Long Polynomial Modular Multiplication using Low-Complexity Number Theoretic Transform

Sin-Wei Chiu, *Student Member, IEEE*; and Keshab K. Parhi, *Fellow, IEEE*

I. Scope

This tutorial aims to establish connections between polynomial modular multiplication over a ring to circular convolution and discrete Fourier transform (DFT). The main goal is to extend the well-known theory of DFT in signal processing (SP) to other applications involving polynomials in a ring such as homomorphic encryption (HE). HE allows any third party to operate on the encrypted data without decrypting it in advance. Since most HE schemes are constructed from the ring-learning with errors (R-LWE) problem, efficient polynomial modular multiplication implementation becomes critical. Any improvement in the execution of these building blocks would have significant consequences for the global performance of HE. This lecture note describes three approaches to implementing long polynomial modular multiplication using the number theoretic transform (NTT): zero-padded convolution, without zero-padding, also referred to as negative wrapped convolution (NWC), and low-complexity NWC (LC-NWC).

II. Relevance

Homomorphic encryption (HE) schemes involve two fundamental operations: homomorphic multiplication and homomorphic addition. Most of the existing HE schemes are constructed from the R-LWE problem [1]. R-LWE-based HE schemes rely on polynomial multiplication/addition as the main building blocks, and the number of polynomial operations required grows with the multiplicative depth and width [2] of the desired function that needs to be homomorphically evaluated. Since the ciphertexts of these schemes are in the form of polynomials, the addition and multiplication operations are performed on the polynomials. While the polynomial addition is simple (coefficient-wise modular addition), the polynomial modular multiplication is complex, especially when the degree of the polynomial is large and the word length of the coefficients is long. Therefore, the most time- and memory-consuming part of an R-LWE-based scheme is the long polynomial modular multiplication. Since polynomial multiplication can be viewed as a linear convolution of the coefficients, the intuitive way to compute the multiplication of two polynomials is to use the schoolbook algorithm with the time complexity of $O(n^2)$. However, the length of the polynomials, n, of a homomorphic encryption scheme can be in the range of thousands [3]. The time complexity of performing homomorphic multiplication can be reduced to $O(n \log n)$ using the number theoretic transform (NTT).

This research was supported in parts by the Semiconductor Research Corporation under contract number 2020-HW-2998, and by the National Science Foundation under grant number CCF-2243053.

In this paper, we provide a comprehensive guide toward efficient NTT-based polynomial modular multiplication. Three NTT-based approaches are described: zero-padded convolution, negative wrapped convolution (NWC), and low-complexity NWC (LC-NWC). Examples, derivations, and comparisons are presented. This tutorial is structured to provide an easy digest of the relatively complex topic.

III. Prerequisites

This article assumes only a familiarity with discrete Fourier transform (DFT), fast Fourier transform (FFT), convolution, and basic polynomial operations.

IV. Problem Statement

Most HE schemes based on the R-LWE problem operate in the ring $R_{n,q} = \mathbb{Z}_q[x]/(x^n + 1)$ [3]. Polynomials over a ring $R_{n,q} = \mathbb{Z}_q[x]/(x^n + 1)$ are defined as:

$$p(x) = a_0 + a_1 x + \cdots + a_{n-2}x^{n-2} + a_{n-1}x^{n-1} \quad (1)$$

where n is a power-of-2 number. The coefficients are integers in $S = \{0, 1, \ldots, q-1\}$. It is important to note that n, q have a relation of $q \mod 2n \equiv 1$. This ensures that the primitive $2n$-th root of unity, ψ_{2n}, exists. The primitive $2n$-th root of unity, ψ_{2n}, is also in set S, and $\psi_{2n}^n \equiv -1 \pmod q$, $\psi_{2n}^{2n} \equiv 1 \pmod q$. Let ω_n be the primitive n-th root of unity in $\mathbb{Z}_q$, which means $\omega_n^n \equiv 1 \pmod q$ and $\omega_n = \psi_{2n}^2$.

For example, let $n = 4$, then we have a 3rd order polynomial $p(x)$. Since we need to make sure $q \mod 2n \equiv 1$, $q = 17$ is selected. Next, let's find the $2n$-th root of unity ψ_{2n}. If we try $\psi_{2n} = 2$, we need to compute powers of ψ_{2n} from 1 to $2n = 8$. We have $[2, 4, 8, 16, 32, 64, 128, 256]$. Let's compute the modulo $q = 17$ reduction of the elements in this vector, we have $[2, 4, 8, 16, 15, 13, 9, 1]$. ψ_{2n}^n should be 16 ($-1 \pmod q$) and ψ_{2n}^{2n} should be 1 after modular reduction. Hence, 2 is the $2n$-th root of unity.

Modular polynomial multiplication

In signal processing, the convolution operation is one of the fundamental operations at the center of many developments related to the Fourier transform, superposition, impulse response, etc. It is well known that the convolution of two sequences $a[n]$ and $b[n]$ can be implemented using DFT. Remember that with the DFT we can implement both circular and standard convolutions. For standard convolutions, it is necessary to zero-pad the two input sequences to ensure proper computation.

In these notes, we intend to provide a comprehensive explanation of the connection between polynomial modular

multiplication, convolution, and DFT with polynomial multiplication in $\mathbb{Z}_q[x]/(x^n+1)$.

For example, $a = [1\ 2]$ and $b = [1\ -1]$. In MATLAB, the conv(a,b) commands yields $[1\ 1\ -2]$. To calculate the same results in the transform domain, we define $\bar{a} = [1\ 2\ 0]$ and $\bar{b} = [1\ -1\ 0]$ as the zero-padded versions of a and b; implement $DFT^{-1}(DFT(a) \odot DFT(b))$, where $\odot$ denotes the pointwise multiplication. This is the convolution theorem [4] in action! Similarly, we can use Z-transform and write $A(z) = 1+2z^{-1}$ and $B(z) = 1-1z^{-1}$, and compute $A(z)B(z)$ from which we can get the convolution result.

Assume that we have two polynomials $a(x)$ and $b(x)$ over the ring $R_{n,q} = \mathbb{Z}_q[x]/(x^n+1)$, where

$$a(x) = \sum_{j=0}^{n-1} a_j x^j, \tag{2}$$

$$b(x) = \sum_{j=0}^{n-1} b_j x^j. \tag{3}$$

Remember that the coefficients of $a(x)$ and $b(x)$ have to be in the range of $[0, q-1]$. Let's assume that we want to compute the modular polynomial multiplication

$$p(x) = a(x) \times b(x) \mod (q,\ x^n+1) \tag{4}$$

It is important to point out that the operation mod (x^n+1) can be viewed as the negated mapping of conventional mod (x^n-1), i.e., the circular convolution [5]. For example, $x^n \mod x^n+1 = -1$ instead of 1; $x^{n+1} \mod x^n+1 = -x$ instead of x. In general, $x^{n+i} \mod x^n+1 = -x^i$, where i is an integer from 0 to $n-1$.

We can revisit the example of $a(x) = 1+2x$ and $b(x) = 1-x$. Computing $a(x) \times b(x) \mod (x^2-1)$ is the same as computing the circular convolution. We have

$$\begin{aligned} &a(x) \times b(x) \mod (x^2-1) \\ &= 1 + x - 2x^2 \mod (x^2-1) \\ &= -1 + x \end{aligned}$$

Computing $a(x) \times b(x) \mod (x^2+1)$ is the same as computing a negated circular convolution, commonly referred as negative wrapped convolution (NWC). We have

$$\begin{aligned} &a(x) \times b(x) \mod (x^2+1) \\ &= 1 + x - 2x^2 \mod (x^2+1) \\ &= 3 + x \end{aligned}$$

The modular polynomial multiplication can be carried out using the convolution property [4] as:

$$\begin{aligned} \hat{p}(x) = INTT_{2n}(&NTT_{2n}(zeropadding(a(x)) \odot \\ &NTT_{2n}(zeropadding(b(x))), \end{aligned} \tag{5}$$

$$p(x) = \hat{p}(x) \mod (q,\ x^n+1). \tag{6}$$

The function $zeropadding(a(x))$ converts $a(x)$ from a length-n polynomial to a length-$2n$ polynomial by padding n zeros at the end.

$$zeropadding(a(x)) = a(x) + \sum_{k=n}^{2n-1} a_k x^k,\ a_k = 0\ \forall\ k. \tag{7}$$

where the NTT [6], a transformation similar to the DFT, is carried out in a finite ring [7], where the twiddle factors are powers of an integer root of unity, i.e., $\omega_n^n \equiv 1 \pmod q$. Note that the twiddle factors in the DFT are expressed in terms of the complex exponential, $e^{-j2\pi/n}$, i.e., the n-th root of unity. The main reason that we are using NTT instead of conventional DFT is that the ciphertext in HE operates over integer arithmetic. No complex number calculations are required in NTT, unlike in DFT. Furthermore, DFT will introduce undesired additional errors in arithmetic operations due to truncation or rounding; these errors do not occur with NTT. NTT is defined as:

$$A_k = \sum_{j=0}^{n-1} a_j \omega_n^{kj} \mod q,\ k \in [0, n-1] \tag{8}$$

We can represent NTT in a matrix form:

$$\mathbf{A} = \mathbf{W}\mathbf{a} \tag{9}$$

where $\mathbf{A}$ and $\mathbf{a}$ are n-by-1 vectors, and $\mathbf{W}$ is the n-by-n NTT matrix given by:

$$\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix} \mod q \tag{10}$$

Note that $\mathbf{W}$ is a symmetric matrix. Let's assume we have $n = 4$ and $q = 17$. From the previous example, we know that $\psi_{2n} = 2$ and $\omega = \psi_{2n}^2 = 4$. For these parameters, the 4-by-4 NTT matrix is given by:

$$\mathbf{W} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 4 & 16 & 13 \\ 1 & 16 & 1 & 16 \\ 1 & 13 & 16 & 4 \end{bmatrix}$$

Take $\mathbf{a} = [1\ 2\ 3\ 4]^T$ as an example, the output $\mathbf{A}$ from Equation (8) before modular reduction will be $[10\ 109\ 100\ 91]^T$. After modular reduction, $\mathbf{A}$ will be $[10\ 7\ 15\ 6]^T$. INTT is defined as:

$$a_j = n^{-1} \sum_{k=0}^{n-1} A_k \omega_n^{-kj} \mod q,\ j \in [0, n-1] \tag{11}$$

Similar to NTT, we can also represent INTT in a matrix form:

$$\mathbf{a} = \mathbf{W^{-1}}\mathbf{A} \tag{12}$$

where $\mathbf{W^{-1}}$ is the inverse matrix of $\mathbf{W}$, and is given by:

$$\mathbf{W}^{-1} = n^{-1} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \cdots & \omega^{-(n-1)} \\ 1 & \omega^{-2} & \omega^{-4} & \cdots & \omega^{-2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \cdots & \omega^{-(n-1)(n-1)} \end{bmatrix} \mod q$$

We can again create an example INTT matrix with the same parameters as above. First, we need to find ω^{-1} and n^{-1}. We can do so by finding the inverse of ω and n, i.e., $\omega\omega^{-1} \mod q \equiv 1$ and $nn^{-1} \mod q \equiv 1$. Therefore, for $\omega = 4$ and $n = 4$, we have $\omega^{-1} = 13$ and $n^{-1} = 13$. The example INTT matrix is shown below:

$$\mathbf{W}^{-1} = 13 \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 13 & 16 & 4 \\ 1 & 16 & 1 & 16 \\ 1 & 4 & 16 & 13 \end{bmatrix} \mod q = \begin{bmatrix} 13 & 13 & 13 & 13 \\ 13 & 16 & 4 & 1 \\ 13 & 4 & 13 & 4 \\ 13 & 1 & 4 & 16 \end{bmatrix}$$

Take $\mathbf{A} = [10\ 7\ 15\ 6]^T$ from the previous example, the output $\mathbf{a}$ from Equation (12) before modular reduction will be $[494\ 308\ 377\ 293]^T$. After modular reduction, $\mathbf{a}$ will be $[1\ 2\ 3\ 4]^T$, which is the same as what we started with. Continuing from Equation (5), we can compute $\hat{p}(x)$, the $2n$-point polynomial multiplication output,

$$\hat{p}(x) = \sum_{j=0}^{2n-1} \hat{p}_j x^j, \text{ and } \hat{p}_j = \sum_{i=0}^{j} a_i b_{(j-i)}. \tag{13}$$

The desired reduced product $p(x)$ can be calculated by using two $2n$-point NTT and one $2n$-point INTT followed by a modular polynomial reduction of $(q, x^n + 1)$. Fig. 1(a) shows a block diagram of modular polynomial multiplication using this approach.

Although we can correctly obtain the modular polynomial multiplication output, appending zeros to the original input polynomials to a length of $2n$ and using an additional modular polynomial reduction block at the end are not efficient for computing the modular multiplication. Furthermore, the use of an n-point NTT instead of a $2n$ point NTT is desirable.

V. Solution

We can use a different approach that does not require the $zeropadding()$ functions and the modular polynomial multiplication block. This approach is referred to as the negative wrapped convolution (NWC) [8]. Before we delve into the concept of NWC, it is crucial to point out that the conventional circular convolution cannot be applied to solve this problem since it requires modulo $(x^n + 1)$ (negacyclic) operations instead of modulo $(x^n - 1)$ (cyclic) operations.

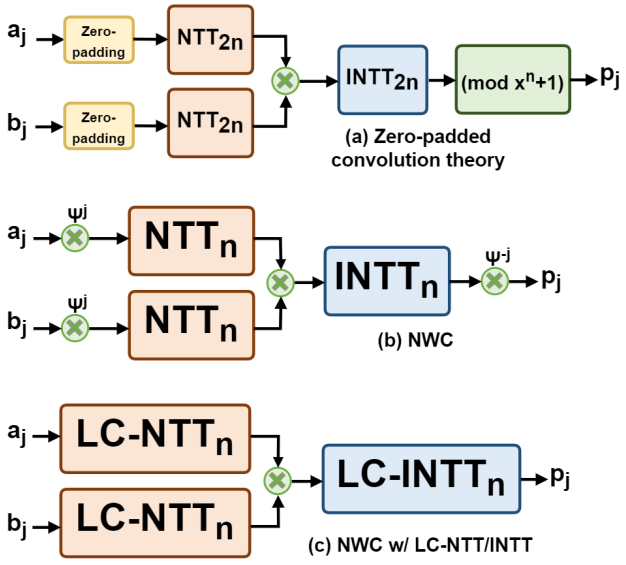

Fig. 1: Block diagrams of modular polynomial multiplication (a) Zero-padded convolution theory. (b) NWC. (c) NWC with low-complexity NTT/INTT.

Negative wrapped convolution

When computing polynomial multiplication $a(x) \times b(x) \mod (q,\ x^n + 1)$ over the ring $R_{n,q} = \mathbb{Z}_q[x]/(x^n + 1)$, NWC can be performed as:

$$\widetilde{p}(x) = INTT_n(NTT_n(\widetilde{a}(x)) \odot NTT_n(\widetilde{b}(x))), \tag{14}$$

$$p(x) = \sum_{j=0}^{n-1} \widetilde{p}_j \psi_{2n}^{-j} x^j \mod q. \tag{15}$$

where $\odot$ denotes point-wise multiplication and

$$\widetilde{a}(x) = \sum_{j=0}^{n-1} a_j \psi_{2n}^{j} x^j \mod q, \tag{16}$$

$$\widetilde{b}(x) = \sum_{j=0}^{n-1} b_j \psi_{2n}^{j} x^j \mod q. \tag{17}$$

NWC makes sure that no zero-padding is required for the operation. With NWC, the desired reduced product $p(x)$ can be calculated by using two n-point NTT operations and one n-point INTT. Although we are able to reduce the 2n-point polynomial multiplication to a n-point polynomial multiplication, there are some tradeoffs. NWC requires pre-processing before NTT and post-processing after INTT. Equations (16) and (17) describe the pre-processing step where the input is multiplied by the $2n$-th roots of unity raised to the power of j, ψ_{2n}^j, and Equation (15) represents the post-processing step where the output coefficients of INTT are multiplied by the inverse of $2n$-th roots of unity raised to the power of j, ψ_{2n}^{-j}. By combining the pre-processing and NTT, we have:

$$\widetilde{\mathbf{A}} = \mathbf{W}\mathbf{\Psi}\mathbf{a} \tag{18}$$

where $\mathbf{\Psi}$ is a n-by-n diagonal matrix, whose diagonal terms

are ψ_{2n}^j:

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & \psi_{2n}^1 & 1 & \cdots & 0 \\ 0 & 0 & \psi_{2n}^2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \psi_{2n}^{n-1} \end{bmatrix} \mod q \tag{19}$$

We can also combine the post-processing with INTT:

$$\mathbf{a} = \mathbf{\Psi^{-1}} \mathbf{W^{-1}} \widetilde{\mathbf{A}} \tag{20}$$

where $\mathbf{\Psi^{-1}}$ is also a n-by-n diagonal matrix, whose diagonal terms are ψ_{2n}^{-j}.

To prove the correctness of NWC, the coefficients of the NWC NTT outputs are denoted as:

$$\widetilde{A}_k = \sum_{i=0}^{n-1} a_i \psi_{2n}^i \omega_n^{ki} \tag{21}$$

$$\widetilde{B}_k = \sum_{j=0}^{n-1} b_j \psi_{2n}^j \omega_n^{kj} \tag{22}$$

The point-wise multiplication of two NWC NTT outputs is given by:

$$\widetilde{P}_k = \widetilde{A}_k \widetilde{B}_k = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j \psi_{2n}^{(i+j)} \omega_n^{k(i+j)} \tag{23}$$

Applying NWC INTT to $\widetilde{P}_k$, we have:

$$\begin{aligned} p_l &= n^{-1} \psi_{2n}^{-l} \left(\sum_{k=0}^{n-1} \left(\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j \psi_{2n}^{(i+j)} \omega_n^{k(i+j)} \right) \omega_n^{-lk} \right) \\ &= n^{-1} \psi_{2n}^{-l} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j \psi_{2n}^{(i+j)} \sum_{k=0}^{n-1} \omega_n^{k(i+j-l)} \end{aligned} \tag{24}$$

Since

$$\sum_{k=0}^{n-1} \omega_n^{k(i+j-l)} = \begin{cases} n & \text{if } (i+j-l) = n \text{ or } 0 \\ 0 & \text{otherwise} \end{cases}$$

Equation (24) can be expressed as:

$$\begin{aligned} p_l &= \psi_{2n}^{-l} \sum_{i=0}^{l} a_i b_{l-i} \psi_{2n}^l + \psi_{2n}^{-l} \sum_{i=l+1}^{n-1} a_i b_{n+l-i} \psi_{2n}^{l+n} \\ &= \sum_{i=0}^{l} a_i b_{l-i} + \sum_{i=l+1}^{n-1} a_i b_{n+l-i} \psi_{2n}^n \qquad (25) \\ &= \sum_{i=0}^{l} a_i b_{l-i} - \sum_{i=l+1}^{n-1} a_i b_{n+l-i} \qquad (26) \end{aligned}$$

Note that subtraction of the second term in Equation (26) implicitly carries out the polynomial modulo $(x^n + 1)$. The negative term results from the fact that $\psi_{2n}^n = -1$ in Equation (25). We can also connect Equation (25) to the circular convolution. If we remove the ψ from the equation, Equation (26) becomes:

$$\sum_{i=0}^{l} a_i b_{l-i} + \sum_{i=l+1}^{n-1} a_i b_{n+l-i} \tag{27}$$

It is easy to see that this is exactly a circular convolution. It is also called positive wrapped convolution because of the plus term.

Fig. 1(b) shows a block diagram of modular polynomial multiplication using NWC. The NWC algorithm is described in Algorithm 1.

Algorithm 1 Negative Wrapped Convolution [8]

Input: $a(x), b(x) \in R_{n,q}$
Output: $p(x) = a(x) \times b(x) \mod (x^n + 1, q)$
1: $\widetilde{a}(x) = \sum_{j=0}^{n-1} a_j \psi_{2n}^j x^j \mod q$
$\widetilde{b}(x) = \sum_{j=0}^{n-1} b_j \psi_{2n}^j x^j \mod q$
2: $\widetilde{A}(x) : \widetilde{A}_k = \sum_{j=0}^{n-1} \widetilde{a}_j \omega_n^{kj} \mod q,\ k \in [0, n-1]$
$\widetilde{B}(x) : \widetilde{B}_k = \sum_{j=0}^{n-1} \widetilde{b}_j \omega_n^{kj} \mod q,\ k \in [0, n-1]$
3: $\widetilde{P}(x) = \widetilde{A}(x) \odot \widetilde{B}(x) = \sum_{k=0}^{n-1} \widetilde{A}_k \widetilde{B}_k x^k$
4: $\tilde{p}(x) : \widetilde{p}_j = n^{-1} \sum_{k=0}^{n-1} \widetilde{P}_k \omega_n^{-kj} \mod q,\ j \in [0, n-1]$
5: $p(x) = \sum_{j=0}^{n-1} \widetilde{p}_j \psi_{2n}^{-j} x^j \mod q$

Although we can reduce the length-$2n$ polynomial multiplication to length-n by using NWC, there are still some tradeoffs. Additional weighted operations are required before NTT and after INTT. This requires a total of $2n$ additional large coefficient modular multiplications compared to classic NTT/INTT computation. Recent works [9], [10] have presented a new method to merge the weighted operations into the butterfly operations. This method is able to merge the pre-processing portion into the NTT block with low-complexity NTT and the post-processing portion into the INTT block with low-complexity INTT. This is illustrated in the block diagram shown in Fig. 1(c).

Low-complexity NTT for NWC

The low-complexity NTT merges the weighted operation before NTT in Step 2 of Algorithm 1 by changing the twiddle factors. In particular, the new NTT operation is re-represented as $\widetilde{A}_k$ and $\widetilde{A}_{k+n/2}$ by using the decimation-in-time (DIT) method [11] in FFT. This method divides the input sequence into the sequence of even and odd numbered samples. Thus, the name "decimation-in-time".

The NTT equation for NWC is described by:

$$\widetilde{A}_k = \sum_{j=0}^{n-1} a_j \psi_{2n}^j \omega_n^{kj} \mod q, \tag{28}$$

we can rewrite Equation (28) by splitting the summation into two groups: one containing the even and the other containing

odd coefficients. For $k = 0, 1, ..., n-1$:

$$\widetilde{A}_k = \sum_{j=0}^{n/2-1} a_{2j}\psi_{2n}^{2j}\omega_n^{2kj} + \sum_{j=0}^{n/2-1} a_{(2j+1)}\psi_{2n}^{2j+1}\omega_n^{k(2j+1)} \mod q$$

With the scaling property of twiddle factors, $\omega_{n/m}^{k/m} = \omega_n^k$:

$$\widetilde{A}_k = \sum_{j=0}^{n/2-1} a_{2j}\psi_n^j\omega_{n/2}^{kj} + \psi_{2n}\omega_n^k \sum_{j=0}^{n/2-1} a_{(2j+1)}\psi_n^j\omega_{n/2}^{kj} \mod q$$

Then we can group them into two parts based on the size of the index k. For indices $k > n/2-1$, we rewrite them as $k + n/2$, where $k = 0, 1, ..., n/2-1$ By applying the symmetry property of twiddle factors ($\omega_n^{k+n/2} = -\omega_n^k$) and the periodicity property of twiddle factors ($\omega_n^{k+n} = \omega_n^k$), we have:

$$\widetilde{A}_k = a_k^{(0)} + \psi_{2n}\omega_n^k a_k^{(1)} \mod q,$$
$$\widetilde{A}_{k+n/2} = a_k^{(0)} - \psi_{2n}\omega_n^k a_k^{(1)} \mod q,$$

where $k \in [0, \frac{n}{2}-1]$ and

$$a_k^{(0)} = \sum_{j=0}^{n/2-1} a_{2j}\psi_n^j\omega_{n/2}^{kj} \mod q, \quad (29)$$

$$a_k^{(1)} = \sum_{j=0}^{n/2-1} a_{(2j+1)}\psi_n^j\omega_{n/2}^{kj} \mod q. \quad (30)$$

It is easy to see that $a_k^{(0)}$ and $a_k^{(1)}$ are essentially same as Equation (28); the only difference is that they are scaled down to $n/2$ points. By recursively applying the decimation process to $a_k^{(0)}$ and $a_k^{(1)}$ to 2-point NTT, we can get the structure shown in Figure Fig. 2 (upper left). Also, Since $\omega_n = \psi_{2n}^2 \mod q$, the integers ψ_{2n} and ω_n^k can be merged to an integer, $\psi_{2n}\omega_n^k = \psi_{2n}^{(2k+1)}$. Thus,

$$\widetilde{A}_k = a_k^{(0)} + \psi_{2n}^{(2k+1)} a_k^{(1)} \mod q,$$
$$\widetilde{A}_{k+n/2} = a_k^{(0)} - \psi_{2n}^{(2k+1)} a_k^{(1)} \mod q.$$

We can further represent this architecture in a matrix form.

$$\widetilde{\mathbf{A}} = \mathbf{W}\mathbf{\Psi}\mathbf{a} = \widetilde{\mathbf{W}}\mathbf{a} \quad (31)$$

where $\widetilde{\mathbf{W}}$ is a modified version NTT matrix for NWC:

$$\widetilde{\mathbf{W}} = \begin{bmatrix} 1 & \psi & \psi^2 & \cdots & \psi^{n-1} \\ 1 & \psi\omega & \psi^2\omega^2 & \cdots & \psi^{n-1}\omega^{n-1} \\ 1 & \psi\omega^2 & \psi^2\omega^4 & \cdots & \psi^{n-1}\omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \psi\omega^{(n-1)} & \psi^2\omega^{2(n-1)} & \cdots & \psi^{n-1}\omega^{(n-1)(n-1)} \end{bmatrix}$$

Note that the $\widetilde{\mathbf{W}}$ matrix is not a symmetric matrix like $\mathbf{W}$.

Low-complexity INTT for NWC

The improved INTT algorithm merges not only the weighted operation but also the multiplication with constant n^{-1} into the butterfly operations, as presented in [9].

The low-complexity NWC operation can be described as:

$$\mathbf{p} = \mathbf{\Psi}^{-1}\mathbf{W}^{-1}\widetilde{\mathbf{P}} = \widetilde{\mathbf{W}}^{-1}\widetilde{\mathbf{P}} \quad (32)$$

and

$$\widetilde{\mathbf{W}}^{-1} = n^{-1}\begin{bmatrix} 1 & 1 & \cdots & 1 \\ \psi^{-1} & \psi^{-1}\omega^{-1} & \cdots & \psi^{-1}\omega^{-(n-1)} \\ \psi^{-2} & \psi^{-2}\omega^{-2} & \cdots & \psi^{-2}\omega^{-2(n-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \psi^{-(n-1)} & (\psi\omega)^{-(n-1)} & \cdots & (\psi\omega^{(n-1)})^{-(n-1)} \end{bmatrix} \quad (33)$$

where $\widetilde{\mathbf{W}}^{-1}$ is the inverse of $\widetilde{\mathbf{W}}$. Equation (32) can be interpreted as the transpose of the low-complexity NTT followed by n^{-1} scaling. We can obtain the low-complexity INTT structure by first transposing the low-complexity NTT structure, changing the twiddle factors to its inverse, and adding multiply by 2^{-1} at the end of every stage, which is equivalent to multiplying n^{-1} ($n^{-1} = (2^{-1})^{(\log_2 n)}$). Thus, transposing the NTT structure in Fig. 2 (upper left), replacing the twiddle factors by their inverse, and inserting 2^{-1} after every stage leads to the low-complexity INTT structure in Fig. 2 (lower left).

Although we could derive the structure based on intuition, we could still derive the low-complexity INTT based on the decimation-in-frequency (DIF) method [11] in FFT. This method divides the output sequence into the sequence of even and odd numbered samples. Thus, the name "decimation-in-frequency". The INTT equation for negative wrapped convolution is given by:

$$p_k = n^{-1}\psi_{2n}^{-k}\sum_{j=0}^{n-1}\widetilde{P}_j\omega_n^{-kj} \mod q \quad (34)$$

we can rewrite Equation (34) by splitting the items in the summation into two parts according to the size of the index of $\widetilde{P}_j$. For $k = 0, 1, ..., n-1$:

$$p_k = n^{-1}\psi_{2n}^{-k}\left(\sum_{j=0}^{n/2-1}\widetilde{P}_j\omega_n^{-kj} + \sum_{j=n/2}^{n-1}\widetilde{P}_j\omega_n^{-kj}\right) \mod q$$

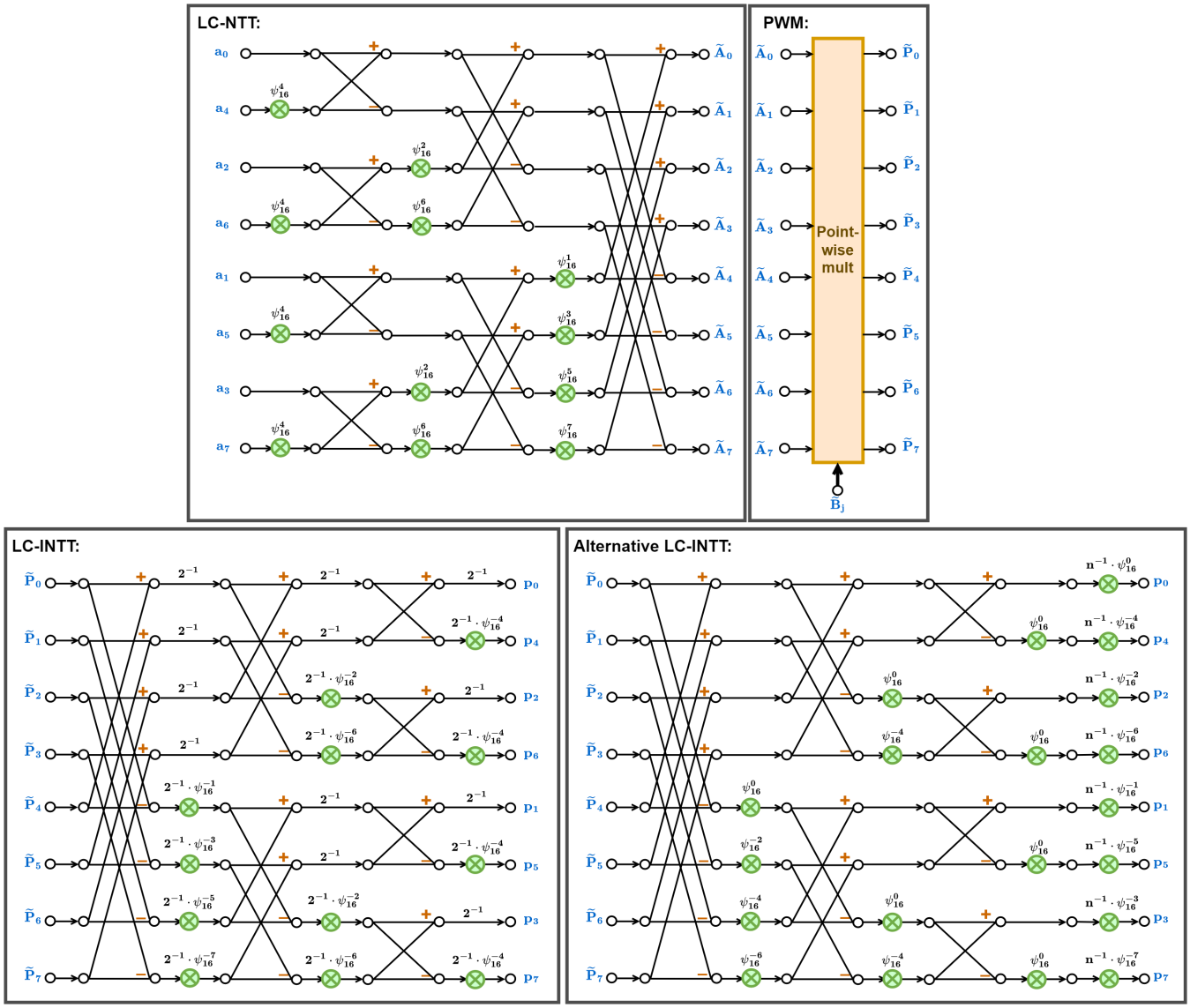

Fig. 2: The data flow graph of an 8-point low-complexity negative wrapped convolution.

Based on the symmetry property and periodicity property of twiddle factors, the index of the second sum can be changed from $[n/2, n-1]$ to $[0, n/2-1]$:

$$p_k = n^{-1}\psi_{2n}^{-k}\left[\sum_{j=0}^{n/2-1}\widetilde{P}_j\omega_n^{-kj} + \sum_{j=0}^{n/2-1}\widetilde{P}_{(j+n/2)}\omega_n^{-k(j+n/2)}\right] \mod q$$

$$= n^{-1}\psi_{2n}^{-k}\left[\sum_{j=0}^{n/2-1}\widetilde{P}_j\omega_n^{-kj} + (-1)^k\sum_{j=0}^{n/2-1}\widetilde{P}_{(j+n/2)}\omega_n^{-kj}\right] \mod q$$

According to the parity of k, we can group them into two parts, where $k = 0, 1, ..., n/2-1$:

$$p_{2k} = n^{-1}\psi_{2n}^{-2k}\left[\sum_{j=0}^{n/2-1}\widetilde{P}_j\omega_n^{-2kj} + (-1)^{2k}\sum_{j=0}^{n/2-1}\widetilde{P}_{(j+n/2)}\omega_n^{-2kj}\right] \mod q$$

$$p_{2k+1} = n^{-1}\psi_{2n}^{-(2k+1)}\left[\sum_{j=0}^{n/2-1}\widetilde{P}_j\omega_n^{-(2k+1)j} + (-1)^{(2k+1)}\sum_{j=0}^{n/2-1}\widetilde{P}_{(j+n/2)}\omega_n^{-(2k+1)j}\right] \mod q$$

With the scaling property of twiddle factors, we can simplify

the equations as:

$$p_{2k} = (\frac{n}{2})^{-1}\psi_n^{-k}\sum_{j=0}^{n/2-1}\left[\frac{\widetilde{P}_j + \widetilde{P}_{(j+n/2)}}{2}\right]\omega_{n/2}^{-kj} \mod q$$

$$p_{2k+1} = (\frac{n}{2})^{-1}\psi_n^{-k}\sum_{j=0}^{n/2-1}\left\{\left[\frac{\widetilde{P}_j - \widetilde{P}_{(j+n/2)}}{2}\right]\psi_{2n}^{-1}\omega_n^{-j}\right\}\omega_{n/2}^{-kj} \mod q$$

Let

$$\widetilde{P}_j^{(0)} = \frac{\widetilde{P}_j + \widetilde{P}_{j+n/2}}{2} \mod q,$$
$$\widetilde{P}_j^{(1)} = \frac{\widetilde{P}_j - \widetilde{P}_{j+n/2}}{2}\psi_{2n}^{-1}\omega_n^{-j} \mod q.$$

We have

$$p_{2k} = (\frac{n}{2})^{-1}\psi_n^{-k}\sum_{j=0}^{n/2-1}\widetilde{P}_j^{(0)}\omega_{n/2}^{-kj} \mod q, \quad (35)$$

$$p_{2k+1} = (\frac{n}{2})^{-1}\psi_n^{-k}\sum_{j=0}^{n/2-1}\widetilde{P}_j^{(1)}\omega_{n/2}^{-kj} \mod q. \quad (36)$$

Similar to NTT, we can easily see that p_{2k} and p_{2k+1} are essentially the same as Equation (34) except scaled down to $n/2$ points. By recursively applying the decimation process to p_{2k} and p_{2k+1} to 2-point NTT, we can get the structure shown in Figure Fig. 2 (lower left). Note that when $n = 2$, $(\frac{n}{2})^{-1} = 1$, $\psi_n^{-k} = 1$, and also $\omega_{n/2}^{-kj} = 1$. In addition, the integers ψ_{2n}^{-1} and ω_n^{-j} can be merged to an integer, $\psi_{2n}^{-1}\omega_n^{-j} = \psi_{2n}^{-(2j+1)}$. The data flow graph of the entire 8-point low-complexity negative wrapped convolution is shown in Fig. 2.

Unlike the NTT butterfly architecture, the intermediate results after the modular addition and modular subtraction operations in the INTT butterfly need to be multiplied by $2^{-1} \mod q$. Although it seems like this will add additional multipliers to the INTT block, the modular multiplication by 2^{-1} can be implemented without a modular multiplier.

$$\frac{x}{2} \mod q = \begin{cases} \frac{x}{2} & \text{if } x \text{ is even} \\ \lfloor\frac{x}{2}\rfloor + \frac{q+1}{2} \mod q & \text{if } x \text{ is odd} \end{cases} \quad (37)$$

If x is even, $x \times 2^{-1}$ can be implemented as a right shift operation, i.e., $x \gg 1$. Here, $\lfloor\ \rfloor$ is the floor function that maps a number to the closest integer that is smaller than or equal to the number. The $\gg$ operation can be implemented easily in hardware. For example, a right shift by 1 bit operation on 8 (100 in binary) results in 4 (010 in binary).

If x is odd, $x \times 2^{-1}$ can be represented as:

$$\frac{x}{2} \equiv (2\lfloor\frac{x}{2}\rfloor + 1)\frac{q+1}{2} \mod q \quad (38)$$
$$\equiv \lfloor\frac{x}{2}\rfloor(q+1) + \frac{q+1}{2} \mod q$$
$$\equiv \lfloor\frac{x}{2}\rfloor + \frac{q+1}{2} \mod q \quad (39)$$

The term $(2\lfloor\frac{x}{2}\rfloor + 1)$ in Equation (38) is equivalent to an odd number x; the term $(\frac{q+1}{2} \mod q)$ is equivalent to $2^{-1} \mod q$ since $(\frac{q+1}{2} \times 2 \mod q) \equiv 1 \bmod q$. $\lfloor\frac{x}{2}\rfloor$ can be implemented as $(x \gg 1)$, and $(q+1)/2$ is a constant. Hence, no modular multiplications are required. This operation requires one modular adder and a multiplexer. Here, the multiplexer is used to select one of the two options in Equation (37) as output depends on whether the input is even or odd.

Alternative Low-complexity INTT for NWC

In addition to the low-complexity INTT structure (LC-INTT) presented in [9], there is another straightforward way of constructing an alternative low-complexity INTT structure. We can merge the two post-processing multipliers n^{-1} and ψ_{2n}^j into a single equivalent multiplier. The structure is shown in Fig. 2 (lower right). It is important to point out that, since $n^{-1}\psi_{2n}^0 = n^{-1}$, the multiplier located at the index 0 position at the output can be implemented either as a multiplier (denoted as Architecture Alt-LC-INTT1) or $\log n$ operations of multiplications by 2^{-1} (denoted as Architecture Alt-LC-INTT2). While the numbers of multipliers in Alt-LC-INTT1 and Alt-LC-INTT2 seem to be larger than the standard LC-INTT, the obvious advantage of these structures is that no shifting operations are required in each butterfly unit. More comparisons are presented in Section VII.

VI. Numerical Example

Fig. 3 illustrates an example of length-4 modular polynomial multiplication using zero padding and 8-point NTT/INTT. The NTT in Fig. 3 corresponds to a DIF NTT redrawn with inputs in the bit-reversed order and outputs in the normal order. The INTT in Fig. 3 is a DIF INTT. In this example, $n = 8$, $q = 17$, where $q \mod 2n \equiv 1$ and it is also a prime, $n^{-1} = 15$ $(8 \times 15 = 120 \equiv 1 \mod 17)$. Since $\psi_{2n}^n \mod q \equiv -1$, we can select $\psi_{16} = 3$ $(3^8 = 6561 \equiv -1 \mod 17)$, and $\psi_{16}^{-1} = 6$ $(3 \times 6 = 18 \equiv 1 \mod 17)$. Assume that both $a(x)$ and $b(x)$ are $x^3 + 3x^2 + 4x + 2$. To begin the computation, we need to first pad 4 zeros to the inputs, and then feed the inputs to the NTT block. After the NTT block, we will perform point-wise multiplications. Since we assume $a(x)$ and $b(x)$ are the same, the coefficients of the results of point-wise multiplications $\widetilde{P}_j$ will be $[10^2, 7^2, 4^2, 15^2, 0^2, 7^2, 11^2, 13^2] \mod 17$ that is equivalent to $[15, 15, 16, 4, 0, 15, 2, 16]$. Next, we feed those outputs from point-wise multiplications to the INTT block. The INTT block is similar to the NTT block with only two differences. One, the $2n$-th roots are now replaced with the inverse of $2n$-th roots. Two, additional multipliers are added for multiplying n^{-1}. The INTT block outputs 8 coefficients. Since we are computing modular polynomial multiplication $\mod (x^4 + 1, 17)$, the

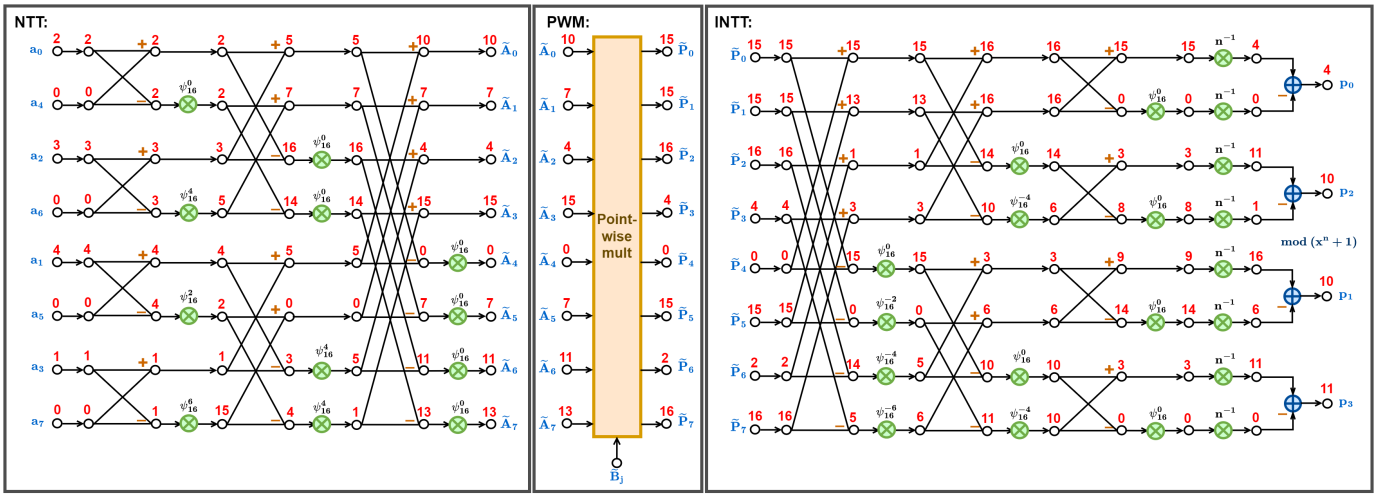

Fig. 3: An example of length-4 modular polynomial multiplication using 8-point convolution.

convolution result $x^6+6x^5+11x^3+11x^2+16x+4 \mod (x^4+1, 17)$ becomes $11x^3+(11-1)x^2+(16-6)x+(4-0) \mod 17 \equiv 11x^3+10x^2+10x+4$.

Fig. 4 illustrates simple examples of NTT and INTT for negative wrapped convolution. On the left is NWC with classic NTT/INTT. Let's consider $n=4$, $q=17$, and $n^{-1}=13$ $(4\times 13=52\equiv 1 \mod 17)$ Since $\psi_{2n}^n \mod q \equiv -1$, we can select $\psi_8=2$ $(2^4=16\equiv -1 \mod 17)$, and $\psi_8^{-1}=9$ $(2\times 9=18\equiv 1 \mod 17)$. Let's consider the same example that both $a(x)$ and $b(x)$ are (x^3+3x^2+4x+2). The first step of NWC is NTT with preprocessing, which correspond to steps 1 and 2 of Algorithm 1. We multiply each coefficient of $a(x)$ with the $2n$-th root to the power of its exponent; this gives us the weighted $\widetilde{a}(x)$. After we obtain $\widetilde{a}(x)$, we feed the weighted input into NTT. Note that twiddle factor ω is the n-th root of unity, which means $\psi^2=\omega$.

Step 3 of Algorithm 1 takes the outputs of both NTT blocks and performs point-wise multiplication. Since we assume $a(x)$ and $b(x)$ are the same, the coefficients of the results of point-wise multiplications $\widetilde{P}_j$ will be $[13^2, 7^2, 15^2, 7^2] \mod q$ that is equivalent to $[16, 15, 4, 15]$.

Steps 4 and 5 of Algorithm 1 are feeding $\widetilde{P}(x)$ into the weighted INTT block. Note that after the INTT block, there are weighted operations that multiply each coefficient with the inverse of $2n$-th root to the power of its exponent. The polynomial we obtain at the output is $11x^3+10x^2+10x+4$. We can verify this result by computing $(x^3+3x^2+4x+2)^2 \mod (x^4+1, 17) = x^6+6x^5+17x^4+28x^3+28x^2+16x+4 \mod (x^4+1, 17)$, which is equivalent to $11x^3+10x^2+10x+4$.

On the right of Fig. 4 is an example of negative wrapped convolution with low-complexity NTT/INTT. We consider the same inputs as in Fig. 4 that both $a(x)$ and $b(x)$ are x^3+3x^2+4x+2. $n=4$, $q=17$, $\psi=2$, $\psi^{-1}=9$ $(2\times 9=18\equiv 1 \mod 17)$, and $2^{-1}=9$ $(2\times 9=18\equiv 1 \mod 17)$. For the low-complexity NTT, the multipliers are now moved before the butterfly addition and subtraction. The output polynomial is the same as what we obtained from the previous example. For the standard low-complexity INTT, additional "multiplication" of 2^{-1} is required after every butterfly addition and subtraction. However, no additional multipliers are implemented according to Equation (39). The output polynomial is $11x^3+10x^2+10x+4$, the same as what we obtained from the previous example.

VII. What We Have Learned

Comparisons

Table I compares the number of multipliers implemented in each method. The first method, zero padding and a polynomial modular reduction, requires $n\log 2n$ and $n\log 2n+2n$ modular multiplications for NTT and INTT, respectively. The second method, the negative wrapped convolution, requires $\frac{n}{2}\log n+n$ and $\frac{n}{2}\log n+2n$ modular multiplications for NTT and INTT, respectively. The improvement comes from reducing $2n$-point NTT/INTT to n-point NTT/INTT. However, the tradeoffs require adding n multipliers to both NTT/INTT blocks. Last but not least, using the low-complexity NTT/INTT, we are able to remove the additional n multipliers for NTT, and the additional $2n$ multipliers for INTT.

Table I includes multipliers that multiply by $\psi_{2n}^0=1$. If we remove those multipliers, the zero-padded convolution method requires $(n\log_2 n)-n+1$ and $(n\log_2 n)+n+1$ modular multiplications for NTT and INTT, respectively. The NWC method requires $\frac{n}{2}\log n$ and $\frac{n}{2}\log n+n$ modular multiplications for NTT and INTT, respectively. The LC-NWC method requires the same number of modular multiplications for NTT and further reduces the number of modular multiplications for INTT by n. The comparison after excluding the multipliers by 1 is shown in Table II.

TABLE I: The comparison of the numbers of multipliers

	# Multipliers	
	NTT	INTT
Conv. w/ mod	$n\log 2n$	$n\log 2n+2n$
NWC	$\frac{n}{2}\log n+n$	$\frac{n}{2}\log n+2n$
LC-NWC	$\frac{n}{2}\log n$	$\frac{n}{2}\log n$

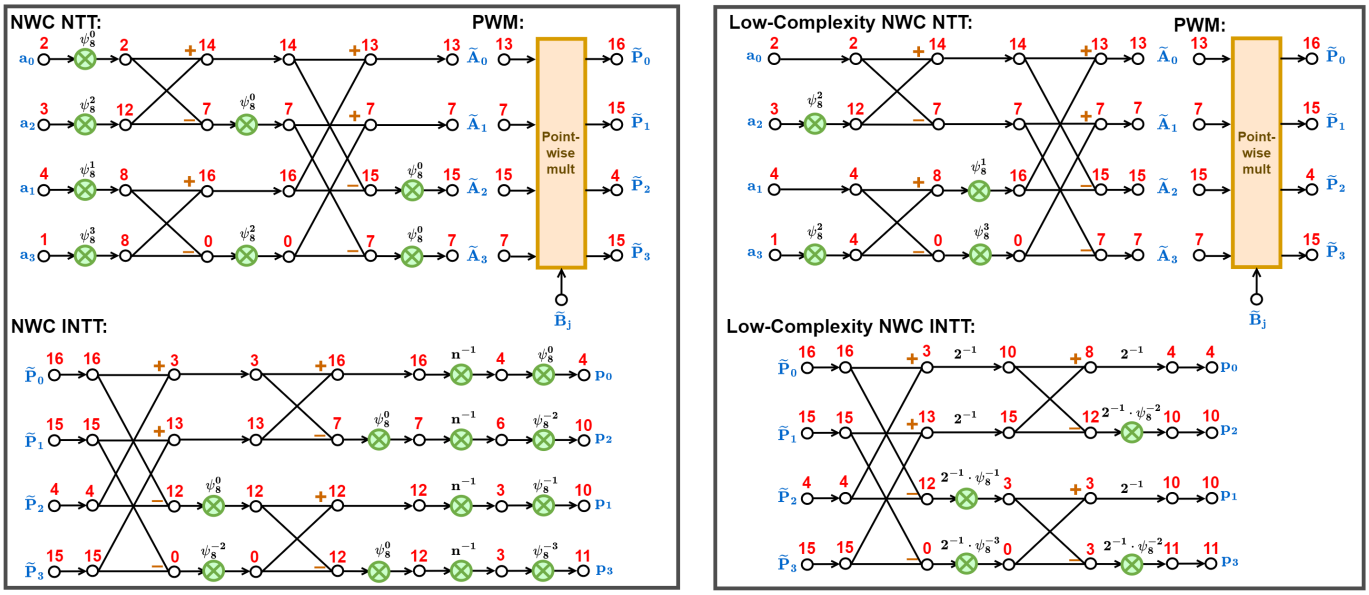

Fig. 4: An example of 4-point negative wrapped convolution.

TABLE II: The comparison of the numbers of multipliers excluding multiplication by 1

	# Multipliers	
	NTT	INTT
Conv. w/ mod	$(n\log_2 n) - n + 1$	$(n\log_2 n) + n + 1$
NWC	$\frac{n}{2}\log n$	$\frac{n}{2}\log n + n$
LC-NWC	$\frac{n}{2}\log n$	$\frac{n}{2}\log n$

TABLE III: Percentage of the number of multipliers saved compared to the zero-padded convolution method

n	Percentage of # multipliers saved			
	NWC		LC-NWC	
	NTT	INTT	NTT	INTT
1024	45.5	46.2	54.5	61.5
2048	45.8	46.4	54.2	60.7
4096	46.2	46.7	53.9	60.0

TABLE IV: Percentage of the number of multipliers saved compared to the zero-padded convolution method excluding multiplication by 1

n	Percentage of # multipliers saved			
	NWC		LC-NWC	
	NTT	INTT	NTT	INTT
1024	44.5	45.5	44.5	54.5
2048	45.0	45.8	45.0	54.2
4096	45.5	46.2	45.5	53.8

Table III illustrates how many modular multipliers can be saved by implementing the NWC methods compared to the traditional convolution theory for $n = \{1024, 2048, 4096\}$. Generally, implementing NWC will save about 46% for both NTT and INTT. If we implement NWC using low-complexity NTT/INTT, these numbers will go up to about 54% for NTT and about 60% for INTT.

Table IV illustrates how many modular multipliers can be saved by implementing the NWC methods compared to the traditional convolution theory for $n = \{1024, 2048, 4096\}$, excluding multiplication by 1. Generally, implementing NWC will save about 45% and 46% for both NTT and INTT, respectively. If we implement NWC using low-complexity NTT/INTT, the percentages saved stay the same for NTT, but the percentages go up to about 54% for INTT.

Table V shows the comparisons between the standard LC-INTT and the alternative LC-INTTs. We first compare the Alt-LC-INTT1 with LC-INTT. Although Alt-LC-INTT1 has n more multipliers compared to LC-INTT, when excluding multiplication by 1, Alt-LC-INTT1 only has one more multiplier compared to LC-INTT. The main advantage of Alt-LC-INTT1 is that it doesn't require operations for multiplication by 2^{-1}. Alt-LC-INTT2 simply replaces one multiplier in Alt-LC-INTT1 with $\log n$ operations of multiplication by 2^{-1}. The number of 2^{-1} operations in LC-INTT in Table V is $\frac{n}{2}\log n$ instead of $n\log n$, because we assume the 2^{-1} operations are merged with ψ_{2n}^{-j} in the bottom outputs of the butterfly operations. Another property worth comparing is the number of parameters that need to be stored for multiplications. For LC-INTT, we need to store the $2n$-th roots from ψ_{2n}^{1} to ψ_{2n}^{n-1}, that's a total of $n-1$ parameters. For Alt-LC-INTT1, we need to store the n-th roots from ω_n^1 to $\omega_n^{(\frac{n}{2}-1)}$ (equivalent to ψ_{2n}^{2}, ψ_{2n}^{4} to ψ_{2n}^{n-2}), and the n merged multipliers, that's a total of $\frac{3n}{2} - 1$ parameters. For Alt-LC-INTT2, we need to store a total of $\frac{3n}{2} - 2$ parameters. Therefore, Alt-LC-INTT1 and Alt-LC-INTT2 require $\frac{n}{2}$ and $\frac{n}{2} - 1$ more parameters compared to LC-INTT. The more parameters are used, the more memory

TABLE V: Comparisons between standard LC-INTT and alternative LC-INTT

	# multipliers	# multipliers excl.1	# 2^{-1} operations	# parameters
LC-INTT	$\frac{n}{2}\log n$	$\frac{n}{2}\log n$	$\frac{n}{2}\log n$	$n-1$
Alt-LC-INTT1	$\frac{n}{2}\log n + n$	$\frac{n}{2}\log n + 1$	0	$\frac{3n}{2}-1$
Alt-LC-INTT2	$\frac{n}{2}\log n + n - 1$	$\frac{n}{2}\log n$	$\log n$	$\frac{3n}{2}-2$

allocation is required to compute the result.

While there are apparent tradeoffs between LC-INTT and Alt-LC-INTT based on the word length of the inputs and the degree of the polynomials, the method of implementation will define which optimized option is better. Since the implementations of these methods aren't usually a one-to-one mapping from algorithm to hardware, different implementations will result in different tradeoffs between the two methods. Hardware implementations in prior works [9], [10], [12] have suggested that LC-NWC with LC-NTT and LC-INTT provides improvements in HE accelerators.

Conclusions

This lecture note introduced several optimization techniques for NTT-based polynomial modular multiplications. These methods include: zero-padded convolution, negative wrapped convolution, and an improved version of NWC with low-complexity NTT/INTT.

With low-complexity NTT/INTT, there is no additional polynomial reduction required after the NTT/INTT blocks and no zero-padding is required for both input polynomials. Also, compared to the classical NWC, the pre-processing and post-processing multiplications are eliminated in the low-complexity NWC.

Like in FFT, several alternate structures for NTT and INTT for NWC can be derived by twiddle shifting transformations where twiddle factors can be pushed to the next stage (or pulled to the previous stage). We point out that the low-complexity NWC structure in Fig. 2 (top) can be derived from the traditional NWC structure where the polynomial coefficients are first multiplied by ψ_{2n}^{j} by using twiddle shifting (pushing). Different alternate structures can also be derived by using transpose operation.

ACKNOWLEDGEMENT

The authors are grateful to an anonymous reviewer and Prof. Cagatay Candan, the Associate Editor, for their numerous constructive comments.

AUTHOR

Sin-Wei Chiu (chiu0091@umn.edu) received his bachelor's degree in electrical engineering from National Central University, Taiwan, in 2020. He is currently pursuing a Ph.D. degree in electrical engineering at the University of Minnesota, Twin Cities. His current research interests include VLSI architecture design, digital signal processing systems, post-quantum cryptography, and homomorphic encryption.

Keshab K. Parhi (parhi@umn.edu) received his Ph.D. degree in electrical engineering and computer sciences from the University of California, Berkeley, in 1988. He has been with the University of Minnesota, Minneapolis, Minnesota, since 1988, where he is currently the Erwin A. Kelen Chair in Electrical Engineering and a Distinguished McKnight University Professor in the Department of Electrical and Computer Engineering. He has published more than 700 papers, is the inventor of 36 patents, and has authored the textbook VLSI Digital Signal Processing Systems (Wiley, 1999). He served as the editor-in-chief of IEEE Transactions on Circuits and Systems, Part I during 2004 and 2005, and currently serves as the editor-in-Chief of the IEEE Circuits and Systems Magazine. He is a Fellow of IEEE, ACM, AIMBE, AAAS, and NAI.

REFERENCES

[1] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2010, pp. 1–23.

[2] E. Crockett, "A low-depth homomorphic circuit for logistic regression model training," Cryptology ePrint Archive, Paper 2020/1483, 2020, https://eprint.iacr.org/2020/1483. [Online]. Available: https://eprint.iacr.org/2020/1483

[3] C. Marcolla, V. Sucasas, M. Manzano, R. Bassoli, F. H. Fitzek, and N. Aaraj, "Survey on fully homomorphic encryption, theory, and applications," *Proceedings of the IEEE*, vol. 110, no. 10, pp. 1572–1609, 2022.

[4] A. Oppenheim and R. Schafer, *Discrete-Time Signal Processing*. Pearson Education, 2011. [Online]. Available: https://books.google.com/books?id=BOVyngEACAAJ

[5] R. E. Blahut, *Theory and practice of error control codes*. Addison-Wesley Pub. Co., 1983. [Online]. Available: https://cir.nii.ac.jp/crid/1130000798027694720

[6] A. Pedrouzo-Ulloa, J. R. Troncoso-Pastoriza, and F. Pérez-González, "Number theoretic transforms for secure signal processing," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 5, pp. 1125–1140, 2017.

[7] J. H. McClellan and C. M. Rader, *Number Theory in Digital Signal Processing*. Prentice Hall Professional Technical Reference, 1979.

[8] V. Lyubashevsky, D. Micciancio, C. Peikert, and A. Rosen, "SWIFFT: A modest proposal for FFT hashing," in *International Workshop on Fast Software Encryption*. Springer, 2008, pp. 54–72.

[9] N. Zhang, B. Yang, C. Chen, S. Yin, S. Wei, and L. Liu, "Highly efficient architecture of NewHope-NIST on FPGA using low-complexity NTT/INTT," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 49–72, 2020.

[10] S. S. Roy, F. Vercauteren, N. Mentens, D. D. Chen, and I. Verbauwhede, "Compact ring-LWE cryptoprocessor," in *International workshop on cryptographic hardware and embedded systems*. Springer, 2014, pp. 371–391.

[11] P. Duhamel and M. Vetterli, "Fast Fourier transforms: a tutorial review and a state of the art," *Signal processing*, vol. 19, no. 4, pp. 259–299, 1990.

[12] W. Tan, S.-W. Chiu, A. Wang, Y. Lao, and K. K. Parhi, "PaReNTT: Low-latency parallel residue number system and NTT-based long polynomial modular multiplication for homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 1646–1659, 2024.