full-system performance, which will be specified next.

2) Full-System Performance: Table V exhibits the full-system performance result. Apart from Table IV targeting the "pure" operator computation performance, this experiment evaluates the computation, data movement, and their impact on the full-system performance.

For the CPU, we use Microsoft SEAL library to deploy the selected applications and report the overall runtime data. For HEAX, the data are all "theoretical," because the literature does not elaborate on the memory access methodology. All the data are reported assuming the in-situ data fetching, which is unpractical. However, to compare the theoretical and actual performance gap, we also let Poseidon-NDP work under the same in-situ situation and report the ideal-case result. As shown in the table, the "theoretical" Poseidon-NDP performs way better than HEAX and the CPU or GPU. Just report the least improvement: 107× and 2× faster than CPU and HEAX for the application `Encrypted LR-Train`. The "practical" NDP-enabled Poseidon-NDP behaves a little less optimal than the theoretical; for example, 11.75-s runtime compared with the theoretical 9.33 s for the `MatrixMult`, but still outweighs 72× and 1.4× relative to the CPU and HEAX. Similarly, benefiting from the optimized design and parallelism, Poseidon-NDP achieves a 4× to 9× performance improvement compared to the Medha. The performance of our accelerator is 2× to 4× slower than that of the FAB, mainly because the FAB is based on a high-end FPGA development board, which has much higher-hardware resources than the SmartSSD. However, our optimized micro-architecture design and efficient bandwidth usage will further improve performance on a high-resource platform.

Discussion: A worth mentioning observation is that the theoretical and practical Poseidon-NDP does not exhibit a very large performance gap. In other words, the intractable data movement challenge is well dealt with in Poseidon-NDP. The intrinsic reason stems from the effective NDP architecture. The abundant bandwidth is fully utilized by LBHR and HBLR. Especially for the HBLR, trading fewer FPGA resources with more data movements in NDP is profitable. The inevitably increased memory accesses are well accommodated by NDP, which diminishes the negative influence brought by the vast data movements.

C. Bootstrapping Support

Poseidon-NDP also supports the bootstrapping operation, which consists of basic FHE operators, and our optimization of the operators also benefits it. We evaluate the performance of Poseidon-NDP using the most advanced fully packed bootstrapping algorithm [36] under two different settings.

1) *Set-A:* The polynomial degree is 2^{15} and the high-noise-level ciphertext with the multiplication depth $L = 4$ will be refreshed to the low-noise-level ciphertext with the multiplication depth $L = 15$;
2) *Set-B:* The polynomial degree is 2^{16} and the multiplication depth before and after bootstrapping is 4 and 47.

We explore the impact of NTT-fusion on the performance of bootstrapping under the two settings. As shown in Table VI, NTT-fusion benefits the performance in the both instances, i.e., LBHR and HBLR, and achieves more than 10× performance improvement compared to the CPU, which proves the efficiency of the Poseidon-NDP in the FHE bootstrapping.

TABLE VI
PERFORMANCE OF BOOTSTRAPPING. WE ALSO EVALUATE THE IMPACT OF THE NTT-FUSION IN TWO INSTANCES OF POSEIDON-NDP

Platform / Application	CPU	w/o Fusion	LBHR	HBLR
Bootstrap (Set-A)	8.4 s	1281.5 ms	964.7 ms	777.5 ms
Bootstrap (Set-B)	17.2 s	2869.9 ms	2161.8 ms	1700.5 ms

TABLE VII
IMPACT OF NTT-FUSION ON THE NUMBER OF MODULO OPERATIONS IN EACH BENCHMARK. (10^9)

Application / Platform	Enc. LR (Train)	Enc. LR (Test)	MNIST Evaluation	Matrix Mult
wo NTT Fusion	266	0.828	31900	428
w NTT Fusion	102	0.276	12300	165

TABLE VIII
BENEFIT BREAKDOWN OF THE NTT-FUSION. TAKING `MNIST` AS AN EXAMPLE

#PE	Fusion Degree (FD)			
	2	3	4	5
2	7860.07s	3252.14s	1716.16s	884.17s
4	4276.12s	1972.16s	1204.17s	788.18s
8	2484.15s	1332.17s	948.17s	740.18s
16	1588.16s	1012.17s	820.18s	716.18s

D. Poseidon-NDP Specifics

1) NTT-Fusion: NTT-fusion in Poseidon-NDP plays a vital role in boosting the accelerator performance. This experiment aims to quantify the runtime reduction after NTT-fusion. The runtime reduction is actually achieved by fusing the twiddle factors to further cut down the modulo operations, a.k.a. the most time-consuming bottleneck. As reported in Table VII,

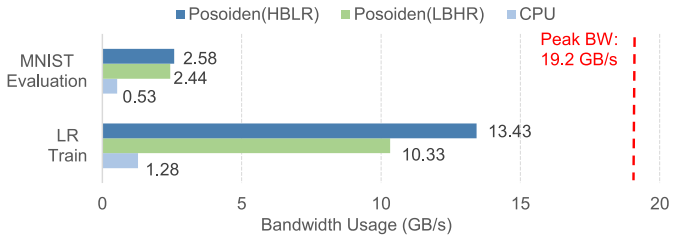

Fig. 10. Bandwidth utilization analysis. The benchmarks demonstrates the uniform behavior of enjoying the broadened bandwidth provided by NDP.

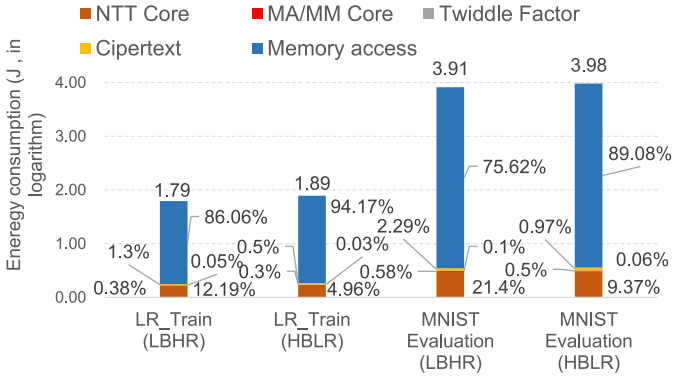

Fig. 11. Energy consumption and breakdown. The DDR4 access takes the major proportion, as expected. HBLR consumes minor extra energy than LBHR.

NTT-fusion helps to reduce the modulo operations, the reduction rate ranges from 61.7% to 66.7%. The optimization at the software level is finally reflected on the overall performance improvement.

We also explore the breakdown of the NTT-Fusion benefits. As shown in Table VIII, we can see that, as the FD increase, the impact of parallelism on performance improvement is reduced. This indicates that the reduction of time-costly operations has more significant impact on the performance improving.

2) Near Data Processing: The affinity to the NDP architecture in Poseidon-NDP also contributes to the more efficient data movement, especially for the huge ciphertext flood in FHE. By quantifying the bandwidth utilization in Fig. 10, we can explore the actual contributions of NDP. The CPU system employed in this experiment coincides with Fig. 9. We profile the bandwidth utilization of the CPU communicating with its associate DDR4 DRAM in the host. The Poseidon-NDP bandwidth utilization is directly obtained by profiling the FPGA communicating with the storage system inside the SmartSSD. The peak bandwidth that DDR4 DRAM can provide is 19.2 GB/s. Benefit from NDP, LBHR enjoys 10.33 GB/s bandwidth (`LR_Train`) and HBLR even enjoys a slightly higher bandwidth—13.43 GB/s, due to the increased twiddle factors after NTT fusion. `MNIST_Evaluation` has a relatively much larger dataset and model size after the encryption, so it needs to frequently interact with the vast storage - the SSD in SmartSSD, for the data movement. The overall bandwidth utilization is 2.44 and 2.58 GB/s, but it still outperforms the CPU (0.53 GB/s) by 4.6× and 4.9×.

E. Energy

1) Energy Consumption and Breakdown: Fig. 11 shows the total energy consumption of Poseidon-NDP. HBLR consumes slightly more energy than LBHR. This is in line with the intuitive expectation, because HBLR handles more twiddle factors than LBHR. Reflected in the hardware activities, it generates more DDR4 accesses and consumes more BRAM resources. Although its NTT core is much faster than LBHR, the shortened runtime does not adequately compensate for the increased power consumption.

The energy breakdown further proves the off-FPGA DDR4 access is the largest energy consumer, and HBLR exhibits ∼15% more memory access energy than LBHR, which explains the reason for the increased "total" energy of HBLR. This experiment demonstrates the power and performance tradeoff in Poseidon-NDP. If a faster FHE computation is the priority, HBLR is possibly the best choice. However, if the thermal design power is a tight constraint, LBHR consumes less energy but also provides a sensible acceleration.

TABLE IX
EFFICIENCY ANALYSIS. WE USE EDP AS THE METRIC (ENERGY × TIME). LOWER IS BETTER

Operation / Platform	LR Train		MNIST Evaluation	
	EDP (10^2)	Enhanced Efficiency	EDP (10^8)	Enhanced Efficiency
CPU Xeon	63900	1×	4570	1×
GPU P100 [6]	7910	8.09×	581	7.87
HEAX [39]	1.7	37122.62×	0.125	36626.25×
LBHR	0.2202	**290330.07**×	0.016	**286448.02**×
HBHR	0.059	**1084566.37**×	0.0043	**1070064.45**×

2) Energy Efficiency: We use energy delay product (EDP hereafter) as the efficiency metric. Table IX lists the efficiency after deploying the FHE applications on the baseline platforms and Poseidon-NDP. Poseidon-NDP exhibits the shortest runtime and the lowest-energy consumption, so it outperforms all the baselines in efficiency.

F. Design Space Exploration

1) Key Design Parameter—Fusion Degree: FD denotes how many TAMs will be fused at a time in LBHR or HBLR. This parameter directly affects the NTT computations in that higher FD indicates fewer modulo operations. However, a higher-FD setting also spawns new twiddle factors that will burden the computation for LBHR or the memory access for HBLR, and thus harms the accelerator performance in return. In order to explore the finest setting so as to harness this tradeoff, we evaluate several hardware metrics and one operator performance metric (execution time perfused NTTs) scaling with FD. Fig. 12 illustrates the results. The 4 metrics uniformly denote an inflection at nearly FD = 3, marked as bold. For the hardware metrics like number of Register (Reg), digital signal processor (DSP), and look-up-table (LUT), the FD = 3 inflection denotes the most optimized resource and area in FPGA, while for the performance metric it denotes a relatively shorter execution time. This setting is also chosen in the previous evaluations in this section.

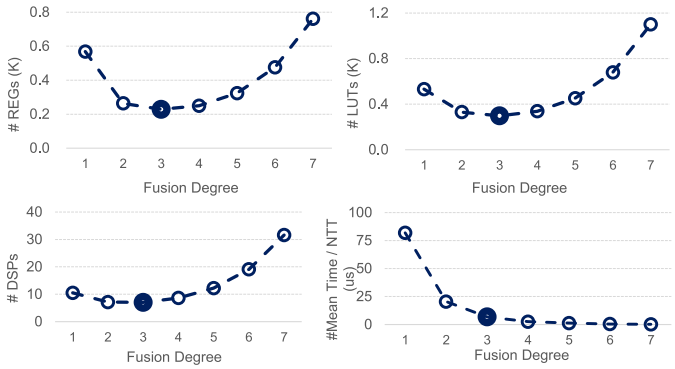

Fig. 12. Key design parameter—FD. We evaluated the FPGA resource usage of a TAM on average (in actual #) as well as the average execution time per NTT (right-bottom figure) scaled by FD. The optimal point emerges at $FD = 3$, where it consumes the lowest resources but performs relatively faster NTT.

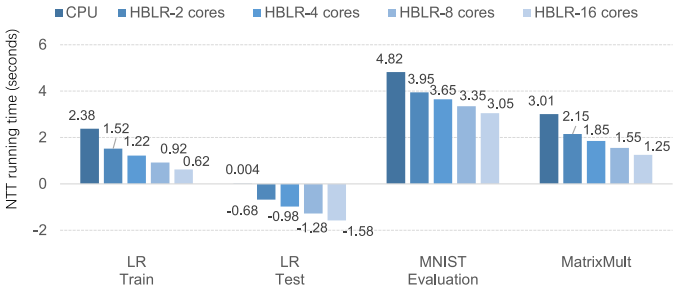

Fig. 13. NTT core scaling analysis. We use the logarithm scale on the y-axis, so the negative value denotes the actual value is below 1.

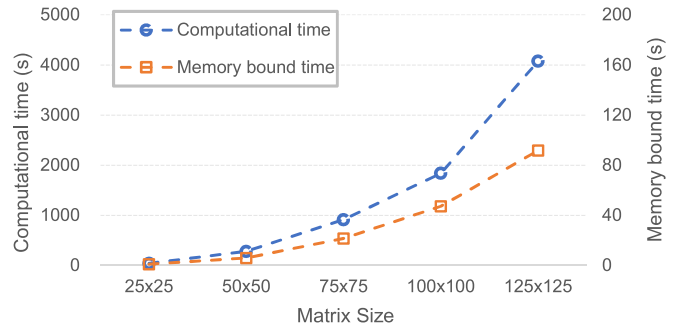

Fig. 14. Sensitivity of the matrix size.

2) NTT Core Scaling: As the faster representative in Poseidon-NDP, HBLR is selected to explore the performance scaling with the NTT cores. Intuitively, more NTT cores will lead to a faster FHE performance, and Fig. 13 proves this notion. The y-axis indicates the runtime perfused NTT in logarithm. For each benchmark, the more the NTT cores, the shorter the runtime. Note that for `LR_Test`, all the runtime is below 1 so the logarithmic result is below 1. In Poseidon-NDP, we choose eight cores for LBHR and 16 cores for HBLR to attain the highest performance according to the equipped FPGA resources in SmartSSD.

3) MatrixMult Size Scaling: We also analyze the computation and memory access overhead of encrypted matrix multiplication at different sizes. Fig. 14 shows the matrix size scaling from 25, 50, 75, and 100 to 125, with respect to the computational time and memory bound time. We can see that the computational time increases in accordance with the matrix size. This is reasonable because intuitively the overhead of the computation should upgrade with the increased matrix size. Meanwhile, the growth becomes progressively larger due to the matrix size increasing bringing the nonlinearly growing number of ciphertext additions and keyswitches. In addition, the memory overhead is consistently linear than the computational time. Therefore, the design of efficient software computation algorithms is also significant for improving the execution efficiency of FHE applications.

TABLE X
COMPARISON OF FPGA RESOURCE CONSUMPTION. COMPARED TO THE OTHER SOLUTIONS, POSEIDON-NDP ACHIEVES BETTER PERFORMANCE EVEN BASED ON THE LESS POWERFUL FPGA

	Roy [41]	HEAX [39]	Medha [35]	FAB [4]	**Poseidon-NDP**	
					LBHR	**HBLR**
Max.bit	30	40	54	54	40	40
Freq.(MHz)	200	300	200	300	300	300
LUT (K)	55	33	22	352	112	96
REG (K)	22	101	52	832	58	64
DSP	182	160	160	2560	1232	1744
BRAM (KB)	1746	1813	3344	6912	1044	1044

G. FPGA Resource Utilization

As mentioned before, an NDP-based platform is more likely to consume a larger power because of its vast storage in the platform SoC. Therefore, the headroom between the normal power consumption and its preset TDP is very little. According to the experience from our real-world experimental platform (Fig. 9), the SmartSSD is very prone to generate heat and it must instrument the active cooling device to emit away from the mass of heat. For this reason, the associate programmable logic cannot own the most powerful FPGA device. SmartSSD is only equipped with the Xilinx *xcku*15*p* device. Table X compares the resource utilization. The consumption of several resources like #BRAMs and #Regs in Poseidon-NDP is on par with the prior baselines, i.e., HEAX, Medha, and FAB. Although they own the more powerful FPGA devices, Poseidon-NDP still provides the better performance compared to Medha and HEAX.

VI. CONCLUSION

In this article, we present a co-design solution for FHE acceleration. We investigate the execution of FHE and observe that the FHE performance suffers from the intractable data movement and high-computation intensity. Based on such analysis, we propose a novel optimization technique for the FHE algorithm named "NTT-fusion," which significantly reduces the computational complexity of the key operator in FHE. Based on the NTT-fusion, we proposed an NDP-based FHE accelerator named "Poseidon-NDP." Poseidon-NDP includes abundant configuration choices and provides flexible scalability, which allows the Poseidon-NDP structure to explore the best-performance/power/bandwidth tradeoff and meet the various requirements in different platforms. In this work, we evaluate the Poseidon-NDP design on SmartSSD,

which is a practical NDP device, and prove the effectiveness of Poseidon-NDP. We hope this work can inspire new ideas for future FHE accelerator designs, by adjusting the FHE algorithm in combination with hardware characteristics, digging out the deep optimization space to design hardware-oriented FHE accelerators efficiently.

REFERENCES

[1] "Vitis rev." Jan. 2021. [Online]. Available: https://www.xilinx.com/products/design-tools/vitis.html

[2] "Vivado rev." Jan. 2021. [Online]. Available: https://www.xilinx.com/products/design-tools/vivado.html

[3] "XilinxRuntime rev." Jan. 2021. [Online]. Available: https://www.xilinx.com/products/design-tools/vitis/xrt.html

[4] R. Agrawal et al., "FAB: An FPGA-based accelerator for bootstrappable fully homomorphic encryption," in *Proc. IEEE Int. Symp. High-Perform. Comput. Architecture (HPCA)*, 2023, pp. 882–895.

[5] S. Akleylek, Ö. Dağdelen, and Z. Yüce Tok, "On the efficiency of polynomial multiplication for lattice-based cryptography on GPUs using CUDA," in *Proc. Int. Conf. Cryptogr. Inf. Security*, 2015, pp. 155–168.

[6] A. Al Badawi, B. Veeravalli, C. F. Mun, and K. M. M. Aung, "High-performance FV somewhat homomorphic encryption on GPUs: An implementation using CUDA," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2018, no. 2, pp. 70–95, 2018.

[7] J.-C. Bajard, J. Eynard, M. A. Hasan, and V. Zucca, "A full RNS variant of FV like somewhat homomorphic encryption schemes," in *Proc. Int. Conf. Sel. Areas Cryptogr.*, 2016, pp. 423–442.

[8] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," *ACM Trans. Comput. Theory*, vol. 6, no. 3, pp. 1–36, 2014.

[9] X. Cao, C. Moore, M. O'Neill, E. O'Sullivan, and N. Hanley, "Optimised multiplication architectures for accelerating fully homomorphic encryption," *IEEE Trans. Comput.*, vol. 65, no. 9, pp. 2794–2806, 2015.

[10] X. Cao, C. Moore, M. O'Neill, N. Hanley, and E. O'Sullivan, "High-speed fully homomorphic encryption over the integers," in *Proc. Int. Conf. Financ. Cryptogr. Data Secur.*, 2014, pp. 169–180.

[11] X. Cao, C. Moore, M. O'Neill, E. O'Sullivan, and N. Hanley, "Accelerating fully homomorphic encryption over the integers with super-size hardware multiplier and modular reduction," IACR Cryptol. ePrint Arch., Bellevue, WA, USA, Rep. 2013/616, 2013.

[12] D. D. Chen et al., "High-speed polynomial multiplication architecture for ring-LWE and SHE cryptosystems," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 62, no. 1, pp. 157–166, Jan. 2015.

[13] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A full RNS variant of approximate homomorphic encryption," in *Proc. Int. Conf. Sel. Areas Cryptogr.*, 2018, pp. 347–368.

[14] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Security*, 2017, pp. 409–437.

[15] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachene, "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Security*, 2016, pp. 3–33.

[16] W. Dai and B. Sunar, "cuHE: A homomorphic encryption accelerator library," in *Proc. Int. Conf. Cryptogr. Inf. Security Balkans*, 2015, pp. 169–186.

[17] L. Deng, "The MNIST database of handwritten digit images for machine learning research [best of the Web]," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 141–142, Nov. 2012.

[18] D. Naveen. "Logistic regression to predict heart disease." Jun. 2019. [Online]. Available: https://www.kaggle.com/datasets/dileep070/heart-disease-prediction-using-logistic-regression

[19] Y. Doröz, E. Öztürk, and B. Sunar, "Evaluating the hardware performance of a million-bit multiplier," in *Proc. Euromicro Conf. Digit. Syst. Design*, 2013, pp. 955–962.

[20] Y. Doröz, E. Öztürk, and B. Sunar, "Accelerating fully homomorphic encryption in hardware," *IEEE Trans. Comput.*, vol. 64, no. 6, pp. 1509–1521, Jun. 2015.

[21] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," Cryptol. ePrint Arch., IACR, Bellevue, WA, USA, Rep. 2012/144, 2012.

[22] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. 41st Annu. ACM Symp. Theory Comput.*, 2009, pp. 169–178.

[23] K. Han and D. Ki, "Better bootstrapping for approximate homomorphic encryption," in *Proc. Cryptograph. Track RSA Conf.*, 2020, pp. 364–390.

[24] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*. New York, NY, USA: Springer, 2004.

[25] "He benchmarking framework: HEBench." 2023. [Online]. Available: https://hebench.github.io/

[26] "Intel® VTune™ profiler." Intel. 2023. [Online]. Available: https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.html

[27] W. Jung, S. Kim, J. H. Ahn, J. H. Cheon, and Y. Lee, "Over 100x faster bootstrapping in fully homomorphic encryption through memory-centric optimization with GPUs," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2021, no. 4, pp. 114–148, 2021.

[28] W. Jung et al., "Accelerating fully homomorphic encryption through architecture-centric analysis and optimization," *IEEE Access*, vol. 9, pp. 98772–98789, 2021.

[29] A. Khedr and G. Gulak, "Homomorphic processing unit (HPU) for accelerating secure computations under homomorphic encryption," U.S. Patent 10 298 385, 2019.

[30] J. Kim et al., "ARK: Fully homomorphic encryption accelerator with runtime data generation and inter-operation key reuse," 2022, *arXiv:2205.00922*.

[31] S. Kim, W. Jung, J. Park, and J. H. Ahn, "Accelerating number theoretic transformations for bootstrappable homomorphic encryption on GPUs," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, 2020, pp. 264–275.

[32] S. Kim et al., "BTS: An accelerator for bootstrappable fully homomorphic encryption," in *Proc. 49th Annu. Int. Symp. Comput. Archit.*, 2022, pp. 711–725.

[33] S. Kim, K. Lee, W. Cho, J. H. Cheon, and R. A. Rutenbar, "FPGA-based accelerators of fully pipelined modular multipliers for homomorphic encryption," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig)*, 2019, pp. 1–8.

[34] S. Kim, K. Lee, W. Cho, Y. Nam, J. H. Cheon, and R. A. Rutenbar, "Hardware architecture of a number theoretic transform for a bootstrappable RNS-based homomorphic encryption scheme," in *Proc. IEEE 28th Annu. Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM)*, 2020, pp. 56–64.

[35] A. C. Mert et al., "Medha: Microcoded hardware accelerator for computing on encrypted data," 2022, *arXiv:2210.05476*.

[36] C. Mouchet, J.-P. Bossuat, J. Troncoso-Pastoriza, and J. Hubaux, "Lattigo: A multiparty homomorphic encryption library in go," in *Proc. 8th Workshop Encrypted Comput. Appl. Homomorphic Cryptogr.*, 2020, pp. 1–6.

[37] "Nucyper, Nufhe, a GPU-powered torus the implementation." Accessed: Mar. 10, 2022. [Online]. Available: https://github.com/nucypher/nufhe

[38] E. Öztürk, Y. Doröz, E. Savaş, and B. Sunar, "A custom accelerator for homomorphic encryption applications," *IEEE Trans. Comput.*, vol. 66, no. 1, pp. 3–16, Jan. 2017.

[39] M. S. Riazi, K. Laine, B. Pelton, and W. Dai, "HEAX: An architecture for computing on encrypted data," in *Proc. 25th Int. Conf. Archit. Support Program. Languages Oper. Syst.*, 2020, pp. 1295–1309.

[40] S. S. Roy, K. Järvinen, J. Vliegen, F. Vercauteren, and I. Verbauwhede, "HEPCloud: An FPGA-based multicore processor for FV somewhat homomorphic function evaluation," *IEEE Trans. Comput.*, vol. 67, no. 11, pp. 1637–1650, Nov. 2018.

[41] S. S. Roy, F. Turan, K. Jarvinen, F. Vercauteren, and I. Verbauwhede, "FPGA-based high-performance parallel architecture for homomorphic computing on encrypted data," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2019, pp. 387–398.

[42] S. S. Roy, F. Vercauteren, N. Mentens, D. D. Chen, and I. Verbauwhede, "Compact ring-LWE cryptoprocessor," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst.*, 2014, pp. 371–391.

[43] N. Samardzic et al., "F1: A fast and programmable accelerator for fully homomorphic encryption," in *Proc. 54th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2021, pp. 238–252.

[44] N. Samardzic et al., "CraterLake: A hardware accelerator for efficient unbounded computation on encrypted data," in *Proc. ISCA*, 2022, pp. 173–187.

[45] (Microsoft Res., Redmond, WA, USA).*Microsoft SEAL (Release 4.0)*. (Mar. 2022). [Online]. Available: https://github.com/Microsoft/SEAL

[46] Vernamlab. "CUDA-accelerated fully homomorphic encryption library." Accessed: Mar. 15, 2022. [Online]. Available: https://github.com/vernamlab/cuFHE

[47] W. Wang, Z. Chen, and X. Huang, "Accelerating leveled fully homomorphic encryption using GPU," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2014, pp. 2800–2803.
[48] W. Wang, Y. Hu, L. Chen, X. Huang, and B. Sunar, "Exploring the feasibility of fully homomorphic encryption," *IEEE Trans. Comput.*, vol. 64, no. 3, pp. 698–706, Mar. 2015.
[49] W. Wang, X. Huang, N. Emmart, and C. Weems, "VLSI design of a large-number multiplier for fully homomorphic encryption," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 9, pp. 1879–1887, Sep. 2014.
[50] "Product brief of SmartSSD." Xilinx. Accessed: Sep. 2, 2022. [Online]. Available: https://www.xilinx.com/content/dam/xilinx/publications/product-briefs/xilinx-smartssd-computational-storage-drive-product-brief.pdf
[51] Y. Zhai et al., "Accelerating encrypted computing on Intel GPUs," 2021, *arXiv:2109.14704*.

Yinghao Yang is currently pursuing the Doctoral degree with the Institute of Computing Technology, Chinese Academy of Sciences, University of Chinese Academy of Sciences, Beijing, China.

His research interests include fully homomorphic encryption acceleration, FPGA accelerator design, and fully homomorphic processor design.

Hang Lu received the B.S. and M.S. degrees in electronic information engineering from the Beijing University of Aeronautics and Astronautics, Beijing, China, in 2008 and 2011, respectively, and the Ph.D. degree in computer science from the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing, China, in 2015.

He is currently an Associate Professor and a Master Tutor with the ICT, CAS, University of Chinese Academy of Sciences, Beijing. He is also a Research Scientist with Shanghai Innovation Center for Processor Technologies, Beijing. His research interests include fully homomorphic encryption acceleration, FPGA accelerator design, and fully homomorphic processor design.

Dr. Lu is a member of the Youth Innovation Promotion Association of CAS, and the New Best Star of ICT.

Xiaowei Li (Senior Member, IEEE) received the B.Eng. and M.Eng. degrees in computer science from the Hefei University of Technology, Hefei, China, in 1985 and 1988, respectively, and the Ph.D. degree in computer science from the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing, China, in 1991.

He was an Associate Professor with the Department of Computer Science and Technology, Peking University, Beijing, from 1991 to 2000. In 2000, he joined ICT, CAS, as a Professor, where he is currently the Deputy Director of the State Key Laboratory of Computer Architecture. He has coauthored over 280 papers in journals and international conferences, and holds 60 patents and 30 software copyrights. His current research interests include VLSI testing, design for testability, design verification, dependable computing, and wireless sensor networks.

Prof. Li has been the Vice Chair of the EEE Asia and Pacific Regional Test Technology Technical Council since 2004. He was the Chair of the Technical Committee on Fault-Tolerant Computing, China Computer Federation from 2008 to 2012, and the Steering Committee Chair of IEEE Asian Test Symposium from 2011 to 2013. He was the Steering Committee Chair of IEEE Workshop on RTL and High-Level Testing from 2007 to 2010. He serves as an Associate Editor for the *Journal of Computer Science and Technology*, the *Journal of Low Power Electronics*, the *Journal of Electronic Testing: Theory and Applications*, and the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS.