MCHEAS: Optimizing Large-Parameter NTT Over Multi-Cluster In-Situ FHE Accelerating System

Zhenyu Guan, *Member, IEEE*, Yongqing Zhu, *Student Member, IEEE*, Luchang Lei, *Student Member, IEEE*, Hongyang Jia, *Member, IEEE*, Yi Chen, Bo Zhang, Changrui Ren, Jin Dong, Song Bian, *Member, IEEE*

***Abstract*—Fully Homomorphic encryption (FHE) enables high-level security but with a heavy computation workload, necessitating software-hardware co-design for aggressive acceleration. Recent works on specialized accelerators for HE evaluation have made significant progress in supporting lightweight RNS-CKKS applications, especially those with high-density in-memory computing techniques. To fulfill higher computational demands for more general applications, this paper proposes MCHEAS, an accelerating system comprising multiple in-situ HE processing accelerators, each functioning as a cluster to perform large-parameter RNS-CKKS evaluation collaboratively. MCHEAS features optimization strategies including the synchronous, preemptive swap, square-diagonal, and odd-even index separation. Using these strategies to compile the computation and transmission of number theoretic transform (NTT) coefficients, the method optimizes the inter-cluster data swaps, a major bottleneck in NTT computations. Evaluations show that under 1 GHz, with different inter-cluster data transfer bandwidths, our approach accelerates NTT computations by 26.40% to 51.75%. MCHEAS also improves computing unit utilization by 10.30% to 33.97%, with a maximum peak utilization rate of up to 99.62%. MCHEAS achieves 17.63% to 34.67% speedups for HE operations involving NTT, and 15.12% to 30.62% speedups for demonstrated applications, while enhancing the computing units' utilization by 5.18% to 21.87% during application execution. Furthermore, we compare MCHEAS with SOTA designs under a specific inter-cluster data transfer bandwidth, achieving up to 81.45× their area efficiencies in applications.**

***Index Terms*—Fully homomorphic encryption (FHE), Number theoretic transform (NTT), Cryptographic accelerating system, In-situ computing.**

I. Introduction

The introduction of fully homomorphic encryption (FHE) [1] in 2009 offered a secure but computationally expensive way to compute encrypted data. In the following decade of research, scholars proposed more efficient and practical schemes such as BGV [2], B/FV [3], CKKS [4], and TFHE [5]. Almost all FHE schemes are based on lattice-based problems [6], [7], offering high security and are more resistant to quantum attacks [8], [9]. Due to these advantages, FHE has been continuously studied and developed by researchers [10]–[13], with CKKS and TFHE remaining popular choices that find applications across various domains [14]–[22], meeting different privacy protection requirements in various scenarios.

To achieve high security and greater computational capacity, ring-learning with errors (RLWE) FHE including BGV, BFV, CKKS, commonly employs large parameters for computation, resulting in extremely high computational complexity and significant processing delays. To mitigate this issue, various hardware acceleration solutions have been proposed, which can generally be classified into two categories. One focuses on utilizing extensive resources (e.g., large chip area) to design architectures capable of supporting computations with high dimension which typically reaches 65536, achieving considerable speeds, and meeting higher computational requirements, such as F1 [23], ARK [24], and BTS [25], et al. [26], [27]. In contrast, the other targets practical tape-out with area-constrained architectures, usually suitable for lightweight HE evaluations with smaller parameters (e.g., smaller dimensions). For instance, Yoon et al. have introduced a 55nm test-chip prototype supporting BGV encryption and decryption [28], but its dimension has been limited to 16. Das et al. have developed a 28nm CMOS HE engine for BFV encryption and decryption [29], achieving a dimension of 4096. Lei et al. have proposed a fabricated in-situ HE accelerator with a dimension of 4096 [30], supporting only 19-bit coefficients. Wang et al. have introduced CAEA, an accelerator for RNS-CKKS, synthesized using SMIC 40nm technology and implemented on FPGA [31], which supports a maximum dimension of 8192. Lee et al. have proposed an accelerator supporting RNS-CKKS [32], but its insufficient on-chip data storage limits it to achieving considerable performance only for HE evaluation with a maximum dimension of 16384[1], preventing optimal computational speeds for higher-dimensional calculations. Due to the use of smaller parameters in these lightweight HE accelerators, they are limited in their ability to compute deeper FHE operations, making it challenging to support more complex applications such as neural network inference with bootstrapping. To achieve a larger-parameter HE scheme, a common approach is to employ multiple chips to collectively implement large-parameter operators, such as Chiplets or multi-chip modules (MCMs) [33], [34], thereby constructing a multi-cluster system where each cluster corresponds to a chip.

Higher computational demands necessitate the use of multiple accelerators to support large-parameter HE evaluations for deeper computation circuits. However, due to the lower trans-

This work is partially supported by the National Key R&D Program of China (2023YFB3106200), the National Natural Science Foundation of China (62202028, 62172025, U2241213). This work is also supported by Huawei Technologies Co. and Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing.

Corresponding authors are Jin Dong and Song Bian.

Z. Guan, Y. Zhu and S. Bian are with School of Cyber Science and Technology, Beihang University, Beijing 100191, China (e-mail: {guanzhenyu, zhuyongqing100, sbian}@buaa.edu.cn).

L. Lei and H. Jia are with Department of Electronic Engineering, Tsinghua University, Beijing 100084, China (e-mail: llc22@mails.tsinghua.edu.cn; hjia@tsinghua.edu.cn).

Y. Chen, B. Zhang, C. Ren and J. Dong are with the Beijing Academy of Blockchain and Edge Computing, Beijing 100190, China (e-mail: {chenyi, zhangb, rencr, dongjin}@baec.org.cn).

[1]Estimated dimension when executing ciphertext addition.

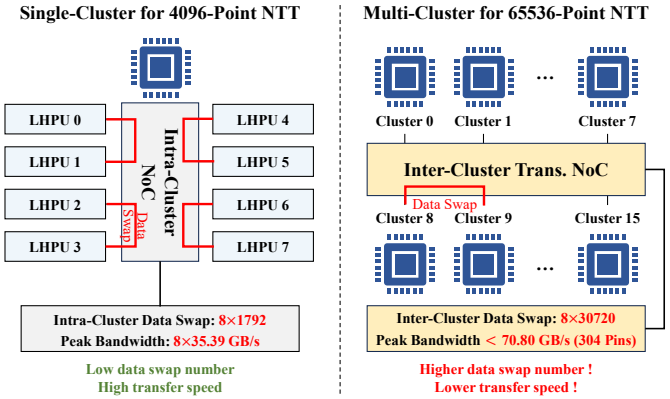


Fig. 1. The data swap numbers and transfer bandwidths of performing (a) 4096-point 38-bit NTT using one cluster (b) 65536-point 38-bit NTT using 16 clusters at 1 GHz, where LHPU refers to the ISHEPC described in [30].

mission bandwidth between accelerators compared to internal transmissions within an accelerator, collaborative use without data flow optimization would result in significant transmission delays. For example, we consider the accelerator sharing similar ideas in [30] which leverages eDRAM widely used for in-memory/in-situ computation [35]–[40], to empower edge devices with limited computing power to perform HE evaluation, effectively mitigating the "memory wall" issue [41], [42] and the high transmission latency caused by ciphertext expansion [43] in traditional FHE-based outsourcing computation schemes at a dimension of 4096. For larger-parameter evaluation, we treat each accelerator as a cluster and integrate multiple clusters into an extended accelerating system, using an inter-cluster NoC for data transfer, as shown in Fig. 1. When performing a 4096-point 38-bit number theoretic transform (NTT) using one cluster at 1 GHz, the number of data swaps required within the cluster is relatively low. Additionally, the intra-cluster high-speed NoC design ensures that these data swaps are completed at an extremely fast speed. In contrast, when using 16 clusters to perform a 65536-point 38-bit NTT, the number of inter-cluster data swaps is significantly higher. Moreover, since each cluster is a pre-fabricated chip designed for edge applications, and most edge chips are equipped with fewer than 300 pins [44], [45], the inter-cluster bandwidth is limited to less than 70.80 GB/s (corresponding to 304 pins), which is significantly lower than the intra-cluster bandwidth. A higher number of data swaps and lower transfer speed result in increased transmission latency. Experiments have evaluated the NTT computation performance under different numbers of pins used for data swapping within these clusters, which determined the data transfer bandwidth, ranging from 8.85 to 35.39 GB/s. The results reveal that the computing unit utilization of the multi-cluster system dropped to only 39.03% to 71.76%, indicating that transmission significantly restricts NTT computation speed. On the other hand, experiments show that at a bandwidth of 35.39 GB/s, the NTT time proportion can exceed 68% for keyswitch (KS) in RNS-CKKS ($N = 65536, L = 44$) and ResNet-20 [46], which shows accelerating NTT can effectively enhance the performance of HE evaluation using the system. Therefore, the inter-cluster transmission optimization for NTT is the key to tolerating the limited bandwidth, harnessing the gain of high-density HE processors in scenarios with higher computing demands.

To enable efficient collaboration of multiple clusters for large-parameter RNS-CKKS HE evaluation, this paper proposes multi-level NTT optimization strategies for reducing inter-cluster data transfer latencies and promoting computing unit utilization over a multi-cluster in-situ FHE accelerating system. The contributions of this paper are listed as follows:

- **Multi-cluster accelerating system for large-parameter HE evaluation**: We propose MCHEAS, a multi-cluster HE accelerating system comprising multiple in-situ HE processing accelerators sharing similar ideas with [30], each functioning as a cluster to perform large-parameter RNS-CKKS evaluation collaboratively. Even for CPUs with larger caches [47], [48], if configured like and computed for NTT, with a structure and characteristics similar to MCHEAS, optimization can still be achieved according to the strategies we have proposed.
- **NTT optimization strategies:** We propose multi-level NTT optimization strategies to address the significant time consumption of inter-cluster data swaps, including **the synchronous**, **preemptive swap**, **square-diagonal**, and **odd-even index separation**, and applying deep optimizations tailored to the phases of NTT.
- **Large-parameter RNS-CKKS evaluation:** We map large-parameter RNS-CKKS operators and operations onto MCHEAS for evaluation. Under a frequency of 1 GHz, with parameters of 65536-point 38-bit and inter-cluster data transfer bandwidths of 8.85 to 35.39 GB/s, our method achieves NTT computation speedups of 26.40% to 51.75%, computing unit utilization improvements of 10.30% to 33.97%, and a maximum peak utilization rate of 99.62%. HE operations involving NTT see speedups of 17.63% to 34.67%, and applications see speedups of 15.12% to 30.62%, and the strategies also improve the computing units' utilization by 5.18% to 21.87% during application execution. Furthermore, we compare MCHEAS, under an inter-cluster data transfer bandwidth of 26.54 GB/s, with SOTA designs, achieving up to 81.45× their area efficiencies in applications.

II. Preliminaries

A. Number Theoretic Transform

Number theoretic transform (NTT) is a widely-used method for accelerating polynomial multiplication. It can be comprehended as discrete Fourier transform (DFT) over a finite field, which is defined as Eq. (1), where $f[i]$ represents the coefficient of X^i in the polynomial $f(X)$, $F[j]$ represents the coefficient of X^j in $F(X) = \mathsf{NTT}[f(X)]$, and ω_N called twiddle factor is the N^{th} primitive root of unity in $\mathbb{Z}_q$. To perform inverse NTT (INTT), the exponents of the twiddle factors need to be changed to negative values, and the result is multiplied by $N^{-1} \bmod q$. Inverse NTT (INTT) is the inverse operation of NTT, which is defined as Eq. (2).

$$F[j] = \sum_{i=0}^{N-1} f[i]\omega_N^{ij} \bmod q \tag{1}$$

$$f[i] = N^{-1} \sum_{j=0}^{N-1} F[j]\omega_N^{-ij} \bmod q \tag{2}$$

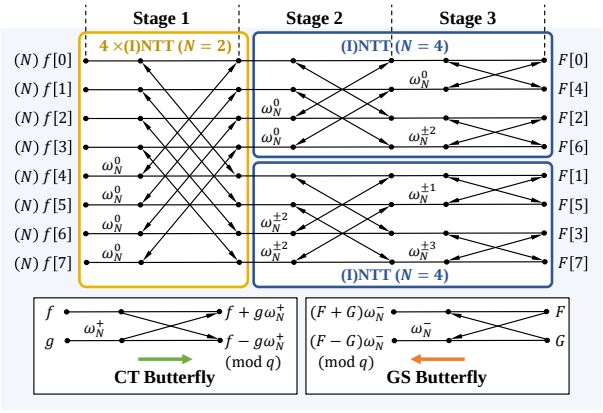


Fig. 2. An 8-point NTT circuit can be viewed as performing four 2-point NTTs and two 4-point NTTs in the appropriate order. The CT NTT and GS INTT are symmetrical and inverse to each other.

NTT butterfly operations are widely adopted to accelerate polynomial multiplications over R_q, reducing the computational complexity of the original NTT from $O(N^2)$ to $O(N \log N)$. The radix-2 butterflies are the most commonly used circuits, with Cooley-Tukey (CT) and Gentleman-Sande (GS) being the most prevalent types. Both two can be used for NTT and INTT calculations. When NTT is performed using CT butterflies and INTT is calculated using GS butterflies, the resulting circuit is entirely symmetrical. This property enables the compilation of reversible computation data flows. On the other hand, an N-point butterfly circuit can be viewed as sequentially completing smaller point NTTs, though the twiddle factors might differ. For example, an 8-point CT NTT circuit can be viewed as performing four 2-point NTTs and two 4-point NTTs in the appropriate order, as shown in Fig. 2. Additionally, the data flow in the figure from left to right represents the CT NTT process, while the flow from right to left corresponds to the GS INTT process.

B. RLWE Ciphertexts of Fully Homomorphic Encryption

At present, fully homomorphic encryption (FHE) schemes predominantly contain two types of ciphertexts, including LWE(Learning with Errors)-based and RLWE(Ring-Learning with Errors)-based. In RLWE-based homomorphic encryption, plaintext and ciphertext polynomials are defined over the ring $R_Q = \mathbb{Z}_Q[X]/(X^N + 1)$, where N is an integer that is a power of 2, $\mathbb{Z}_Q[X]$ is the ring of polynomials with coefficients modulo Q, with Q being either a large prime or the product of several different primes, especially in the case of using a residue number system (RNS). If Q is a prime, it must satisfy $Q \equiv 1 \bmod 2N$; If Q is a product of primes, each prime factor q_i composing Q must satisfy $q_i \equiv 1 \bmod 2N$. To encrypt a plaintext polynomial $m(X)$ ("(X)" is omitted in some expressions for simplicity), choose a random polynomial $a \leftarrow U(R_Q)$, an error polynomial $e \leftarrow \Psi$ and a secret polynomial $s \leftarrow \chi$, then calculate $b = -a \cdot s + \Delta \cdot m + e \bmod Q$, where the scaling factor Δ is a larger integer less than Q, and $(b, a) \in R_Q^2$ is the ciphertext of m. Because ciphertext and plaintext are defined over the polynomial ring R_Q, many operators have a significant number of (I)NTTs for optimization, as mentioned above.

C. RNS-CKKS Homomorphic Operations

An FHE scheme consisted of a series of basic homomorphic operations. For example, we consider the full-RNS (residue number systems) version of CKKS [49], [50], a robust and widely adopted RLWE-based approach. To support more complex computations, RNS-CKKS primarily includes the following homomorphic operations: homomorphic addition, homomorphic multiplication, rescale, keyswitch, rotation and bootstrapping. These operations can be decomposed into four basic operators: modular addition (ModAdd), modular multiplication (ModMult), NTT/INTT and Automorphism.

Notation: For a fixed power-of-two N and $L+1$ distinct primes $q_0, q_1, ..., q_L$ where L is the level, we define $Q_L = \prod_{i=0}^{L} q_i$ and $R_{Q_L} = \mathbb{Z}_{Q_L}[X]/(X^N+1)$, the cyclotomic polynomial ring over the integers modulo Q_L. Due to operations such as "rescale", the level L might change to l. An element $f(X)$ in the ring R_{Q_l} can be represented in the RNS domain as follows: $(f^{(0)}, f^{(1)}, ..., f^{(l)}) \in \prod_{j=0}^{l} R_{q_j} \cong R_{Q_l}$.

• **Homomorphic Addition (HAdd):** There are two cases of homomorphic addition: ciphertext-plaintext and ciphertext-ciphertext. For ciphertext-plaintext addition, given a ciphertext $\mathsf{ct} = (c_0, c_1) \in \prod_{j=0}^{l} R_{q_j}^2$ and an encoded plaintext $\mathsf{pt} \in \prod_{j=0}^{l} R_{q_j}$, the process can be represented as Eq. (3), where "+" denotes the polynomial addition of the two polynomials over the ring $\prod_{j=0}^{l} R_{q_j}$. In this example, "+" is expressed as $(c_0^{(j)} + \mathsf{pt}^{(j)}) \bmod q_j$ for all j. Similarly, given another ciphertext, $\mathsf{ct'} = (c_0', c_1')$, the ciphertext-ciphertext addition can be represented as Eq. (4). Both homomorphic additions obviously involve ModAdd, commonly used in HE operations.

$$\mathsf{AddPlain(ct,pt)} = (c_0 + \mathsf{pt}, c_1) \tag{3}$$

$$\mathsf{Add(ct,ct')} = (c_0 + c_0', c_1 + c_1') \tag{4}$$

• **Ciphertext-Plaintext Multiplication (PMult):** Similar to homomorphic addition, homomorphic multiplication also has two cases. For ciphertext-plaintext multiplication, given a ciphertext $\mathsf{ct} = (c_0, c_1)$ and an encoded plaintext pt, the process can be represented as Eq. (5), where "$\cdot$" denotes the polynomial multiplication. Normally, we can accelerate polynomial multiplication using NTT/INTT, as mentioned in Sec. II-A. Therefore, under this example, $c_0 \cdot \mathsf{pt}$ can be expressed as $\mathsf{INTT}(\mathsf{NTT}(c_0)^{(j)} \odot \mathsf{NTT}(\mathsf{pt})^{(j)}) \bmod q_j$ for all j, where "$\odot$" denotes component-wise modular multiplication. This operation involves two basic operators: NTT and ModMult. These two operators are also key to constructing homomorphic operations. In contrast to ModAdd, the modular reduction in ModMult requires more complex algorithms for implementation, such as the NTT-friendly Montgomery reduction [51].

$$\mathsf{Pmult(ct,pt)} = (c_0 \cdot \mathsf{pt}, c_1 \cdot \mathsf{pt}) \tag{5}$$

• **Rescale (RS):** Following certain homomorphic operations, the scaling factor of the ciphertext significantly increases. For instance, the scaling factor is increased to Δ^2 after PMult. Consequently, it is necessary to reduce this scaling factor by Δ. Given a level l ciphertext $\mathsf{ct} = (\mathsf{ct}^{(j)} = (c_0^{(j)}, c_1^{(j)}))_{0 \le j \le l} \in \prod_{j=0}^{l} R_{q_j}$, this process can be described as Eq. (6), where $c_i'^{(j)} \leftarrow q_l^{-1} \cdot (c_i^{(j)} - c_i^{(l)}) \bmod q_j$ for $i = 0, 1$.

$$\mathsf{RS(ct)} = \left(\mathsf{ct}'^{(j)} = (c_0'^{(j)}, c_1'^{(j)})\right)_{0\le j\le l-1} \in \prod_{j=0}^{l-1} R_{q_j} \quad (6)$$

• **Keyswitch (KS):** This operation derives keyswitch keys from the original secret s and transforms the ciphertext into a new one encrypted with these keys, facilitating more complex computations. RNS-CKKS keyswitch requires sub-operations to be implemented: approximate modulus raising (ModUp) and reduction (ModDown). These sub-operations essentially perform modulus switching on the ciphertext over R_{Q_L}. The steps of a keyswitch on a ciphertext $\mathsf{ct} = (c_0, c_1)$ using the keyswitch key ksk are shown as Eq. (7), where $\mathsf{SwitchKey}(a, \mathsf{ksk})$ including ModUp and ModDown means converting the polynomial a into an intermediate ciphertext.

$$\mathsf{KS(ct,ksk)} = (c_0, 0) + \mathsf{SwitchKey}(c_1, \mathsf{ksk}) \quad (7)$$

• **Ciphertext-Ciphertext Multiplication (CMult):** Due to the format of RLWE ciphertexts, ciphertext-ciphertext multiplication is not as straightforward as PMult. For the multiplication of ciphertexts $\mathsf{ct} = (c_0, c_1)$ and $\mathsf{ct'} = (c_0', c_1')$ by a relinearization key rlk, the process can be described by Eq. (8), where $(d_0, d_1, d_2) = (c_0c_0', c_0c_1' + c_0'c_1, c_1c_1')$.

$$\mathsf{CMult(ct, ct', rlk)} = (d_0, d_1) + \mathsf{SwitchKey}(d_2, \mathsf{rlk}) \quad (8)$$

• **Rotation (Rotate):** The rotation operation in RNS-CKKS is defined by the automorphism $\phi_k : X \to X^k \mod X^N + 1$, whose index mapping operation maps the current index i of the ciphertext polynomial to $i \cdot k \mod N$ as shown in Eq. (9), where the messages are rotated by k positions to the left. The k-position rotation of a ciphertext $\mathsf{ct} = (c_0, c_1)$ with a rotation key rot_k is described as Eq. (10).

$$\phi_k(f(X)) = \sum_{i=0}^{N-1} sgn[i] f[i] X^{[i\cdot k]_N}$$
$$sgn[i] = \begin{cases} -1 \text{ if } i \cdot k \bmod 2N > N \\ 1 \text{ if } i \cdot k \bmod 2N < N \end{cases} \quad (9)$$

$$\mathsf{Rotate}_k(\mathsf{ct}, \mathsf{rot}_k) = \mathsf{KS}(\phi_k(\mathsf{ct}), \mathsf{rot}_k) \quad (10)$$

• **Bootstrapping (Boot):** After performing homomorphic multiplication followed by rescaling, the level of the ciphertext progressively decreases. To continue performing homomorphic operations, it is necessary to use "bootstrapping" to increase the ciphertext level for subsequent computations. Although this operation is complicated, it can be decomposed into all these four basic operators as mentioned above: ModAdd, ModMult, NTT/INTT and Automorphism.

III. Motivation

To enable HE evaluation with larger parameters, one viable approach is to coordinate multiple chips (clusters) as an accelerating system to perform basic operators. Notably, while extending bit width does not pose a significant challenge, the main issue lies in the inter-cluster transmission bandwidth, which struggles to match the speed of the intra-cluster bandwidth, resulting in inter-cluster speed being significantly lower than intra-cluster's. When performing large-point NTTs across multiple clusters, inter-cluster data swaps are inevitable. If the same method as intra-cluster data swap is applied—waiting for all stages to complete before inter-cluster data swaps—this results in substantial delays, impacting the overall NTT computation speed. In addition, larger-dimension NTTs involve more data swaps. When computing a 65536-point NTT using the 16-cluster accelerating system, the number of data swaps required by the inter-cluster NoC is 8.28× the total number of data swaps needed for a 4096-point NTT using a single-cluster system. For example, using the in-situ accelerator (cluster) sharing similar ideas in [30], when performing a 4096-point 38-bit NTT within a single cluster, the peak utilization of the computing units can reach 99.49%, with data swap accounting for only 0.51%, as shown in Fig. 3 (a). Thus, it is not a limiting factor for NTT computation speed. However, Fig. 3 (b) illustrates that when using 16 clusters to achieve a 65536-point NTT at 1 GHz, with inter-cluster transmission bandwidths ranging from 8.85 to 35.39 GB/s, the computing unit utilization drops to only 39.03% to 71.76%, making transmission time a critical limiting factor for NTT computation.

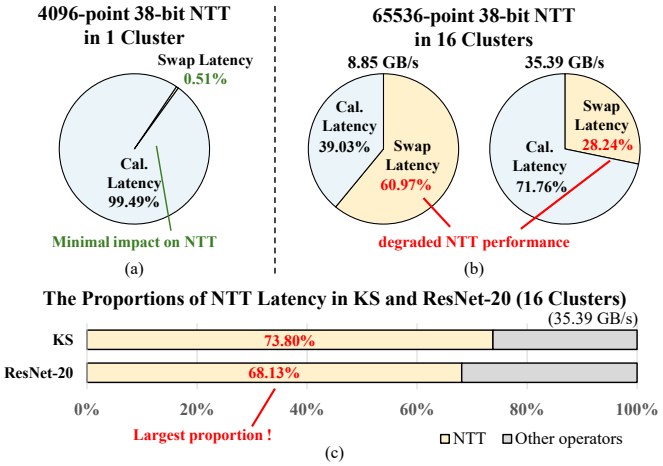


Fig. 3. The proportions of swap latency of (a) 4096-point 38-bit NTT in 1 cluster and (b) 65536-point 38-bit NTT in 16 clusters with 8.85 GB/s and 35.39 GB/s inter-cluster data transfer bandwidths. (c) The proportions of NTT latency in KS and ResNet-20 under 35.39 GB/s bandwidth (16 clusters).

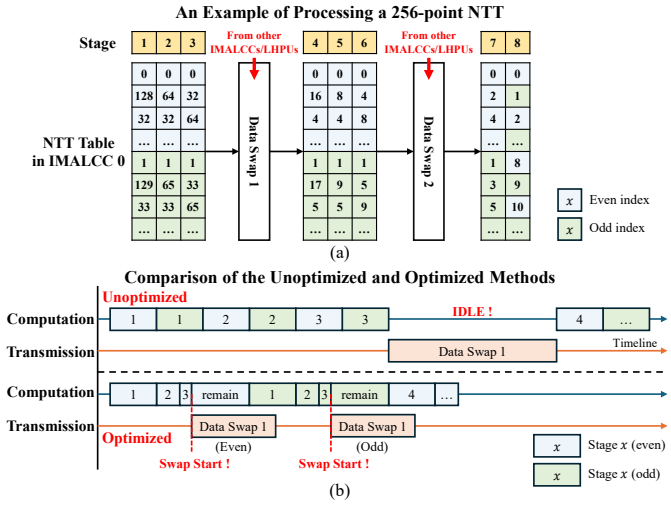


Fig. 4. (a) An example of processing a 256-point NTT, divided into 8-point, 8-point, and 4-point NTTs, where IMALCC refers to DIMPE in [30]. In this example, each IMALCC stores two groups of eight coefficients for performing the 8-point NTT, which are represented separately by even and odd indices. (b) Comparison of the computation and transmission timelines between the unoptimized and optimized methods when processing a 256-point NTT.

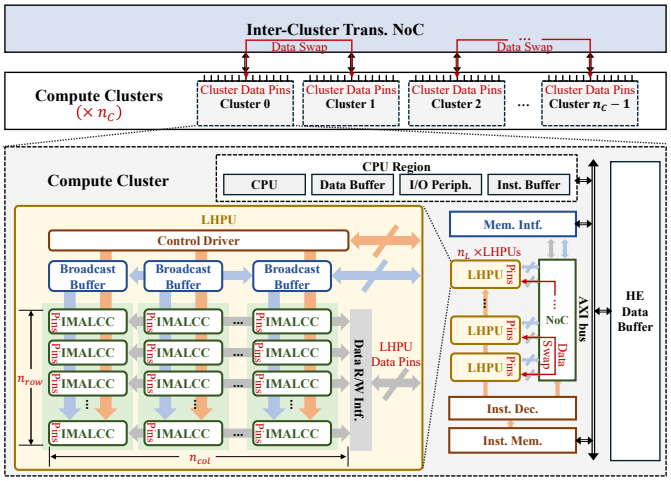


Fig. 5. The architecture of the proposed multi-cluster in-situ FHE accelerating system.

On the other hand, when using the multi-cluster system to speed up HE operations and applications, NTT remains the most time-consuming operator. Fig. 3 (c) indicates that, at a bandwidth of 35.39 GB/s, the NTT time proportion can reach over 68% for KS ($N = 65536, L = 44$) and ResNet-20 [46], which means accelerating NTT can significantly enhance the performance of HE evaluation using the multi-cluster system. Fig. 4 illustrates a specific example. As shown in Fig. 4 (a), using a scaled accelerator based on [30] to compute a 256-point NTT requires two data swaps between the three smaller-point NTTs. The unoptimized method from [30] performs data swaps only after completing all butterfly operations within each stage as shown in Fig. 4 (b). While intra-cluster bandwidth is sufficient to prevent significant performance degradation, inter-cluster transmission, with its lower bandwidth, suffers from higher latency. A straightforward solution is to begin swapping data earlier in the final stage, though further optimization opportunities remain. Furthermore, most existing FPGA/ASIC accelerators follow a Von Neumann architecture [18], [23]–[26], and, along with differences in in-memory designs such as [52], the coefficient ordering modules in these designs are often optimized only for their own or similar architectures, such as F1's Transpose Unit. This limits the applicability of their data reordering optimizations in a multi-cluster system based on [30]. Thus, optimized scheduling strategies for the system are needed to reduce inter-cluster data swap delay, improve computing unit utilization, and enhance overall NTT and HE performance. These strategies should aim to start data swaps as early as possible while ensuring that the accelerating system remains less idle during the swap process.

IV. System Overview

Fig. 5 presents the architecture of our proposed multi-cluster HE accelerating system (MCHEAS). The compute clusters are the core components of MCHEAS, where the n_L local homomorphic processing units (LHPUs) serve as the principal functional units. Each LHPU comprises $n_{row} \times n_{col}$ in-memory atomic logic computation cores (IMALCCs), with the IMALCC representing the basic storage and computational unit within the architecture. Each IMALCC comprises some RAM for storage and logic gates for calculation. Meanwhile,

Algorithm 1: The grouping algorithm for the 1^{st} phase of CT NTT

Input: N, p_s
Input: A coefficient index i
Output: The group number grp of the coefficient index i
1: $n_{gap} \leftarrow N/(2^{p_s})$
2: $grp \leftarrow i \bmod n_{gap}$

the extensive parallelism of IMALCCs is also crucial to enhancing the computation speed of HE evaluations. The intra-cluster transmission NoCs, including intra-LHPU's and inter-LHPU's, facilitate in-memory swaps and high-bandwidth data exchanges. This design enables LHPU to efficiently manage both local and long-range butterfly operations in staged (I)NTT computations, as well as matrix transpose operations during automorphisms. When data requires intra-cluster transmission or data swapping, an IMALCC utilizes its own data pins or the pins of the LHPU it resides in to transfer the data to the target NoC for communication. Each cluster also comprises an HE data buffer and a CPU with peripherals. For a cluster to perform a specific NTT computation task, the NTT executed within a cluster can be decomposed into 3 phases $[p_s, p_s, p'_s]$, where $0 < p'_s \leq p_s$ and $\log_2 N = p'_s + \Sigma p_s$. In this setup, each IMALCC stores $2^{(p_s+1)}$ coefficients. The inter-cluster transmission NoC acts as the connection between various clusters for large-parameter operators, such as NTT and automorphism. The transmission process can be viewed as different clusters transferring data to the NoC via their respective data pins for swapping. However, since each cluster is a pre-fabricated chip, it lacks the numerous pins available to LHPU and IMALCC, leading to slower transmission speed and making it challenging to match the data transfer bandwidth of the intra-cluster NoC. Therefore, it is imperative to optimize the computation and transmission steps of NTT, which involve substantial inter-cluster data transfer and account for a significant proportion of the execution time in most applications.

V. NTT Optimization

As mentioned previously, when using multiple clusters to perform larger-point NTT, the inter-cluster transmission time becomes the primary limiting factor for NTT computation speed due to the lack of high bandwidth comparable to that of intra-cluster transmission NoC. Therefore, an optimized method for NTT data swapping is needed to reduce the additional time caused by data transfer and enhance the utilization of computing units, ultimately improving speed.

A. Optimization Setup

As mentioned in Sec. II-A, a large-point NTT can be decomposed into smaller-point NTTs. Since the NTT is performed in phases, we define a phase-stage configuration as $[p_s, p_s, ..., p_s, p'_s]$, where $0 < p'_s \leq p_s$ and $\log_2 N = p'_s + \Sigma p_s$, to represent the number of NTT stages executed in each phase. Taking the computation of NTT using the CT butterfly circuit as an example, we refer to the row of the circuit where the input $f[i]$ is located as the coefficient index i. This means that regardless of the stage the circuit is currently in, any coefficient of the circuit that is aligned with the input $f[i]$ in the same row retains the index i. Based on the given phase-stage

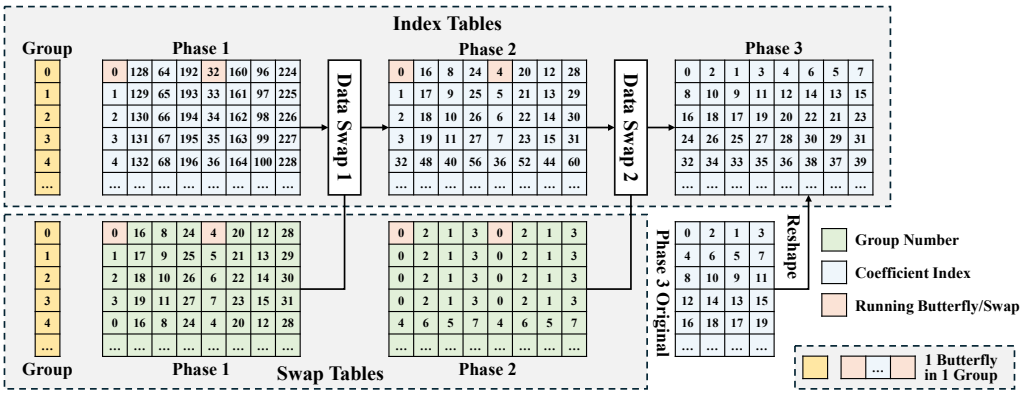


Fig. 6. The NTT index tables, swap tables, and the reshaping procedure for phase 3 under the phase-stage configuration [3,3,2].

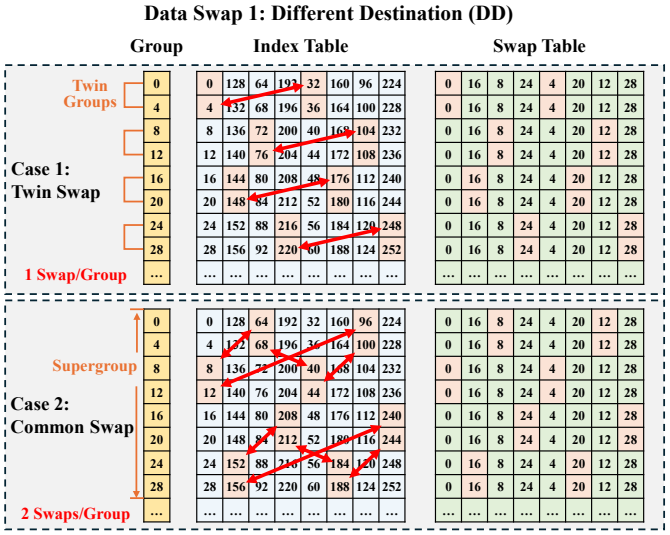


Fig. 7. Two cases of the data swap 1 (DD) under the configuration [3,3,2]: twin swap and common swap.

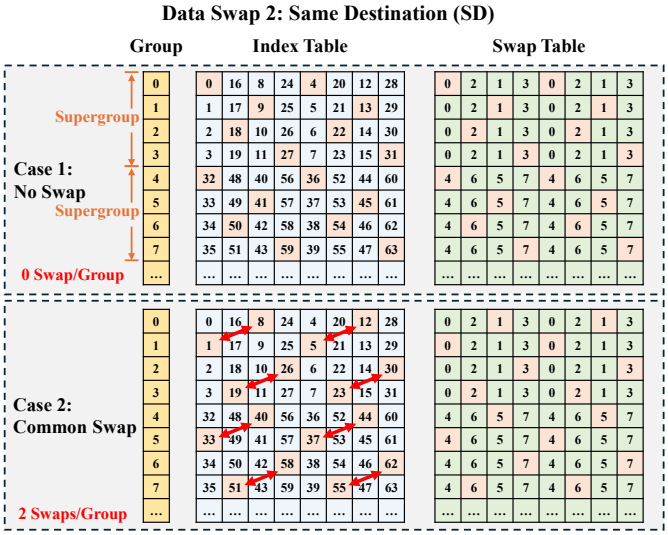


Fig. 8. Two cases of the data swap 2 (SD) under the configuration [3,3,2]: no swap and common swap.

configuration, all coefficient indices are divided into multiple groups to ensure that each group can independently complete a 2^{p_s} or $2^{p'_s}$-point NTT within each phase. The grouping method for the 1^{st} phase of CT NTT is detailed in Algorithm 1. When $p'_s < p_s$, the original group containing $2^{p'_s}$ indices needs to be reshaped into a new group containing 2^{p_s} indices. An index table and a swap table are generated based on the coefficient indices of each phase and their group numbers. The index table indicates which group each coefficient belongs to in the current phase, while the swap table shows which group the coefficient at this position in the current phase will be transferred to in the next phase. For the configuration [3,3,2], the index tables, swap tables, and the reshaping procedure for phase 3 are illustrated in Fig. 6, where the indices within a group are arranged in ascending order after being bit-reversed in each index table.

B. Stage-Level Optimization

For an accelerating system that executes butterfly computations serially, the most straightforward approach to reduce transmission-induced delays is to allow the immediate data swap of the two relevant coefficients as soon as a butterfly computation is completed during the last stage of each phase, which avoids waiting for the entire stage to finish before initiating data transfer. To achieve this, multiple groups involved in the approaching data swaps must complete their respective butterfly computations simultaneously, named **synchronous strategy**. Therefore, the butterfly computation order within each group needs to be carefully planned and controlled. Based on whether the destinations of the two coefficients from a butterfly computation are within the same group or different groups, we classify data swaps into two categories: different destination (DD) and same destination (SD).

When the two phases of a CT NTT have the same number of stages (both p_s or $p'_s = p_s$), or when the inverse of this CT NTT (GS INTT) is performed, the data swap between these phases is classified as DD. In this scenario, data swaps are divided into two cases: twin swap and common swap. For example, under the configuration [3,3,2], data swap 1 (DD) is illustrated in Fig. 7, where each group calculates butterflies simultaneously. From the figure, based on the group numbers within a single row of the swap table, these groups can be viewed as forming a supergroup. All data swaps within a supergroup are confined to the groups within it. Therefore, each supergroup can independently and parallel complete its internal data swaps. On the other hand, in the twin swap case, there will always be two groups within a supergroup simultaneously computing the butterfly at the same position. For instance, groups 0 and 4 are such groups in this example, named twin groups. In the twin swap, one of the two indices

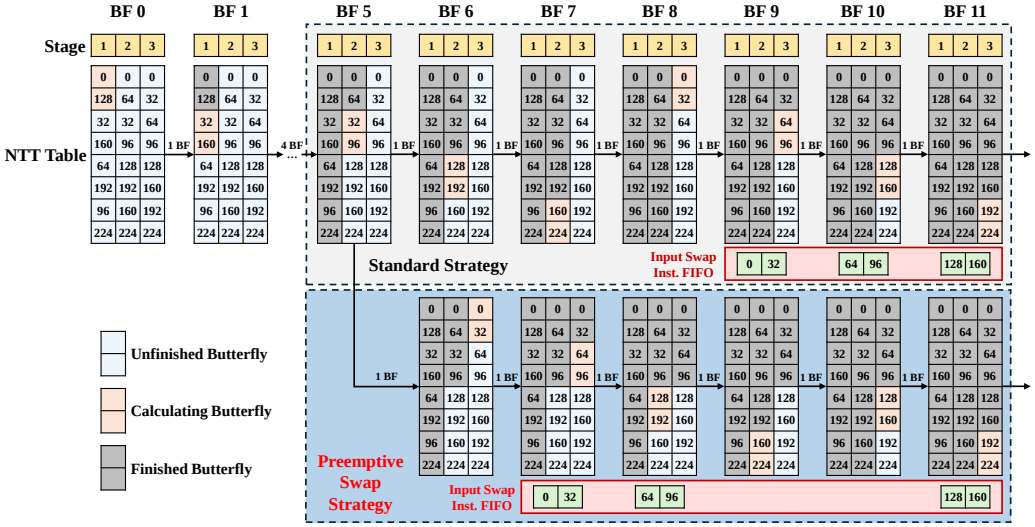


Fig. 9. The detailed timings of the butterfly computation sequence and the input of data swaps into the swap instruction FIFO with the standard strategy and **preemptive swap strategy** for group 0 in phase 1 under the configuration [3,3,2].

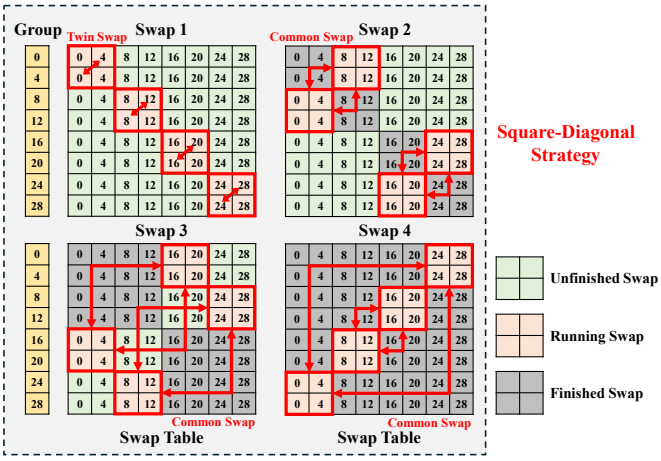


Fig. 10. The swap sequence of the butterflies in phase 1 under the configuration [3,3,2], described by the swap table.

involved in the butterfly computation within each group will always remain in the same group in the next phase. As a result, the group only needs to perform one data exchange with its twin. In the common swap case, neither of the two indices from the butterfly computation remains in the same group; instead, they are distributed across another pair of twin groups. Meanwhile, the twin of the group will also have a pair of indices at the same position that fall into this pair of twin groups. Therefore, on average, each group needs to perform two data swaps.

When $p'_s < p_s$, due to the reshaping process, the final swap in the CT NTT becomes an SD swap. SD swaps can be categorized into two cases: no swap and common swap. The [3,3,2] configuration's data swap 2 (SD) is illustrated in Fig. 8. In this scenario, the two indices involved in a group's butterfly computation correspond to coefficients that, after computation, are transferred to the same group. No swap occurs when the new group is the same as the original. In a common swap case, where the new group differs from the original, the new group that receives the coefficients also conducts a butterfly computation involving coefficients that need to be transferred back to this group, achieving synchronized butterfly computation and data swapping. Under these conditions, each group, on average, performs two data swaps.

C. Phase-Level Optimization

Sec. V-B introduces the **synchronous strategy**, which enables data transfer to occur immediately after each butterfly computation at the last stage of each phase, thereby reducing the additional time incurred by data swaps. At the phase level, the standard strategy involves each group completing all stages sequentially, with data transfer initiated only after the last stage using the method outlined in Sec. V-B. However, due to the characteristics of the NTT circuit, completing a butterfly in the final stage does not require all previous stages to be completed. Instead, it only requires specific butterflies to be computed based on a geometric series with a ratio of 0.5. To initiate data swaps as early as possible, we adopt this method, referred to as the **preemptive swap strategy**. For the [3,3,2] configuration, Fig. 9 illustrates the detailed timing of the butterfly computation sequence and the input of data swaps into the swap instruction FIFO for group 0 in phase 1, where BF denotes butterfly, and the NTT table represents the arrangement of a group's indices across all the stages of a 2^{p_s}-point NTT. As shown in the figure, the start of the data swap is advanced by 2 butterflies compared to the standard strategy. For a 2^{p_s}-point NTT where $p_s \geq 3$, the data swap can be advanced by $(p_s - 3)2^{p_s-1} + 2$ butterflies.

To implement the NTT computation order shown in Fig. 9, combined with the **synchronous strategy**, the final stage computation and swapping need performed in a sequence similar to that shown in Fig. 10. In this figure, the rows are rearranged in ascending order, and it can be seen that the butterfly circuits computed by each twin form a 2×2 square matrix, with these matrices positioned as a block diagonal matrix (e.g., the 2×2 submatrices at the top left and bottom right of a 4×4 matrix), and each running 2×2 matrix is always positioned on the diagonal of a larger matrix, which motivates the naming of the approach as the **square-diagonal strategy**. By performing swaps following this strategy, it ensures that each swap operation involves the same type of swap. For instance, in the figure, Swap 1 has only Twin Swaps, while

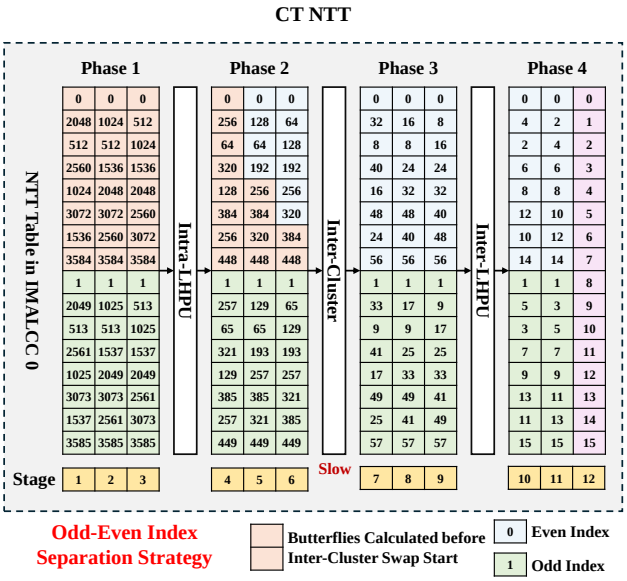


Fig. 11. The data flow of IMALCC 0 during CT NTT in MCHEAS under the configuration [3,3,3,3].

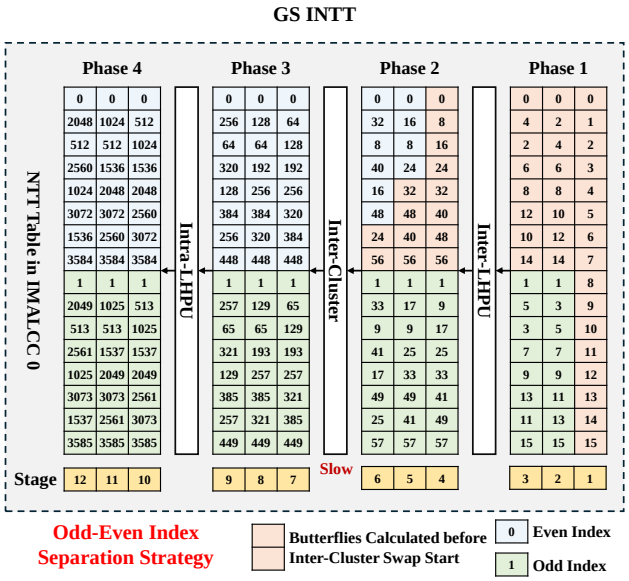


Fig. 12. The data flow of IMALCC 0 during GS INTT in MCHEAS under the configuration [3,3,3,3].

Swaps 2, 3, and 4 have Common Swaps. Additionally, this strategy guarantees that each group pair will always have one running swap during every swap operation, which maximizes the parallelism of swaps when bandwidth permits and reduces the latency. On the other hand, since special cases such as Twin Swap or No Swap result in fewer swaps, the swap operations are typically executed in reverse order (i.e., Swap 4, 3, 2, 1 in the figure). This preemptive approach ensures that swaps can begin as early as possible, further optimizing performance.

D. Architecture-Level Optimization

Finally, each group needs to be mapped to MCHEAS. When multiple clusters are employed, a large-parameter NTT is divided into 4 phases. Considering that the computation orders of the forward CT NTT and the inverse GS INTT are opposite and symmetric, to facilitate earlier inter-cluster data exchange in both 2 NTTs, the inter-cluster data swap should be positioned as close to the middle of the entire computation process as possible. On the other hand, since each IMALCC stores 2 groups and only the last stage of the CT NTT (the first stage of the GS INTT) requires butterfly computations between even and odd indices, while all other stages involve even-to-even and odd-to-odd butterflies, each IMALCC can initially store one group with even indices and one with odd indices when inputting the CT NTT coefficients. The computation and transmission can then proceed with even indices first, followed by odd indices, thereby improving the utilization of the computing units, named the **odd-even index separation strategy**. According to this strategy, the initial input state of MCHEAS for the CT NTT is set such that the $(i_I)^{th}$ IMALCC stores the $(2i_I)^{th}$ and $(2i_I+1)^{th}$ groups. The $(i_I)^{th}$ IMALCC is placed in the $(i_C)^{th}$ cluster and the $(i_L)^{th}$ LHPU in this cluster, which is determined according to Algorithm 2. According to this algorithm, the initial storage locations of the coefficients within the architecture can be determined. In the subsequent CT NTT, the three kinds of data swaps are performed in sequence as follows: intra-LHPU swaps, inter-cluster swaps (with some additional inter-LHPU swaps when $p'_s < p_s$), and inter-LHPU swaps.

Algorithm 2: IMALCC placement algorithm for initial CT NTT input

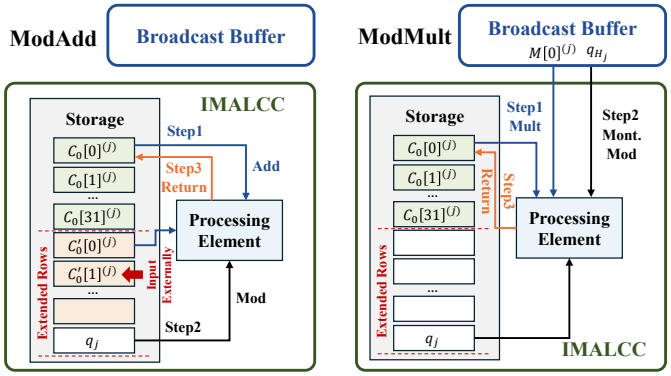
Input: n_C, n_L
Input: An IMALCC index i_I
Output: The LHPU index i_L and the cluster index i_C
1: $i_L \leftarrow i_I \bmod n_L$
2: $i_C \leftarrow \lfloor (i_I \bmod (n_C n_L))/n_L \rfloor$

For example, the data flow of IMALCC 0 during CT NTT and GS INTT on MCHEAS under the configuration [3,3,3,3] is shown in Fig. 11 and 12. Although the computation and transmission steps of GS INTT are not entirely the reverse of CT NTT, the underlying concept remains the same. As shown in these figures, taking CT NTT as an example, the computation steps include the following:

- **Step 1:** Compute all even-index groups in phase 1 and complete the data swap rapidly using the intra-LHPU NoC, as the first stage of phase 2 requires all even indices to have completed the phase 1 butterfly computation.
- **Step 2:** Process the data for phase 2 and perform the inter-cluster swap (even) according to the **synchronous**, **preemptive swap**, and **square-diagonal strategy**.
- **Step 3:** Repeat Steps 1 and 2 for the odd-index groups.
- **Step 4:** Once the inter-cluster swap (even) is completed, perform the full computation of phase 3, the inter-LHPU swap, and phase 4 (excluding the last stage) for even.
- **Step 5:** Once the inter-cluster swap (odd) is completed, process the odd-index groups through phase 3 and the inter-LHPU swap, and complete the last stage together with the even-index groups.

VI. Mapping

As mentioned in Sec. II-C, most existing RNS-CKKS HE operations can be decomposed into basic operators such as modular addition/subtraction, modular multiplication, NTT, and automorphism. The following discussion details how these basic operators and typical operations for different dimensions are implemented in MCHEAS.



(a) (b)

Fig. 13. The detailed steps of (a) ModAdd when performing Add(ct, ct'); (b) ModMult when performing PMult(ct, pt), both in an IMALCC.

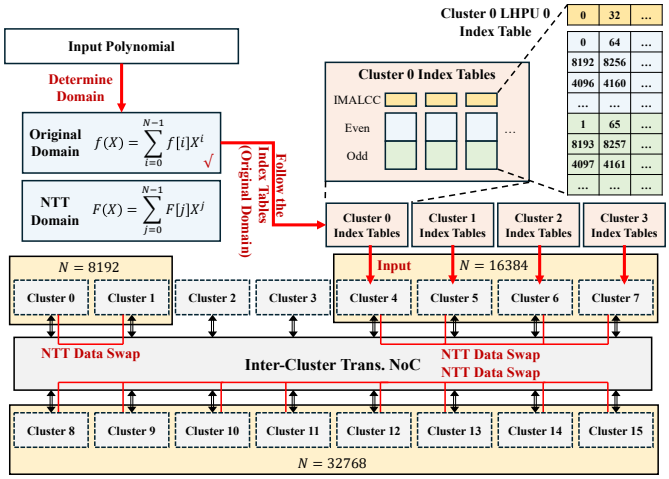


Fig. 14. Accelerating operators of different dimensions N using a 16-cluster MCHEAS supporting up to 65536 points, and the process of sorting and inputting a polynomial in the original domain ($N = 16384$) into clusters.

A. Basic Operators Implementation

There are four basic operators: ModAdd, ModMult, NTT and Automorphism. The implementation of NTT has been described in detail in Sec. V and will not be repeated.

- **ModAdd:** ModAdd refers to performing modular addition or subtraction on two coefficients. For RNS-CKKS plaintexts and ciphertexts, the Add performs modular addition or subtraction on coefficients with the same index in polynomials that are in the same domain (original or NTT). Meanwhile, each row of IMALCCs in the architecture can directly perform modular addition on two internally stored data with the help of the processing element in them. Taking the Add(ct, ct') operation with the configuration $Config = (N = 65536, n_C = 16, n_L = 8, n_{row} = 8, n_{col} = 16)$ as an example, where CT $= (C_0, C_1) =$ NTT(ct), the detailed steps of a single modular addition are shown in Fig. 13 (a). The figure indicates that during ModAdd, IMALCCs have addends stored internally that share the same index as newly input addends. Therefore, when a polynomial is input from external, the indices of the input coefficients need to align with the indices stored in IMALCCs.
- **ModMult:** ModMult is similar to ModAdd as both involve modular operations on two coefficients. ModMult typically occurs in two scenarios: when a number is multiplied by a polynomial, or when two polynomials in the NTT domain are multiplied element-wise. For the latter, it involves performing modular multiplication on coefficients with the same index. Taking PMult(ct, pt) as an example, the detailed steps of a single ModMult are shown in Fig. 13 (b), where $M =$ NTT(pt) and NTT-friendly Montgomery reduction [51] is used.
- **Automorphism:** The automorphism is primarily composed of a sign transformation and permutation. The sign transformation is realized using the operator ModAdd, while the permutation is managed by the intra-cluster and inter-cluster transmission NoCs. In our design, a permutation table is generated based on the parameters of an automorphism, and the inter-cluster transmission NoC accurately transfers data between compute clusters down to the specific cluster level. After the cluster-level transfer is completed, the intra-cluster NoC will handle the arrangement of data within the clusters.

B. Polynomial Mapping for Different-Dimension Evaluation

Different applications with varying dimensions N are used for different levels of computational demands in HE evaluation. A single cluster in MCHEAS supports an NTT of $N_C = 2^{(p_s+1)} \times n_{col} \times n_L$ points and can store an equivalent number of coefficients regardless of n_{row}. For polynomials of different N, the required number of clusters is first calculated as N/N_C, followed by the calculation of the number of polynomials the architecture can process in parallel for different moduli, given by $n_C/(N/N_C)$. For instance, a 16-cluster MCHEAS that supports $N = 65536$ can also support applications with N ranging from 4096 to 32768 by using fewer clusters in parallel. When $N = 16384$, four clusters can be grouped together to perform a single HE operator as shown in Fig. 14. Multiple groups of four clusters can then operate in parallel to handle data when higher levels are required. Leveraging multiple MCHEAS instances to jointly accelerate the application is also viable for extremely large levels.

To allow the input polynomials to benefit from the proposed NTT optimization strategies, they need to be arranged in a specific order. As previously mentioned, for operators other than NTT, coefficients used in ModAdd and ModMult must have the same indices, and automorphism does not change the input domain. Therefore, for the input polynomials, maintaining the input order for NTT/INTT as described in Sec. V ensures efficient data swaps in subsequent NTT/INTT operations using the proposed strategies. Specifically, the domain (original or NTT) must first be identified, and the polynomial should be input to the appropriate cluster based on its domain. If in the original domain, the polynomials can be input to clusters according to the cluster index tables generated by Algorithm 2, allowing efficient data swaps in future NTTs. If in the NTT domain, the input position should be adjusted to prepare for an efficient INTT. An example of inputting a polynomial with $N = 16384$ in the original domain is shown in Fig. 14. Thus, the reconfigurability of MCHEAS offers significant flexibility. For smaller N, multiple clusters can be grouped to run polynomials with different moduli in parallel. Additionally, FHE typically decomposes moduli via RNS into smaller polynomial components, allowing the system to facilitate parallel processing of multiple smaller moduli, thereby enhancing the overall utilization of MCHEAS.

VII. Experimental Results

A. Experiment Setup

For testing the performance of MCHEAS, we implement a 38-bit design in RTL, synthesize it in 28nm process technology, and evaluate it with the configuration $Config = (N = 65536, n_C = 16, n_L = 8, n_{row} = 8, n_{col} = 16)$ at a frequency of 1GHz. The area is approximately 29.57mm^2, and the power consumption at 1.24V is around 3.23W. To evaluate the optimization effectiveness of the proposed strategies under different inter-cluster data transfer bandwidths, we measure the performance of the system at four bandwidth levels: 8.86, 17.70, 26.54, and 35.39 GB/s, denoted as BW 1, 2, 3, and 4, respectively. These bandwidth levels correspond to the number of data pins per cluster. For example, when each cluster has 38 pins, the transmission bandwidth reaches 8.86 GB/s.

We first test the performance of the unoptimized and optimized MCHEAS instances for computing NTT under different BWs with $N = 65536$. Next, we evaluate the throughput of "clusters collaborating on a polynomial" and "each cluster processing a polynomial independently" in MCHEAS for processing polynomials at different levels, and test the performance for processing with varying numbers of clusters activated in MCHEAS under BW3. By varying the number of clusters and the number of LHPUs within each of them, we measure the number of data swaps for various configurations and their NTT performance. Subsequently, we evaluate the performance of NTT at various dimensions N using different numbers of clusters under BW 1. Furthermore, we test RNS-CKKS operations involving NTT with parameters ($N = 65536, L = 44$), including Rescale, Rotation, KS, and CMult [18]. We also evaluate some RNS-CKKS applications which are conducted on four benchmarks: fully-packed bootstrapping [53], LR [54], LSTM and ResNet-20 [46].

- **Fully-packed bootstrapping** transforms a noisy ciphertext at level $L = 3$ into a lower-noise ciphertext at level $L = 57$, using an advanced and commonly used RNS-CKKS bootstrapping algorithm [53].
- **LR** uses the CKKS-based HELR algorithm [54]. The training begins at the level $L = 38$ and involves 10 iterations with two bootstrapping operations.
- **LSTM** is a Long-Term Short-Term (LSTM) benchmark in natural language processing [46], which iteratively computes $h_{i+1} = \sigma(W_0 h_i + W_1 x_i)$, where σ is a non-linear activation function estimated by a cubic polynomial, and W_0 and W_1 are two 128×128 weight matrices. The benchmark involves a high level of multiplicative complexity, requiring 50 bootstrapping operations for each inference.
- **ResNet-20** is an image inference task using the ResNet-20 model, implemented with FHE [46].

B. NTT Evaluation

We evaluate performance based on the number of operations performed per second (Op/s). The NTT performance comparison under different inter-cluster data transfer bandwidths is shown in Table I. From the table, the optimized system achieves higher utilization compared to the unoptimized version for all given bandwidths, with utilization improvements ranging from 10.30% to 33.97%. The peak utilization of the computing units can reach up to 99.62%. The optimization reduces swap latency by 34.25% to 99.27%, resulting in a speedup of 26.40% to 51.75% for the unoptimized one. The speedup is positively correlated with the inter-cluster data transfer bandwidth at lower bandwidths because, although using the proposed strategies to optimize swaps cannot completely eliminate the latency it introduces, the optimized throughput increases at a higher rate as the bandwidth increases. Thus, the speedup compared to the unoptimized

TABLE I
NTT PERFORMANCE COMPARISON UNDER DIFFERENT INTER-CLUSTER DATA TRANSFER BANDWIDTHS WHEN $N = 65536$

	BW 1	BW 2	BW 3	BW 4
Freq (GHz)	1	1	1	1
Inter-Cluster Data Transfer Bandwidth (GB/s)	8.85	17.70	26.54	35.39
Throughput (Op/s) (Utilization Rate)	19799 (39.03%)	28451 (56.08%)	33303 (65.65%)	36406 (71.76%)
Optimized Throughput (Op/s) (Utilization Rate)	25026 (49.33%)	40652 (80.13%)	50536 (99.62%)	50536 (99.62%)
Utilization Improvement	**10.30%**	**24.05%**	**33.97%**	**27.85%**
Swap Latency Reduced	**34.25%**	**68.34%**	**99.27%**	**99.02%**
Speedup	**26.40%**	**42.88%**	**51.75%**	**38.81%**

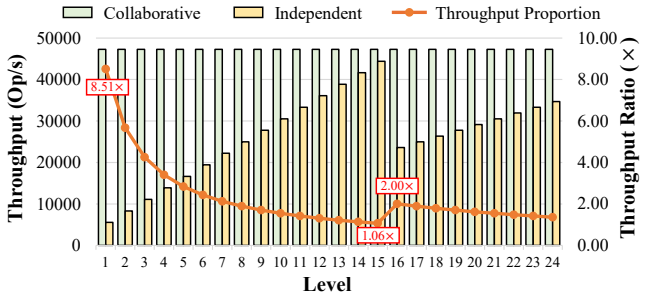


Fig. 15. The throughput of "clusters collaborating on a polynomial" and "each cluster processing a polynomial independently" in MCHEAS for processing polynomials at different levels, along with the throughput ratio of collaborative processing to independent processing under BW3.

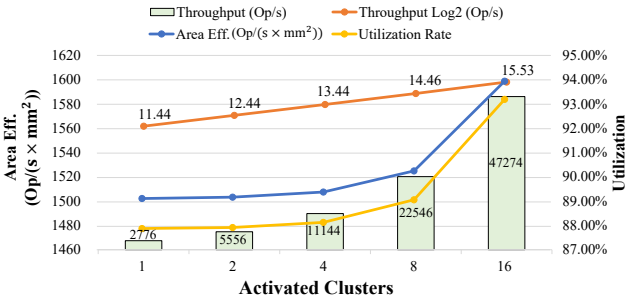


Fig. 16. The throughput, area efficiency, and utilization for processing 65536-point NTTs with varying numbers of clusters activated under BW3.

TABLE II
NUMBER OF DATA SWAPS AND MCHEAS PERFORMANCE ACROSS DIFFERENT PARAMETER SETTINGS WHEN $N = 65536$

	4096×16	8192×8	16384×4	32768×2
IMALCCs in 1 LHPU	128	128	128	128
LHPUs in 1 Cluster	8	16	32	64
Clusters in 1 MCHEAS	16	8	4	2
Intra-LHPU Swaps	30720	30720	30720	30720
Inter-LHPU Swaps	28672	30720	34816	43008
Inter-Cluster Swaps	30720	28672	24576	16384
Total Swaps (1 Poly.)	90112	90112	90112	90112
Throughput (BW2) (Op/s)	40652	42409	46421	50445
Min. Bandwidth for Opt. (GB/s)	26.54	22.12	22.12	13.27
Max. Speedup	51.75%	50.93%	49.64%	52.23%

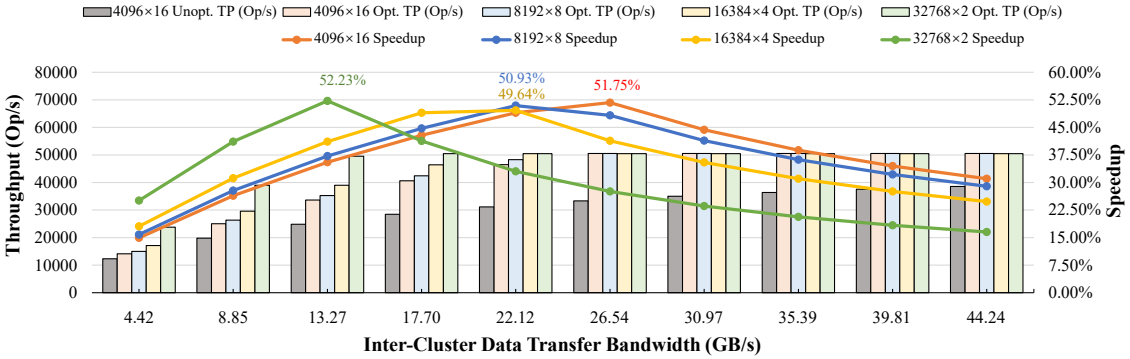


Fig. 17. The unoptimized throughput of 4096×16, the optimized throughput and speedup compared to the unoptimized one across different parameter settings under bandwidths ranging from 4.42 GB/s to 44.24 GB/s when $N = 65536$.

TABLE III
NTT PERFORMANCE COMPARISON UNDER BW 1 WHEN $N = 8192, 16384, 32768, 65536$

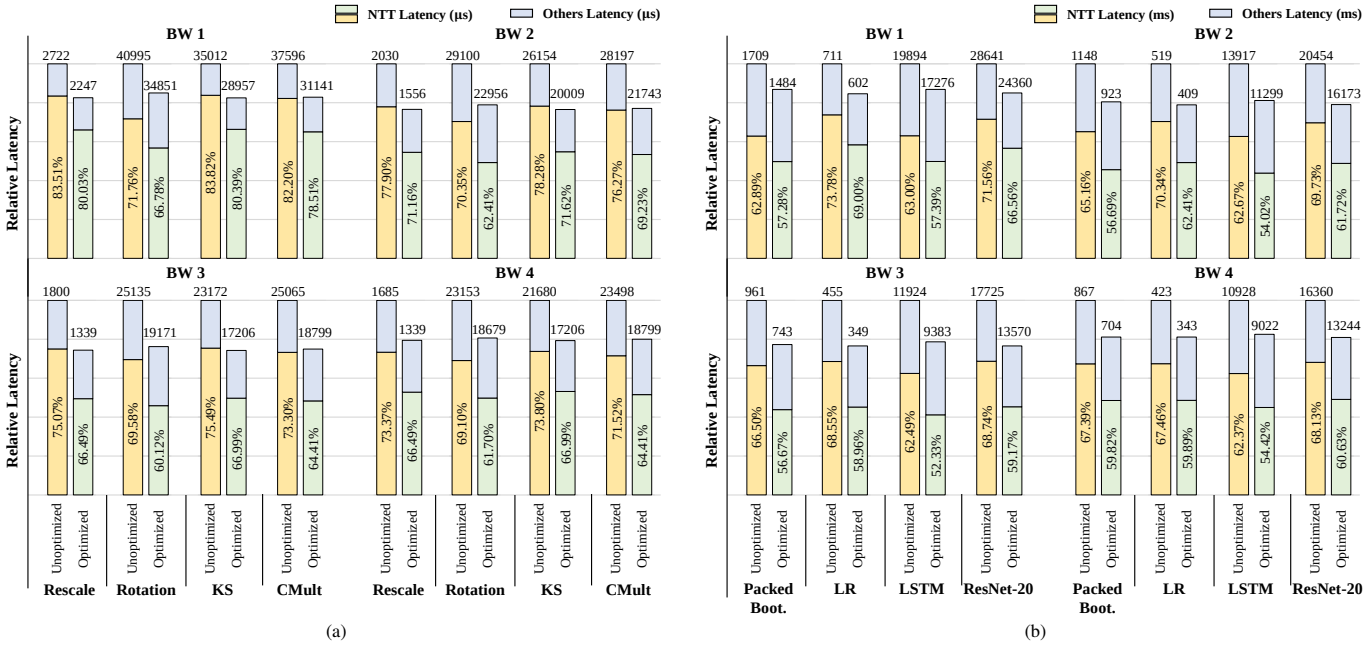
Dimension N	8192	16384	32768	65536
Cluster	2	4	8	16
Freq (GHz)	1	1	1	1
Inter-Cluster Data Transfer Bandwidth (GB/s)	8.85	8.85	8.85	8.85
Throughput (Op/s) (Utilization Rate)	55128 (88.29%)	42612 (73.50%)	30403 (56.18%)	19799 (39.03%)
Optimized Throughput (Op/s) (Utilization Rate)	62144 (99.53%)	57724 (99.56%)	43556 (80.49%)	25026 (49.33%)
Utilization Improvement	**11.24%**	**26.07%**	**24.31%**	**10.30%**
Swap Latency Reduced	**96.43%**	**98.78%**	**68.92%**	**34.25%**
Speedup	**12.73%**	**35.47%**	**43.26%**	**26.40%**

continues to rise.

To highlight the advantages of using multiple clusters to collaboratively process a polynomial, we evaluate the throughput of "each cluster collaborating on a polynomial" versus "each cluster processing a polynomial independently" in MCHEAS at different levels, along with the throughput ratio of collaborative processing to independent processing under BW3, as shown in Fig. 15, where the initial coefficient loading time is also included in the measurements. From the figure, it can be seen that as the level increases, the throughput for independent processing increases linearly. However, after every 16 levels, it decreases due to incomplete utilization of all 16 clusters when the level is not a multiple of 16. Since the level changes during the CKKS process, collaborative processing, which maintains consistent performance across levels, offers greater advantages. Moreover, in independent processing, each cluster can only handle 4096 numbers, so for larger N, clusters must repeatedly load 65536 coefficients from the external buffer, leading to additional transmission delays. Even when all 16 clusters are fully utilized, collaborative throughput is 1.06× that of independent processing, meaning the throughput of the latter can at most reach 93.96% of the former. Furthermore, we evaluate the throughput, area efficiency, and utilization for processing 65536-point NTTs with varying numbers of clusters activated in MCHEAS. As shown in Fig. 16 where area efficiency refers to the area relative to the activated clusters, it can be seen that the area efficiency and utilization rate are lower when the number of activated clusters is fewer than 16 due to the need to load the remaining coefficients. Additionally, the figure shows that the throughput increases almost linearly with the number of activated clusters, as computational resources grow linearly. Both area efficiency and utilization rate also increase, as a complete NTT computation can be split into two 256-point computations, with an inter-cluster data swap in between which is realized by data loading for the cases when the number of activated clusters is less than 16. Before the inter-cluster data swap begins, in the 256-point NTT, part of the coefficient data in the activated clusters can be optimized to perform data swaps instead of loading other coefficients by our proposed strategies. The more activated clusters there are, the more optimization can be achieved.

When the bandwidth is sufficiently large, the proposed strategy nearly eliminates the swap latency, causing the optimized throughput to stop increasing with further bandwidth growth. In contrast, the unoptimized throughput continues to increase with higher bandwidth, leading to a decrease in speedup as the bandwidth grows. Taking the 65536-point NTT as an example, the optimized throughput, unoptimized throughput, and speedup under bandwidths ranging from 4.42 GB/s to 44.24 GB/s are shown in Fig. 17. It can be observed that the speedup reaches its maximum at a bandwidth of approximately 26.54 GB/s, which corresponds to each cluster having 3×38 data pins. We also measure the number of data swaps for various configurations and their NTT performance which are shown in Table II and Fig. 17. As shown in the figure and table, as the number of LHPUs within each cluster increases, inter-cluster data swaps are replaced by inter-LHPU swaps, leading to a reduction in inter-cluster data transfer. As a result, the minimum bandwidth required to achieve optimal performance decreases progressively. When the bandwidth reaches the minimum bandwidth, the acceleration effect is optimal. Once the bandwidth exceeds the minimum bandwidth, the inter-cluster data swap in the optimized solution no longer causes idle computing units, and the total latency no longer decreases as the bandwidth increases. In contrast, the idle time in the unoptimized solution continues to decrease as the bandwidth increases, resulting in diminishing speedups. Obviously, the more LHPUs a cluster integrates, the better MCHEAS performs. However, increasing the number of LHPUs also enlarges the area and on-chip storage of the cluster, making fabrication more challenging, and may increase power consumption. Given these limitations, the chips produced may not be practical, so we use the 4096×16 configuration for our main results.

MCHEAS (4096×16) can also support computations of other dimensions N. Under BW 1, we evaluate the NTT com-



(a) (b)

Fig. 18. The latencies and the proportions of NTT in (a) HE operations and (b) applications.

TABLE IV
THE SPEEDUP OF HE OPERATIONS AND APPLICATIONS UNDER DIFFERENT INTER-CLUSTER DATA TRANSFER BANDWIDTH

	BW 1	BW 2	BW 3	BW 4
Freq (GHz)	1	1	1	1
Inter-Cluster Data Transfer Bandwidth (GB/s)	8.85	17.70	26.54	35.39
Rescale	21.13%	30.51%	34.41%	25.81%
Rotation	17.63%	26.76%	31.11%	23.95%
KS	21.22%	30.71%	34.67%	26.00%
CMult	20.73%	29.69%	33.33%	25.00%
Packed Boot.	15.12%	24.31%	29.33%	23.22%
LR	18.22%	26.76%	30.51%	23.25%
LSTM	15.15%	23.17%	27.08%	21.12%
ResNet-20	17.57%	26.47%	30.62%	23.53%

TABLE V
THE UTILIZATION IMPROVEMENT OF HE APPLICATIONS UNDER DIFFERENT INTER-CLUSTER DATA TRANSFER BANDWIDTH

		BW 1	BW 2	BW 3	BW 4
Packed Boot.	Unopt. Utilization	34.24%	50.98%	60.91%	67.47%
	Opt. Utilization	39.42%	63.38%	78.77%	83.14%
	Improvement	**5.18%**	**12.39%**	**17.86%**	**15.66%**
LR	Unopt. Utilization	45.86%	62.84%	71.68%	77.11%
	Opt. Utilization	54.22%	79.65%	93.55%	95.03%
	Improvement	**8.35%**	**16.82%**	**21.87%**	**17.92%**
LSTM	Unopt. Utilization	39.81%	56.91%	66.42%	72.48%
	Opt. Utilization	45.85%	70.10%	84.41%	87.79%
	Improvement	**6.03%**	**13.19%**	**17.99%**	**15.31%**
ResNet-20	Unopt. Utilization	42.72%	59.82%	69.93%	74.79%
	Opt. Utilization	50.23%	75.65%	90.17%	92.39%
	Improvement	**7.51%**	**15.83%**	**21.14%**	**17.60%**

putation performance for different N using various numbers of clusters, as shown in Table III. It can be observed that the proposed strategies accelerate NTT for all these dimensions, with speedups 12.73% to 43.26%. For $N = 8192, 16384$, BW 1 is significantly exceeding the requirement to eliminate most of the NTT swap latency, resulting in less speed improvement. In contrast, for $N = 32768, 65536$, BW 1 is insufficient to optimize most of the NTT swap latency effectively.

C. HE Operation and Application Evaluation

The latencies of HE operations and applications, as well as the proportion of time taken by NTT within them, are shown in Fig. 18. As illustrated in the figure, when using MCHEAS for HE operations and applications, NTT accounts for a significant portion of the total computation time. Therefore, accelerating NTT can substantially enhance overall computational speed. The acceleration results for various operations and applications are detailed in Table IV, and the utilization improvements of HE applications under different inter-cluster data transfer bandwidths are shown in Table V. Under different bandwidths, the proposed strategies reduce the NTT computation time by 3.43% to 9.46%, leading to speedups of 17.63% to 34.67% for HE operations and 15.12% to 30.62% for applications, and improve the computing units' utilization by 5.18% to 21.87% during application execution. It is important to note that due to the design characteristics of MCHEAS, the internal PEs execute each operator serially, and each operator performs basic logical operations, resulting in a relatively large execution cycle for each operator. Moreover, in specific HE operations, we can optimize the sequence of operators so that the data in MCHEAS is updated after multiple operators are executed. Each cluster in MCHEAS contains 438KB of on-chip memory, allowing it to fetch the next set of involved data and key material from off-chip memory while processing multiple operators. Through evaluation, we determine that for MCHEAS to pipeline, each cluster requires about 8.38GB/s of off-chip memory bandwidth. Therefore, MCHEAS needs at least 134.11 GB/s of total off-chip memory bandwidth.

Furthermore, we compare optimized MCHEAS with SOTA ASIC designs [23]–[25] in applications, as shown in Table VI, where the results for MCHEAS are obtained under an inter-cluster bandwidth of 26.54 GB/s. From the table, it can be observed that due to the smaller area of MCHEAS, the computation time is longer compared to other designs. However, the area efficiency of MCHEAS surpasses that of the compared designs in most cases, reaching up to 81.45×

TABLE VI
COMPARISON WITH SOTA ASIC DESIGNS IN APPLICATIONS

Design	F1+[a][23]	ARK [24]	BTS INS-2[b][25]	MCHEAS
Freq. (GHz)	1.00	1.00	1.20	1.00
Technology	14/12nm	7nm	7nm	28nm
Area (mm^2)	636	418.3	373.6	29.57
Area (mm^2) (7nm-scaled)	219.3	418.3	373.6	4.93
Packed Boot.				
Time (ms)	58.3	/	/	743
Area Eff. (Op/min × mm^2)	4.69 (**3.49**×)	/	/	16.38
LR				
Time (ms)	639	7.72	14.2	349
Area Eff. (Op/min × mm^2)	0.43 (**81.45**×)	18.59 (**1.88**×)	11.31 (**3.08**×)	34.87
LSTM				
Time (ms)	2573	/	/	9383
Area Eff. (Op/min × mm^2)	0.11 (**12.20**×)	/	/	1.30
ResNet-20				
Time (ms)	2693	125	1010	13570
Area Eff. (Op/min × mm^2)	0.10 (**8.83**×)	1.15 (**0.78**×)	0.16 (**5.64**×)	0.90

a. The experimental results for F1+ are derived from [26].
b. Since the BTS experimental results were obtained with $N = 2^{17}$, we scale their results based on the complexity $O(N \log N)$.

the efficiency of the other designs.

VIII. CONCLUSION

In conclusion, this paper has presented a multi-cluster HE accelerating system named MCHEAS, designed for high-efficiency HE evaluation, addressing the computational challenges posed by larger-parameter RNS-CKKS operations. By leveraging a multi-cluster design and implementing optimization strategies including synchronous, preemptive swap, square-diagonal, and odd-even index separation, significant improvements in NTT computation have been achieved. These strategies specifically target the inter-cluster data swap bottleneck, enhancing HE evaluation performance. Our experiments demonstrate that under 1 GHz and inter-cluster data transfer bandwidths of 8.85 to 35.39 GB/s, the proposed approach accelerates NTT computations by 26.40% to 51.75% and boosts computing unit utilization by 10.30% to 33.97%, with peak utilization reaching 99.62%. This results in speedups of 17.63% to 34.67% for HE operations involving NTT and 15.12% to 30.62% for applications, and improves the utilization by 5.18% to 21.87% during application execution. Furthermore, we compare MCHEAS, under an inter-cluster data transfer bandwidth of 26.54 GB/s, with SOTA designs, achieving up to 81.45× their area efficiencies in applications.

REFERENCES

[1] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. Stanford University, 2009.
[2] Z. Brakerski *et al.*, "(Leveled) Fully Homomorphic Encryption Without Bootstrapping," in *ITCS '12*, ACM, 2012.
[3] J. Fan *et al.*, *Somewhat Practical Fully Homomorphic Encryption*, 2012.
[4] J. H. Cheon *et al.*, "Homomorphic Encryption for Arithmetic of Approximate Numbers," in *Advances in Cryptology – ASIACRYPT 2017*, Springer, Cham, 2017.
[5] I. Chillotti *et al.*, "TFHE: Fast Fully Homomorphic Encryption Over the Torus," *Journal of Cryptology*, 2020.
[6] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the forty-first annual ACM STOC*, ACM, May 31, 2009.
[7] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical GapSVP," in *Advances in Cryptology – CRYPTO 2012*, Springer, 2012.
[8] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *J. ACM*, 2009.
[9] V. Lyubashevsky *et al.*, "On ideal lattices and learning with errors over rings," in *SpringerLink*, Springer, Berlin, Heidelberg, May 30, 2010.
[10] C. Gentry *et al.*, "Homomorphic evaluation of the AES circuit," in *Advances in Cryptology – CRYPTO 2012*, Springer, 2012.
[11] I. Chillotti *et al.*, "CONCRETE: Concrete operates oN ciphertexts rapidly by extending TfhE," presented at the WAHC 2020, Dec. 15, 2020.
[12] J. Ha *et al.*, "Rubato: Noisy ciphers for approximate homomorphic encryption," in *EUROCRYPT 2022*, Springer, 2022.
[13] A. Guimarães *et al.*, "MOSFHET: Optimized software for FHE over the torus."
[14] S. Bian *et al.*, "APAS: Application-specific accelerators for RLWE-based homomorphic linear transformations," *IEEE TIFS*, 2021.
[15] B. Reagen *et al.*, "Cheetah: Optimizing and accelerating homomorphic encryption for private inference," in *HPCA 2021*, 2021.
[16] R. Banno *et al.*, "Oblivious online monitoring for safety LTL specification via fully homomorphic encryption," in *CAV*, Springer, 2022.
[17] K. Cong *et al.*, "Sortinghat: Efficient private decision tree evaluation via homomorphic encryption and transciphering," in *ACM CCS 2022*, 2022.
[18] Y. Yang *et al.*, "Poseidon: Practical Homomorphic Encryption Accelerator," in *HPCA 2023*, 2023.
[19] A. Putra *et al.*, "Strix: An end-to-end streaming architecture with two-level ciphertext batching for fully homomorphic encryption with programmable bootstrapping," in *MICRO-56*, Oct. 2023.
[20] S. Bian *et al.*, "HE3DB: An efficient and elastic encrypted database via arithmetic-and-logic fully homomorphic encryption," in *ACM CCS 2023*, 2023.
[21] S. Bian *et al.*, "HEIR: A unified representation for cross-scheme compilation of fully homomorphic computation," in *NDSS 2024*, 2024.
[22] Z. Guan *et al.*, "Autohog: Automating homomorphic gate design for large-scale logic circuit evaluation," *IEEE TCAD*, 2024.
[23] N. Samardzic *et al.*, "F1: A Fast and Programmable Accelerator for Fully Homomorphic Encryption," in *MICRO-54*, ACM, 2021.
[24] J. Kim *et al.*, "ARK: Fully Homomorphic Encryption Accelerator with Runtime Data Generation and Inter-Operation Key Reuse," in *MICRO-55*, 2022.
[25] S. Kim *et al.*, "BTS: An accelerator for bootstrappable fully homomorphic encryption," in *ISCA '22*, ACM, 2022.
[26] N. Samardzic *et al.*, "CraterLake: A hardware accelerator for efficient unbounded computation on encrypted data," in *ISCA '49*, ACM, 2022.
[27] J. Kim *et al.*, "SHARP: A Short-Word Hierarchical Accelerator for Robust and Practical Fully Homomorphic Encryption," in *ISCA '23*, ACM, 2023.
[28] I. Yoon *et al.*, "A 55nm 50nJ/encode 13nJ/decode homomorphic encryption crypto-engine for IoT nodes to enable secure computation on encrypted data," in *IEEE CICC 2019*, Apr. 2019.
[29] S. Das *et al.*, "A 10.33 μJ/encryption homomorphic encryption engine in 28nm CMOS with 4096-degree 109-bit polynomials for resource-constrained IoT clients," in *ESSCIRC 2023*, Sep. 2023.
[30] L. Lei *et al.*, "An eDRAM-based in-situ-computing processor for homomorphic encryption evaluation on the edge," presented at the ESSERC 2024, Sep. 9, 2024.
[31] J. Wang *et al.*, "A compact and efficient hardware accelerator for RNS-CKKS en/decoding and en/decryption," *IEEE TCAS-II*, 2024.
[32] H. Lee *et al.*, "16.1 a 2.7-to-13.3μJ/boot/slot flexible RNS-CKKS processor in 28nm CMOS technology for FHE-based privacy-preserving computing," in *ISSCC 2024*, Feb. 2024.
[33] Y. Du *et al.*, "Chiplever: Towards effortless extension of chiplet-based system for FHE," in *DAC '24*, ACM, Nov. 7, 2024.
[34] S. Kim *et al.*, "CiFHER: A chiplet-based FHE accelerator with a resizable structure," in *SEED 2024*, May 2024.
[35] N. Verma *et al.*, "In-memory computing: Advances and prospects," *IEEE SSC-M*, 2019.
[36] Y. He *et al.*, "34.7 a 28nm 2.4Mb/mm^2 6.9 - 16.3TOPS/mm^2 eDRAM-LUT-based digital-computing-in-memory macro with in-memory encoding and refreshing," in *IEEE ISSCC 2024*, Feb. 2024.
[37] A. Shafiee *et al.*, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *SIGARCH Comput. Archit. News*, Jun. 18, 2016.
[38] W. Sun *et al.*, "A survey of computing-in-memory processor: From circuit to application," *IEEE OJ-SSCS*, 2024.
[39] S. Kim *et al.*, "16.5 DynaPlasia: An eDRAM in-memory-computing-based reconfigurable spatial accelerator with triple-mode cell for dynamic resource switching," in *IEEE ISSCC 2023*, Feb. 2023.
[40] J. Song *et al.*, "A calibration-free 15-level/cell eDRAM computing-in-memory macro with 3T1C current-programmed dynamic-cascoded MLC achieving 233-to-304-TOPS/W 4b MAC," in *IEEE CICC 2023*, Apr. 2023.
[41] O. Villa *et al.*, "Scaling the power wall: A path to exascale," in *SC '14*, Nov. 2014.
[42] S. Li *et al.*, "DRISA: A DRAM-based reconfigurable in-situ accelerator," in *MICRO-50*, ACM, Oct. 14, 2017.
[43] J.-L. Watson *et al.*, "Piranha: A gpu platform for secure computation," presented at the 31st USENIX Security Symposium (USENIX Security 22), 2022.
[44] NVIDIA Corporation, *Jetson orin nano developer kit carrier board specification*, version 1.2, 2024.
[45] NVIDIA Corporation, *Jetson orin nx series and jetson orin nano series product design guide*, version 1.3, 2024.
[46] J.-W. Lee *et al.*, "Privacy-preserving machine learning with fully homomorphic encryption for deep neural network," *IEEE Access*, 2022.
[47] AMD, *Amd epyc 7003 series product datasheet*, 2023.
[48] AMD, *Amd epyc 9004 series processors product datasheet*, 2023.
[49] J. H. Cheon *et al.*, "A Full RNS Variant of Approximate Homomorphic Encryption," in *Selected Areas in Cryptography – SAC 2018*, Springer, 2019.

[50] J.-P. Bossuat *et al.*, "Efficient Bootstrapping for Approximate Homomorphic Encryption with Non-sparse Keys," in *EUROCRYPT 2021*, Springer, 2021.
[51] A. C. Mert *et al.*, "Design and Implementation of a Fast and Scalable NTT-Based Polynomial Multiplier Architecture," in *DSD 2019*, 2019.
[52] J. Lin *et al.*, "INSPIRE: In-storage private information retrieval via protocol and architecture co-design," in *ISCA '22*, ACM, Jun. 11, 2022.
[53] C. V. Mouchet *et al.*, "Lattigo: A multiparty homomorphic encryption library in go," in *WAHC 2020*, 2020.
[54] H. Chen *et al.*, "Logistic regression over encrypted data from fully homomorphic encryption," *BMC Medical Genomics*, Oct. 11, 2018.

Zhenyu Guan (Member, IEEE) received the Ph.D. degree in electronic engineering from Imperial College London, the United Kingdom, in 2013. Now, he is a professor of the School of Cyber Science and Technology at Beihang University. His current research interests include image processing and high performance computing. He has published more than 45 technical papers in international journals and conference proceedings.

Yongqing Zhu (Student Member, IEEE) received his B.E. degree from Beihang University, China, in 2021, and is currently pursuing a Ph.D. degree at the School of Cyber Science and Technology, Beihang University, Beijing, China. His current research interests include homomorphic encryption and hardware accelerators.

Luchang Lei (Student Member, IEEE) received the B.E. degree in Electronic Science and Technology from Tsinghua University, Beijing, China, in 2022. He is currently working toward the Ph.D. degree in Electronic Science and Technology with the Department of Electronic Engineering, Tsinghua University, Beijing, China. His research interests include in-memory-processing architecture and circuit, and hardware acceleration for homomorphic encryption.

Hongyang Jia (Member, IEEE) received the B.E. degree from Tsinghua University, Beijing, China, in 2014, and the M.A. and Ph.D. degrees from Princeton University, Princeton, NJ, USA, in 2016 and 2021, respectively.

From 2021 to 2022, he was a Research Postdoctorate with NVIDIA, Santa Clara, CA, USA. Since August 2022, he has been with the Department of Electronic Engineering, Tsinghua University, where he is currently an Assistant Professor. His research focuses on advanced computing technologies and systems, rethinking how emerging computing fabrics can be integrated with architectures and algorithms to enhance system efficiency and performance. His primary research interests include approximate computing, mixed-signal computing, in-memory computing, energy-efficient circuits, and architectures for robotic computing and security applications.

Prof. Jia received the 2017 Outstanding Student Designer Award from ADI Inc., and the 2022 Bede Liu Best Dissertation Award in Electrical and Computer Engineering from Princeton ECE. He serves for the technical program committees for the IEEE/ACM ICCAD Conference.

Yi Chen is currently the Vice Director of Hardware R&D Center of Beijing Academy of Blockchain and Edge Computing (BABEC), Senior Researcher of National Blockchain Technology Innovation Center. His research interest includes low-power SoC design for edge computing, ASIC design for blockchain and privacy-preserving computing. He received the B.Sc. degree in microelectronics from Xiamen University, Fujian, China, and the M.Sc. degree from the Chinese Academy of Science, Beijing, China, in 2007 and 2010, respectively, and the Ph.D. degree in electrical and electronic engineering from Nanyang Technological University, Singapore, in 2014. He was a scientist in Institute of Microelectronics, A*STAR, Singapore.

Bo Zhang is currently the Director of Hardware R&D Center of Beijing Academy of Blockchain and Edge Computing (BABEC), Principle Researcher of National Blockchain Technology Innovation Center. His main areas of research interest include blockchain and privacy computing domain specific integrated circuit, micro-sensor integrated circuit, and advanced computing system. He received B.S. from Tsinghua University and Ph.D. from Pennsylvania State University. He was a Research Staff Member of IBM Research and a three-time winner of "IBM Outstanding Technical Achievement".

Changrui Ren is currently deputy director of Beijing Academy of Blockchain and Edge Computing (BABEC). His main areas of research interest include blockchain and distributed systems, business analytics and optimization, etc. He received B.S. and Ph.D. from Tsinghua University. He has published more than 70 papers, and filed more than 30 patents.

Jin Dong is currently the General Director of Beijing Academy of Blockchain and Edge Computing (BABEC), Director of National Blockchain Technology Innovation Center, Director of Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing. His main areas of research interest include blockchain, privacy computing, artificial intelligence, and integrated circuits design. He received Ph.D. from Tsinghua University and has filed more than 40 US patents. He was the Deputy Director of IBM Research and was granted the title of "IBM distinguished scientist".

Song Bian (Member, IEEE) is currently an associate professor at Beihang University. His main areas of interest include fully homomorphic encryption, privacy-preserving computing, and domain-specific hardware accelerators. He received B.S. from University of Wisconsin-Madison in 2014. He received M.S. and Ph.D. from Kyoto University, in 2017 and 2019, respectively. He was an assistant professor at Kyoto University from 2019 to 2021. He served as technical committee members/reviewers for top-tier international conferences/journals across different fields of studies, including CVPR, IEEE TIFS and IEEE TCAD. He is a member of ACM and IEEE.