An NTT/INTT Accelerator with Ultra-High Throughput and Area Efficiency for FHE

Zhaojun Lu
Hubei Key Laboratory of Distributed System Security, School of Cyber Science and Engineering, HUST
Wuhan JinYinHu Laboratory
Wuhan, Hubei, China
lzj_cse@hust.edu.cn

Weizong Yu
School of Cyber Science and Engineering, HUST
Wuhan, Hubei, China
yu_wiz@hust.edu.cn

Peng Xu*
Hubei Key Laboratory of Distributed System Security, School of Cyber Science and Engineering, HUST
Wuhan JinYinHu Laboratory
Wuhan, Hubei, China
xupeng@hust.edu.cn

Wei Wang
School of Computer Science and Technology, HUST
Wuhan, Hubei, China
viviawangwei@hust.edu.cn

Jiliang Zhang*
College of Semiconductors (College of Integrated Circuits), HNU
Changsha, Hunan, China
zhangjiliang@hnu.edu.cn

Dengguo Feng
State Key Laboratory of Cryptology
Beijing, China
fengdg@263.net

ABSTRACT

As a core arithmetic operation and security guarantee of Fully Homomorphic Encryption (FHE), Number Theoretic Transform (NTT) of a large degree is the primary source of computational and time overhead. In this paper, we propose a scalable and conflict-free memory mapping algorithm that breaks the memory bound and releases a large amount of on-chip resources. A flexible and no-stall hardware/software pipeline architecture is designed to boost the throughput of NTT/INTT of $N = 2^{16}$ to over 48,543 operations per second with area efficiency, which 4× and 10× speed up the FPGA-based (HPCA'23) and GPU-based (HPCA'23) schemes.

KEYWORDS

FHE, NTT, FPGA, Throughput

1 INTRODUCTION

In 2009, Gentry [10] introduced the Fully Homomorphic Encryption (FHE) algorithm to preserve the privacy of sensitive data, which enables users to upload the encrypted data to cloud services that it can be processed without ever being decrypted. Once the operations are completed, users can then download and decrypt the results, ensuring that their private information remains shielded and inaccessible to any third parties [3].

Nonetheless, the performance of an FHE-based application is still orders of magnitude lower than its unencrypted version [20]. Since the polynomial multiplication is the most computationally expensive operation in FHE, the Number Theoretic Transform (NTT) is utilized to reduce the complexity from $O(N^2)$ to $O(NlogN)$ [1]. FHE requires a high degree of $N(2^{15} \sim 2^{17})$ to guarantee the security. In [4], $N = 2^{16}$ with a fixed PQ (Q is the initial polynomial modulus and P is the auxiliary modulus used with Q) offers 128-bit security, which is a standard security target [3] throughout this paper.

Accelerating NTT operations is the key to improving the performance of FHE [16]. Various NTT accelerators based on Graphics Processing Unit (GPU) [12, 8], Field Programmable Gate Array (FPGA) [20, 16, 15, 2], and Application Specific Integrated Circuit (ASIC) [17] outperform the CPU-based solutions. Notably, FPGA-based and ASIC-based NTT accelerators demonstrate a performance improvement of two orders of magnitude than state-of-the-art GPU-based solutions [20]. Given the considerations of manufacturing cost and turnaround time, NTT accelerators built on FPGA platforms offer exceptional adaptability and resilience for the deployment of future FHE algorithms.

However, designing an FPGA-based high-performance NTT accelerator confronts three challenges. **Challenge 1.** The quantity of Butterfly Units (BFU) for parallel processing and their associated latency are both key factors limiting the accelerator's overall throughput. **Challenge 2.** NTT operations of a large degree ($N = 2^{16}$) are required to ensure the security of FHE. The finite on-chip sources of FPGAs are insufficient to handle the voluminous intermediate data, making external memory exchanges a necessity. **Challenge 3.** An effective memory mapping algorithm complemented by a flexible underlying architecture is essential to completely eradicate memory conflicts and prevent pipeline stalls, especially in the context of large-scale NTT operations. It represents the most complex aspect of designing an NTT accelerator of large N.

In this paper, we exploit the FPGA platform integrated with High Bandwidth Memory (HBM) [19] to accelerate NTT/INTT of $N = 2^{16}$. Our approach centers on architectural optimization and the implementation of a conflict-free memory mapping algorithm to boost the throughput of NTT/INTT:

First, a hardware/software architecture is tailored for NTT/INTT accelerator on large datasets. We develop a specialized memory access engine for each channel, capable of handling up to 32 outstanding transactions with arbitrary addresses in a single read/write burst, which is crucial in resolving access conflicts for parallel radix-16 BFUs. Additionally, A set of shallow-depth FIFOs is precisely controlled for the data flow to release on-chip memory overhead.

Second, a conflict-free memory mapping algorithm is proposed to fully leverage the substantial bandwidth provided by the 32-channel structure. Through innovative planning, the minimum Cross-Stage Interval (CSI) of each BFU is maximized, which significantly surpasses the access latency inherent in HBM. Consequently,

*Both authors share equal responsibility as corresponding authors for this research.

even under the worst case, the BFUs can function at full capacity in parallel, entirely unencumbered by memory conflicts and stalls.

Third, an efficient control flow framework is specifically engineered to implement the memory mapping algorithm with scalability and flexibility. Since the computation process is predetermined for a specified N, the data transformation operations are normalized and calculated offline during the initialization phase. At runtime, instructions are fetched from off-chip memory, a strategy that ensures the system can adaptively handle various computational demands while maintaining high performance and robustness.

2 BACKGROUND

2.1 NTT/INTT based on Radix-2 and Radix-4

As shown in Formula (1), INTT can be obtained by replacing the twiddle factor ω of NTT with its inverse element ω^{-1} followed by multiplying the scale factor N^{-1} [5]. To calculate NTT/INTT, Cooley-Tukey (CT) [7] and Gentleman-Sande (GS) [9] algorithms are two different computational approaches, serving as recursive and iterative methods, respectively.

$$A_i = NTT(a_i) = \sum_{j=0}^{N-1} a_j \omega_N^{ij} \bmod q \qquad i = 0, 1, ..., N-1$$
$$a_j = INTT(A_j) = \frac{1}{N} \sum_{i=0}^{N-1} A_i \omega_N^{-ij} \bmod q \qquad j = 0, 1, ..., N-1 \tag{1}$$

NTT algorithm of $N = 2^4$ can be implemented using radix-2 BFU Formula (2) in Figure 1(a), and using radix-4 BFU as Formula (3) in Figure 1(b).

$$X[0](k) = \Sigma_{i=0}^{\frac{N}{2}-1} a[0]_i \omega_{\frac{N}{2}}^{ki},\ where\ a[0]_i = a_{2i}$$
$$X[1](k) = \Sigma_{i=0}^{\frac{N}{2}-1} a[1]_i \omega_{\frac{N}{2}}^{ki},\ where\ a[1]_i = a_{2i+1}$$
$$x[0](k) = x[0](k + \frac{N}{2}), x[1](k) = x[1](k + \frac{N}{2})$$
$$X(k) = X[0](k) + \omega_N^k X[1](k), X(k + \frac{N}{2}) = X[0](k) - \omega_N^k X[1](k) \tag{2}$$

, where a_i is the vector of polynomial coefficients. $X[0]$ and $X[1]$ represent the sub-polynomials split according to odd and even terms, $X[k]$ and $X[k + \frac{N}{2}]$ are computed using a radix-2 BFU because they only differ by a twiddle factor ω_N^k.

Figure 1(b) shows NTT of $N = 2^4$ using radix-4 BFU. Similarly, a radix-4 BFU can merge four sub-polynomials with coefficients modulo 4 based on the twiddle factor in the matrix of Formula (3).

$$X[sub](k) = \Sigma_{i=0}^{\frac{N}{4}-1} a[sub]_i \omega_{\frac{N}{4}}^{ki},\ where\ \ a[sub]_i = a_{4i+sub},\ sub \in Z_4$$
$$X[sub](k + \frac{N}{4}) = \Sigma_{i=0}^{\frac{N}{4}-1} a[sub]_i \omega_{\frac{N}{4}}^{(k+\frac{N}{4})i} = \Sigma_{i=0}^{\frac{N}{4}-1} a[sub]_i \omega_{\frac{N}{4}}^{ki}$$
$$= X[sub](k) = X[sub](k + \frac{N}{2}) = X[sub](k + \frac{3N}{4})$$
$$\begin{bmatrix} X(k) \\ X(k+\frac{N}{4}) \\ X(k+\frac{N}{2}) \\ X(k+\frac{3N}{4}) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & w_4^1 & -1 & -w_4^1 \\ 1 & -1 & 1 & -1 \\ 1 & -w_4^1 & -1 & w_4^1 \end{bmatrix} \times \begin{bmatrix} w_N^0 X_0(k) \\ w_N^k X_1(k) \\ w_N^{2k} X_2(k) \\ w_N^{3k} X_3(k) \end{bmatrix} \tag{3}$$

Figure 1 illustrates that NTT shares common characteristics with other divide-and-conquer algorithms, i.e. the capability for multi-way division. Eight optimized radix-4 BFUs are arranged in two layers to construct a radix-16 BFU, which is the basic operating unit for NTT of 2^{16}. In comparison to NTT using radix-2 BFUs, the memory overhead for the entire execution decreases from $Nlog_2N$ to $Nlog_{16}N$.

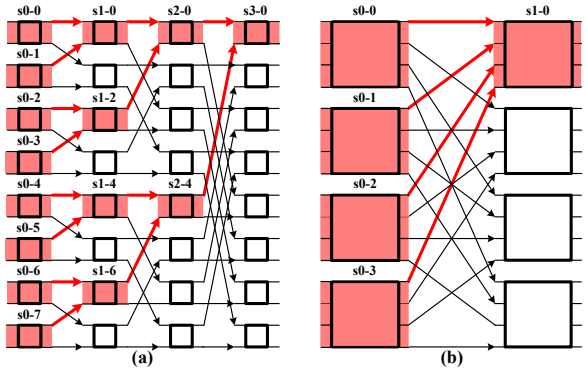

Figure 1: NTT of $N = 2^4$ using (a) radix-2 and (b) radix-4 BFU.

2.2 Memory Conflicts

Access conflicts and Read-After-Write (RAW) conflicts may occur in the data flow between BFUs and HBM, thus becoming a primary throughput bottleneck of NTT/INTT accelerators [15]. Since vast amounts of intermediate data cannot be cached in on-chip RAM, the results of parallel BFUs should be transferred into off-chip HBM, and the input data should be taken from HBM to feed BFUs. As illustrated in Figure 1(a), assume that we have four radix-2 BFUs operating simultaneously. If the eight inputs and the four twiddle factors are stored in the same memory bank, it will consume 12 clock cycles to prepare the data to feed the four BFUs. Therefore, we need separate memory banks to prevent access conflicts.

The nature of the NTT algorithm also determines that it is challenging to avoid RAW conflicts. Take Figure 1(a) as an example, inputs of s3-0 are outputs of s2-0 and s2-4. Thus, s3-0 has to wait until s2-0 and s2-4 write back the outputs before reading the inputs from the memory. To address the issue of RAW conflicts, [14] chose to insert idle cycles after each stage, which severely degraded the throughput.

3D die-stacking enables HBM to have a leap in throughput [19]. Instead of increasing the frequency, HBM uses wide buses and a high number of independent memory channels through parallelizing access to one or more stacks of DRAM chips. Hereby, every chip has two completely independent channels that are each further split into two 64-bit pseudo channels with a common command path, and every pseudo channel provides exclusive access to only its own associated memory subsection that can be directly used by an equal number of bus masters in an accelerator [11]. Although an interconnect structure is available for global addressing, the high complexity results in extra latency for data access. Xilinx Alveo U55C has two HBM stacks that provide up to 32 256-bit wide AXI3 slave ports, each has its own independent clocking with a maximum frequency of 450 MHz [18]. Theoretically, the instantaneous

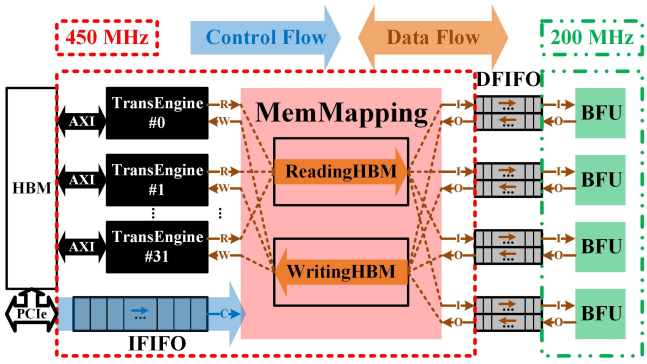

Figure 2: Overall architecture.

maximum bandwidth can reach 32 channels × 450 MHz × 256 bits = 450 GB/s. Moreover, the AXI3 protocol supports outstanding transactions, in which the master can send subsequent transaction addresses on the same channel while previous transactions are proceeding [13].

3 NTT/INTT ACCELERATOR

To address the challenges, we elaborate on the memory mapping algorithm and the underlying hardware/software architecture, thereby improving the throughput and flexibility of the proposed NTT/INTT accelerator.

3.1 Architecture

Figure 2 is the overall architecture of the proposed NTT/INTT accelerator, composed of off-chip HBM, four radix-16 BFUs, a MemMapping module, a TransEngine module for each channel, and a set of FIFOs.

Radix-16 BFU. We adopt the radix-4 multiplication in [5] to build the radix-16 BFU for NTT/INTT of $N = 2^{16}$. The entire computation process only consumes 24 clock cycles and supports uninterrupted input data due to the fully pipelined structure, thus it can output 16 points every clock cycle. By reusing the basic symmetric operators when switching between NTT and INTT modes, the area efficiency is improved.

TransEngine. Each channel is equipped with a TransEngine module to facilitate the data flows between HBM and the logic circuits. Interacting with HBM requires navigating the complex AXI3 signals, as detailed in [13], making it extremely difficult to implement the memory mapping algorithm. The TransEngine seamlessly integrates AXI3 protocols on the HBM side while providing a simplified interface for data and parameters on the MemMapping side. This design enables the execution of multiple outstanding transactions with arbitrary addresses, adhering to simple timing rules. By considerably easing the coordination of data flows, the TransEngine dramatically enhances the scalability and flexibility of the entire architectural framework, making data scheduling more efficient and straightforward.

MemMapping. The MemMapping module serves as the core module of the NTT/INTT accelerator, playing a crucial role in resolving all potential access and RAW conflicts. It comprises two parallel-operating sub-modules, ReadingHBM and WritingHBM, sharing state information. To streamline the system, we develop an innovative hardware/software co-design approach that shifts the bulk of control flow complexity to the initialization phase by transforming the fixed memory mapping algorithm into a set of pre-defined instructions comprising specific parameters and their associated timing rules. These instructions are pre-stored in off-chip memory and are dynamically fetched during runtime operations. MemMapping, following each instruction, effectively associates data with its respective address and routes it to the appropriate ports, independently of the intricacies of the memory mapping algorithm. This design ensures that when specific NTT/INTT applications vary, only the pre-stored instructions need updating, leaving the overarching architecture unchanged, thereby providing adaptability and efficiency in application deployment.

FIFO. One Instruction FIFO (IFIFO) is dedicated to dispatching instructions, while four pairs of Data FIFOs (DFIFOs) handle the intermediate data for the four radix-16 BFUs. Orchestrated by MemMapping, data is transferred rapidly, eliminating the need for extended caching or waiting periods before processing. Consequently, the DFIFOs are designed with a shallow depth to reduce the consumption of on-chip resources. The real-time status of these DFIFOs plays a key role in managing the operation of the BFUs, ensuring that the system remains reliable and efficient, even when operating near its throughput capacity.

3.2 Conflict-Free Memory Mapping Algorithm

We begin by elucidating our conflict-free memory mapping algorithm for NTT of $N = 2^8$ using four radix-4 BFUs, and then extend its application to NTT/INTT of $N = 2^{16}$ using four radix-16 BFUs. The total calculation process is structured into $log_4^{(2^8)} = 4$ stages. In each stage, the four radix-4 BFUs should operate concurrently for $\frac{2^8}{4\times4} = 16$ rounds. To gauge the likelihood of Read-After-Write (RAW) conflicts, we employ the Cross-Stage Internal (*CSI*). A larger *CSI* indicates a reduced risk of RAW conflicts. When the minimum *CSI* (CSI_{min}) surpasses the latency between writing data to the HBM and reading it back, we can assure that the NTT/INTT computation process is entirely free from RAW conflicts.

As illustrated in Figure 3, we have n radix-P BFUs ($n = 4, P = 4$). The label r_s on each radix-4 BFU denotes its operation in round $r \in [0, R]$ of stage $s \in [0, S]$ ($S = log_P N - 1, R = \frac{N}{nP} - 1$). For clarity, let us consider the BFU labeled 0_3 as an example. The input for the BFU labeled 0_3 is dependent on the outputs of BFUs labeled 0_2 and 8_2. Therefore, CSI_{min} for the BFU labeled 0_3 is calculated as $0 + 16 - 8 = 8$. Extending this to the entire computation, we observe that when NTT of $N = 2^8$ using four radix-4 BFUs is fully expanded, the CSI_{min} consistently equals 8. The core objective of our conflict-free memory mapping algorithm is to maximize CSI_{min} for NTT/INTT of $N = 2^{16}$ using four radix-16 BFUs, in which $r \in [0, 1023]$ and $s \in [0, 3]$.

Before explaining the conflict-free memory mapping algorithm for NTT/INTT of $N = 2^{16}$, we define the BitReverse function *BR(x, l)* to reverse the lower l bits of a number x.

$$BR(x, l) = \sum_{i=0}^{l-1} b_{l-i-1} 2^i + \sum_{i=l}^{L-1} b_i 2^i, \quad x = \sum_{i=0}^{L-1} b_i 2^i \tag{4}$$

Algorithm 1 takes the three parameters as inputs: the point of NTT/INTT N, the radix of BFUs P, and the number of BFUs n.

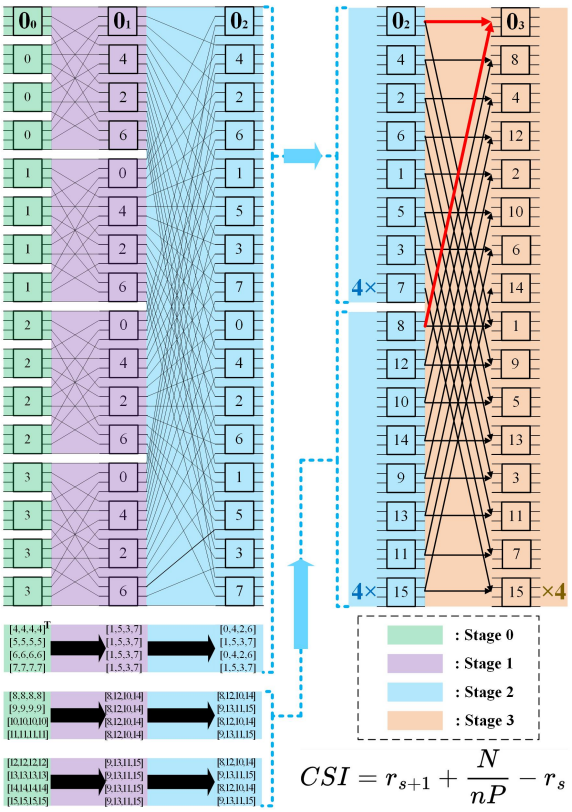

Figure 3: Conflict-free memory mapping algorithm for NTT of $N = 2^8$ using four radix-4 BFUs.

Algorithm 1 Conflict-Free Memory Mapping

Input: $N, P, n = 2^k$
Result: An array $round[S][N]$
1: **function** GETR(s, loc)
2: **if** $s < 0$ **then return endif**
3: $bc \leftarrow P^{(S-s)}$, $ec \leftarrow \frac{N}{bc}$, $l \leftarrow loc \cdot ec$, $r \leftarrow (loc+1) \cdot ec$
4: $bgn \leftarrow (loc < \frac{bc}{2})\ ?\ loc : loc - \frac{bc}{2} + \frac{N}{2P}$
5: $inc \leftarrow \lfloor \frac{bc+1}{2} \rfloor$
6: **for** $i \leftarrow 0$ **to** $r - l - 1$ **do**
7: $round[s][l+i] \leftarrow \text{BR}(i \bmod \frac{r-l}{P}, s \cdot \log_2(P) \cdot inc) + bgn$
8: **end for**
9: **for** $i \leftarrow 0$ **to** $P/2 - 1$ **do**
10: GETR($s-1, (loc \ll log_2 P) + i$)
11: GETR($s-1, (loc \ll log_2 P) + i + \frac{P}{2}$)
12: **end for**
13: **end function**
14: GETR($(S, 0)$)
15: $round \leftarrow \lfloor round/n \rfloor$

Within a specified stage s, it recursively calculates the round r in which each coefficient is engaged. This recursive computation is the core of the algorithm, ensuring precise and efficient processing for each round of the BFUs operation. The results are stored in

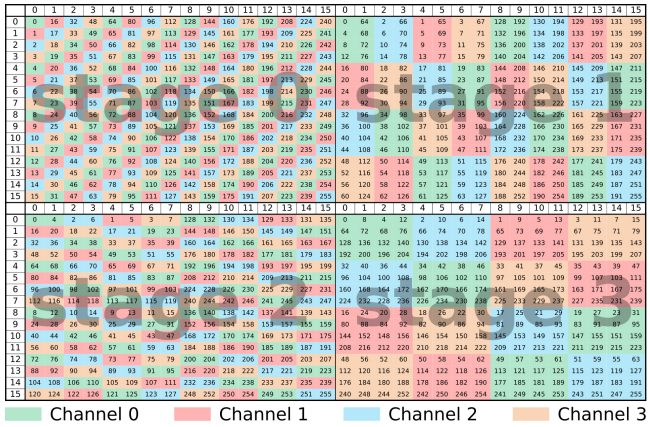

Figure 4: Balanced channel scheduling mechanism for NTT of 2^8 using four radix-4 BFUs.

a two-dimensional array $round[S][N]$. loc denotes the sequence number of the block currently undergoing recursive processing. In lines 3-5, bc signifies the count of blocks, ec indicates the number of elements within each block, l and r define the boundaries of the block in process, bgn and inc represent the initial value and increment for allocated round. Lines 6-8 assign r to each coefficient within the block. In each recursion, the first half of the block is prioritized, as described in lines 10 and 11. When the recursion is completed, each element in $round[S][N]$ is divided by n and rounded down, as in line 15.

3.3 Balanced Channel Scheduling Mechanism

Overloading any single channel with excessive read/write tasks will inadvertently create a bottleneck and hinder overall throughput. The channel scheduling mechanism associated with the memory mapping algorithm maintains a balanced data transfer across all channels, which guarantees evenly distributed access tasks for each channel during each round.

$$idx = \sum_{i=0}^{L-1} d_i (nP)^i, BS(idx) = \sum_{i=0}^{L-1} d_i \bmod nP$$
$$ID = ch(idx) = BS(idx) \bmod n_c \tag{5}$$

, where n_c is the number of HBM channels. Figure 4 illustrates the balanced channel scheduling mechanism for NTT/INTT of $N = 2^8$ using four radix-4 BFUs. The label inside each colored square signifies the index (idx) of the coefficient, while the color of the square corresponds to the channel ID, as determined by the $ch()$ function. In the diagram, the columns and rows of the white squares respectively represent the round and the sequence within that round. This layout clearly demonstrates that, in any given round, the 16 coefficients being transferred are evenly distributed across the four channels.

Consider the property of the BitSum function $BS(idx)$ for $n = 2^k$:

$$A_{(d,m,C)} = \{2^m x + C | x \in Z_d, m \geq 0\},$$
$$C = \sum_{i=0}^{L_C - 1} c_i 2^i, c_i = \begin{cases} 0, & i \in [m, m + log_2 d) \\ 0 \text{ or } 1, & i \notin [m, m + log_2 d) \end{cases} \tag{6}$$
$$BS : A_{(nP,m,C)} \longleftrightarrow Z_{nP}$$

Formula 6 can be proved using $(a+b) \bmod m = ((a \bmod m) + (b \bmod m)) \bmod m$ and $|A_{(nP,m,C)}| = |Z_{nP}|$. According to Algorithm 1, each element in $B_{(s,r)} = \{idx | round(s, idx) = r\}$ can be arranged into a nP arithmetic sequence $(b_0, b_1, ..., b_{nP-1})$ where $b_0 = min(B_{s,r})$ and the common difference is $d = b_i - b_{i-1} = 2^{L_s}$. Since $b_0 \& ((nP-1) << L_s) = 0$ (& denotes the bitwise AND operation), $B_{(s,r)} = A_{(nP,L_s,b_0)}$. Because $BS(A_{(nP,L_s,b_0)}) = Z_{nP}$, a collection of subsets $B_i = \{idx | idx \in B_{(s,r)}, BS(idx) \in [in_c, (i+1)n_c)\}$ forms a partition of set $B_{(s,r)}$, where $i \in [0, \frac{nP}{n_c})$. The following condition is satisfied, making the channel scheduling balanced:

$$\begin{aligned} &\forall i \in [0, \frac{nP}{n_c}), ID_i = \{ch(idx) | idx \in B_i\} \\ &= \{x \bmod n_c | x \in [in_c, (i+1)n_c)\} = Z_{n_c} \end{aligned} \tag{7}$$

4 IMPLEMENTATION AND EVALUATION

The proposed NTT/INTT accelerator is implemented on Xilinx Alveo U55C FPGA [18] for comprehensive on-board performance evaluation. Initially, we conduct an in-depth analysis of the CSI_{min} for the memory mapping algorithm tailored for $N = 2^{16}$ using four radix-16 BFUs. Subsequently, the Integrated Logic Analyzer (ILA) is utilized to measure the actual latency of data transfer of HBM under varying numbers of outstanding transactions. Finally, we increase the clock frequency of the radix-16 BFUs to explore the upper limits of throughput for comparison with the state-of-the-art.

4.1 CSI Analysis

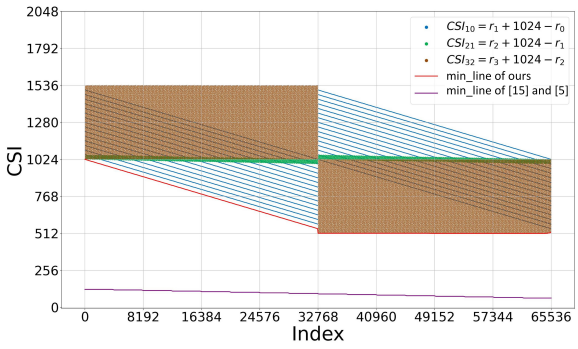

Figure 5: CSI for NTT of $N = 2^{16}$ using four radix-16 BFUs.

For the NTT/INTT accelerator of $N = 2^{16}$ using four radix-16 BFUs ($P = 16, n = 4, S = log_P N - 1 = 3, R = \frac{N}{nP} - 1 = 1023$), scatter plots drawn in Figure 5 are calculated by Algorithm 1 to represent the CSI corresponding to each idx in different stages. Blue, green, and brown represent $CSI_{10} = r_1 + 1024 - r_0$, $CSI_{21} = r_2 + 1024 - r_1$, and $CSI_{32} = r_3 + 1024 - r_2$ respectively. The red line indicates $min(CSI_{10}, CSI_{21}, CSI_{32})$ for each idx. Compared with the memory mapping algorithm proposed in [15] and [6] whose $CSI_{min} = 64$, ours achieves $CSI_{min} = 511$, which is completely conflict-free and balanced on HBM-based FPGA platforms.

4.2 Latency of RAW

The number of outstanding transactions n_{ot} varies between 1 and 32. The procedure initiates by concurrently writing data to n_{ot} different addresses across each channel. Following the successful transmission and confirmation from all 32 channels, the next phase involves reading data from these n_{ot} addresses per channel. This process is deemed complete only when all data is accurately retrieved. Figure 6(a) is the dashboard of ILA during a RAW experiment with $n_{ot} = 8$. Given the inherent variability in read/write operation latency, each RAW experiment is replicated 1,000 times. The best case and worst case of read/write latency under n_{ot} varying between 1 and 32 are shown in Figure 6 (b). It consumes up to 159 clock cycles at 450 MHz to execute a full and reliable RAW operation across all 32 channels simultaneously, which is much lower than $CSI_{min} = 511$ at 200 MHz. Therefore, the RAW conflicts are completely eliminated in the proposed NTT/INTT accelerator.

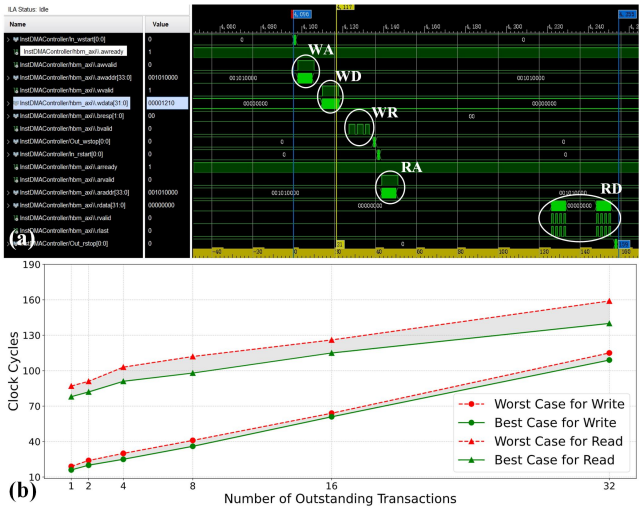

Figure 6: (a) ILA measures the latency of RAW. WA is write address, WD is write data, WR is write response, RA is read address, and RD is read data. (b) Best case and worst case of read/write latency.

4.3 Upper Limit of Throughput

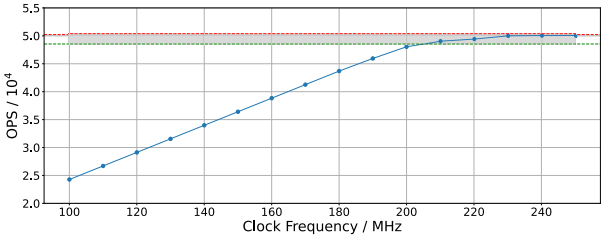

Figure 7: Throughput of the proposed NTT/INTT accelerator of $N = 2^{16}$ using four radix-16 BFUs.

As shown in Figure 7, the throughput reaches its maximum capacity when the clock frequency of the four radix-16 BFUs exceeds 200 MHz. As depicted in Figure 6(b), the latency exhibits variability within a specified range, causing the throughput to fluctuate around a mean of 48,543 Operations Per Second (OPS). This upper limit is primarily due to the inherently higher latency associated with reading from HBM compared to writing to it.

Table 1: Comparisons with GPU-based and FPGA-based schemes.

Scheme	Platform	N	Conflict-Free	OPS	Area
2021 [12]	V100	2^{16}	✗	3,619	-
2023 [8]	A100	2^{16}	✗	4,705	-
2022 [15]	Virtex-7	2^{10}	✓	-	Medium
2022 [6]	UltraScale+	2^{12}	✓	-	Medium
2020 [16]	Stratix 10	2^{16}	✗	237	High
2023 [2]	U280	2^{16}	✗	941	High
2023 [20]	U280	2^{16}	✗	12,474	High
Ours	U55C	$\mathbf{2^{16}}$	✓	**48,543**	**Low**

4.4 Comparisons

[12, 8] implement NTT/INTT on GPU platform. Even though CUDA programming assisted by assembly language can accelerate the calculation process, the lack of high-privilege and flexible memory access control makes the GPU-based schemes slower than the FPGA-based schemes.

[15, 6] cache intermediate data in on-chip BRAM to achieve conflict-free memory mapping. However, the main drawback is that they can only support NTT/INTT of $N = 2^{12}$, when N increases to 2^{16}, the conflict-free algorithm will fail.

[2, 20] employ HBM-based FPGA to implement NTT/INTT of $N = 2^{16}$, achieving higher throughput than [16]. The superiority of our proposed NTT/INTT accelerator over the state-of-the-art schemes can be attributed to three factors. First of all, lots of BRAMs and registers are consumed in [2, 20] to resolve RAW conflict, relegating HBM to a role of mere off-chip memory with high bandwidth. Our pivotal contribution lies in the conflict-free memory mapping algorithm, underpinned by the TransEngine module capable of handling multiple outstanding transactions. By allowing intermediate data to flow at high speeds, we alleviate the reliance on the on-chip memory and maximize the exploitation of HBM's bandwidth. Second, [20] instantiates 64 8-point BFUs to process 512 data in parallel. Although this achieves a theoretically high peak throughput, it is at the expense of vast on-chip resource consumption. Furthermore, the prevalent memory conflicts result in stalls to await the preparation of 512 inputs. In contrast, our design instantiates just four radix-16 BFUs with a fully pipelined structure, ensuring uninterrupted operation and improved area efficiency. Third, the data transformation across stages in [20] is complex, necessitating intricate logic and customized data structures. The complexity of logic design is greatly reduced in our design by fetching instructions from off-chip to accomplish this complex task in a software/hardware co-design pattern, thereby enhancing the overall system efficiency.

5 CONCLUSION

In accelerate the NTT/INTT of $N = 2^{16}$, we propose a versatile hardware/software co-design architecture consisting of a flexible memory access module with outstanding transactions to implement a scalable and conflict-free memory mapping algorithm. Only four radix-16 BFUs and several shallow-depth FIFOs are instantiated to achieve over 48,543 OPS, significantly outperforming the state-of-the-art FPGA-based and GPU-based schemes in terms of throughput and area efficiency.

ACKNOWLEDGMENT

The work is supported in part by the National Key Research and Development Program of China under Grant No. 2022YFB4501502, the National Natural Science Foundation of China under Grant No. 62122023, U20A20202, 62202178, and the Innovation Project of Jinyinhu Laboratory under Grant No. 2023JYH010103.

REFERENCES

[1] Ramesh C Agarwal and C Sidney Burrus. 1975. Number theoretic transforms to implement fast digital convolution. *Proceedings of the IEEE*, 63, 4, 550–560.

[2] Rashmi Agrawal, Leo de Castro, Guowei Yang, Chiraag Juvekar, Rabia Yazicigil, Anantha Chandrakasan, Vinod Vaikuntanathan, and Ajay Joshi. 2023. Fab: an fpga-based accelerator for bootstrappable fully homomorphic encryption. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 882–895.

[3] Martin Albrecht et al. 2021. Homomorphic encryption standard. *Protecting privacy through homomorphic encryption*, 31–62.

[4] Jean-Philippe Bossuat et al. 2021. Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 587–617.

[5] Xiangren Chen, Bohan Yang, Shouyi Yin, Shaojun Wei, and Leibo Liu. 2022. Cfntt: scalable radix-2/4 ntt multiplication architecture with an efficient conflict-free memory mapping scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 94–126.

[6] Xiangren Chen et al. 2022. Efficient access scheme for multi-bank based ntt architecture through conflict graph. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 91–96.

[7] James W Cooley and John W Tukey. 1965. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19, 90, 297–301.

[8] Shengyu Fan et al. 2023. Tensorfhe: achieving practical computation on encrypted data using gpgpu. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 922–934.

[9] W Morven Gentleman and Gordon Sande. 1966. Fast fourier transforms: for fun and profit. In *Proceedings of the November 7-10, 1966, fall joint computer conference*, 563–578.

[10] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, 169–178.

[11] Philipp Holzinger, Daniel Reiser, Tobias Hahn, and Marc Reichenbach. 2021. Fast hbm access with fpgas: analysis, architectures, and applications. In *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 152–159.

[12] Wonkyung Jung, Sangpyo Kim, Jung Ho Ahn, Jung Hee Cheon, and Younho Lee. 2021. Over 100x faster bootstrapping in fully homomorphic encryption through memory-centric optimization with gpus. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 114–148.

[13] Arm Limited. 2021. AMBA AXI and ACE Protocol Specification. https://documentation-service.arm.com/static/5f915b62f86e16515cdc3b1c. (2021).

[14] Ahmet Can Mert et al. 2020. A flexible and scalable ntt hardware: applications from homomorphically encrypted deep learning to post-quantum cryptography. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 346–351.

[15] Jianan Mu et al. 2022. Scalable and conflict-free ntt hardware accelerator design: methodology, proof and implementation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

[16] M Sadegh Riazi, Kim Laine, Blake Pelton, and Wei Dai. 2020. Heax: an architecture for computing on encrypted data. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 1295–1309.

[17] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Nathan Manohar, Nicholas Genise, Srinivas Devadas, Karim Eldefrawy, Chris Peikert, and Daniel Sanchez. 2022. Craterlake: a hardware accelerator for efficient unbounded computation on encrypted data. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 173–187.

[18] Xilinx. 2023. *Alveo U55C Data Center Accelerator Cards Data Sheet*. Version 1.3. https://docs.xilinx.com/r/en-US/ds978-u55c.

[19] Xilinx. 2022. *AXI High Bandwidth Controller LogiCORE IP Product Guide*. Version 1.0. https://docs.xilinx.com/r/en-US/pg276-axi-hbm.

[20] Yinghao Yang et al. 2023. Poseidon: practical homomorphic encryption accelerator. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 870–881.