Polynomial multiplication using NTT

September 25, 2024

1 Introduction

The Number Theoretic Transform (NTT) can efficiently calculate polynomial multiplication using the convolution theorem with a quasi-linear complexity instead of $O(d^2)$

2 Linear convolution

Suppose that $g(x)$ and $h(x)$ are polynomials of degree $d-1$ and x is the polynomial variable. Polynomial multiplication equals a discrete linear convolution between the coefficients' vectors g and h. The traditional multiplication algorithm has an $O(d^2)$ complexity.

$$\begin{aligned} y[k] &= (g * h)[k] \\ &= \sum_{i=0}^{k} g[i]h[k-i] \end{aligned}$$

2.1 Example

If $g(x) = 1 + 2x + 3x^2 + 4x^3$ and $h(x) = 5 + 6x + 7x^2 + 8x^3$ or in vector notation: $g = [1, 2, 3, 4]$, and $h = [5, 6, 7, 8]$. The linear convolution result is $y(x) = 5 + 16x + 34x^2 + 60x^3 + 61x^4 + 52x^5 + 32x^6$, or in vector notation, $y = [5, 16, 34, 60, 61, 52, 32]$.

$$\begin{array}{r} 1 + 2x + 3x^2 + 4x^3 \\ 5 + 6x + 7x^2 + 8x^3 \\ \hline 8x^3 + 16x^4 + 24x^5 + 32x^6 \quad \times \\ 7x^2 + 14x^3 + 21x^4 + 28x^5 \\ 6x + 12x^2 + 18x^3 + 24x^4 \\ 5 + 10x + 15x^2 + 20x^3 \\ \hline 5 + 16x + 34x^2 + 60x^3 + 61x^4 + 52x^5 + 32x^6 \quad + \end{array}$$

Figure 1: The traditional multiplication

```cpp
void mult_poly_naive(const std::vector<uint64_t> & p1,
                     const std::vector<uint64_t> & p2,
                     std::vector<uint64_t>& result)
{
    for (int i = 0; i < p1.size(); ++i)
        for (int j = 0; j < p2.size(); ++j)
            result[i + j] += p1[i] * p2[j];
}
```

Figure 2: The traditional multiplication in C++

3 Integer ring modulo $\mathbb{Z}_q$

An integer ring modulo q (also called integers modulo a prime number), denoted by $\mathbb{Z}_q$, is a mathematical structure known as a cyclic group or finite field in abstract algebra. Specifically, it is the set of residue classes of integers modulo q. This means it's the set of all possible remainders when integers are divided by q. For example, if q equals 2, then $\mathbb{Z}_q$ (which is $\mathbb{Z}_2$ in this case) includes only 0 and 1 because any integer either leaves a remainder of 0 or 1 when divided by 2. In the field of cryptography, we typically work with finite fields. The exciting thing about working in a finite field is that the polynomial coefficients "wrap around" when multiplied. So, regardless of how much polynomial arithmetic we perform, the polynomial coefficients can still be bounded in a fixed range, which is very attractive from an implementation perspective.

```cpp
void mult_poly_naive_q(const std::vector<uint64_t>& p1,
                       const std::vector<uint64_t>& p2,
                       uint64_t q, std::vector<uint64_t>& result) {
    mult_poly_naive(p1,p2, result);
    for (uint64_t & i : result) i = i % q;
}
```

Figure 3: Integer ring modulo $\mathbb{Z}_q$ in C++

3.1 Example

If q equals 17, then $\mathbb{Z}_q$ (which is $\mathbb{Z}_{17}$ in this case) includes only 0 to 16 because any integer leaves a remainder of 0 to 16 when divided by 17. Hence the vector multiplication for vectors $g = [1, 2, 3, 4]$, and $h = [5, 6, 7, 8]$ with modulo a prime number $q = 17$ is $[5, 16, 0, 9, 10, 1, 15]$

4 Positive wrapped convolution (cyclic convolution)

However, one can quickly notice an item that is not so positive in the current setting - the degree of the polynomial will only grow larger and larger as we perform more multiplications. That means we need a more extended array to store coefficients and more complex convolutions every time a

multiplication is performed. It would be great if the degree of the polynomials could "wrap around" just like the coefficients. As it turns out, another algebraic structure is precisely for that - Polynomial Quotient Rings. Like in the base field $\mathbb{Z}_q$, where we would take modulo q after every arithmetic operation, we can also take modulo some polynomial $\Phi(x)$ after every polynomial operation. The resulting polynomial's degree would never be more significant or equal to the degree of $\Phi(x)$. We call such structure $\mathbb{Z}_q[x]/(\Phi(x))$. In the context of lattice-based Cryptography, $\Phi(x)$ is chosen to be a polynomial looking either like $x^d - 1$ or $x^d + 1$. A cyclic convolution or positive wrapped convolution, $PWC(x)$ is defined as: $PWC(x) = \mathbb{Z}_q/x^d - 1$

```
void mult_poly_naive_q_cc(const std::vector<uint64_t>& p1,
                          const std::vector<uint64_t>& p2,
                          uint64_t q,
                          uint64_t d,
                          std::vector<uint64_t>& result) {
    mult_poly_naive_q(p1,p2,q, & result);
    for (uint64_t i = d; i < result.size(); ++i) {
        result[i - d] = (result[i - d] + result[i]) % q;
        result[i] = 0;
    }
}
```

Figure 4: Positive wrapped convolution (cyclic convolution) in C++

4.1 Example

If $g(x) = 1 + 2x + 3x^2 + 4x^3$ and $h(x) = 5 + 6x + 7x^2 + 8x^3$ or in vector notation: $g = [1, 2, 3, 4]$, and $h = [5, 6, 7, 8]$. We already have calculated the linear convolution result is $y(x) = 5 + 16x + 34x^2 + 60x^3 + 61x^4 + 52x^5 + 32x^6$, or in vector notation, $y = [5, 16, 34, 60, 61, 52, 32]$. If we don't apply an integer ring modulo q, then $PWC(x) = y(x) \bmod x^d - 1$. We need a long division by $x^d - 1$ to calculate positive wrapped convolution. Thus, the result of the cyclic convolution is $PWC(x) = 66 + 68x + 66x^2 + 60x^3$ or $[66, 68, 66, 60]$. Notice that we present the result sorted in increasing power.

$$
\begin{array}{r|l}
 & 32x^2 + 52x + 61 \\
\hline
x^4 - 1 & 32x^6 + 52x^5 + 61x^4 + 60x^3 + 34x^2 + 16x + 5 \\
 & \underline{32x^6 + \ 0x^5 + \ 0x^4 + \ 0x^3 - 32x^2} \\
 & \quad 52x^5 + 61x^4 + 60x^3 + 66x^2 + 16x + 5 \\
 & \quad \underline{52x^5 + \ 0x^4 + \ 0x^3 + \ 0x^2 - 52x} \\
 & \quad\quad 61x^4 + 60x^3 + 66x^2 + 68x + 5 \\
 & \quad\quad \underline{61x^4 + \ 0x^3 + \ 0x^2 + 0x - 61} \\
 & \quad\quad\quad 60x^3 + 66x^2 + 68x + 66
\end{array}
$$

Figure 5: Schoolbook method for positive wrapped convolution (cyclic convolution)

5 Negative wrapped convolution (negacyclic convolution)

A cyclic convolution or negative wrapped convolution, $NWC(x)$ is defined as: $PWC(x) = \mathbb{Z}_q/x^d + 1$

```
void mult_poly_naive_q_nwc(const std::vector<uint64_t>& p1,
                           const std::vector<uint64_t>& p2,
                           uint64_t q,
                           uint64_t d,
                           std::vector<uint64_t>& result) {
    mult_poly_naive_q(p1,p2,q, &: result);
    for (uint64_t i = d; i < result.size(); ++i) {
        int64_t tmp = result[i - d] - result[i];
        result[i - d] = tmp > 0 ? tmp % q : tmp + q;
        result[i] = 0;
    }
}
```

Figure 6: Negative wrapped convolution (negacyclic convolution) in C++

5.1 Example

If $g(x) = 1 + 2x + 3x^2 + 4x^3$ and $h(x) = 5 + 6x + 7x^2 + 8x^3$ or in vector notation: $g = [1, 2, 3, 4]$, and $h = [5, 6, 7, 8]$. We already have calculated the linear convolution result is $y(x) = 5 + 16x + 34x^2 + 60x^3 + 61x^4 + 52x^5 + 32x^6$, or in vector notation, $y = [5, 16, 34, 60, 61, 52, 32]$. If we don't apply an integer ring modulo q, then $NWC(x) = y(x) \bmod x^d + 1$. We need a long division by $x^d + 1$ to calculate positive wrapped convolution. Thus, the result of the negacyclic convolution is $NWC(x) = -56 - 36x + 2x^2 + 60x^3$ or $[-56, -36, 2, 60]$. Notice that we present the result sorted in increasing power.

$$
\begin{array}{r|l}
 & 32x^2 + 52x + 61 \\
\hline
x^4 + 1 & 32x^6 + 52x^5 + 61x^4 + 60x^3 + 34x^2 + 16x + 5 \\
 & \underline{32x^6 + \ 0x^5 + \ 0x^4 + \ 0x^3 + 32x^2} \quad - \\
 & \qquad 52x^5 + 61x^4 + 60x^3 + \ 2x^2 + 16x + 5 \\
 & \qquad \underline{52x^5 + \ 0x^4 + \ 0x^3 + \ 0x^2 + 52x} \quad - \\
 & \qquad\qquad 61x^4 + 60x^3 + \ 2x^2 - 36x \ + 5 \\
 & \qquad\qquad \underline{61x^4 \ + \ 0x^3 + \ 0x^2 + \ 0x + 61} \quad - \\
 & \qquad\qquad\qquad 60x^3 + 2x^2 - 36x - 56
\end{array}
$$

Figure 7: Schoolbook method for negative wrapped convolution (negacyclic convolution)

6 NTT-Based Convolutions

In this section, we present the basics of NTT-based convolutions.

6.1 Congruence

The symbol ≡ denotes congruence and is different from equality. While equality means that items are the same, congruence means items have the same remainder when divided by mod. In a ring $\mathbb{Z}_{7681}$ and $d = 4$, the 4-th root of unity, which satisfy the condition $\omega^4 \equiv 1 \mod 7681$ are $3383, 4298, 7680$

6.2 Primitive n-th Root of Unity

Let $\mathbb{Z}_q$ be an integer ring modulo q, and $d-1$ is the polynomial degree of g(x) and h(x). ω is a primitive n-th root of unity in $\mathbb{Z}_q$ if and only if: $\omega^d \equiv 1 \mod q$ and $\omega^k \not\equiv 1 \mod q$ for $k < d$.

6.2.1 Example

In a ring $\mathbb{Z}_{7681}$ and $d = 4$, the 4-th root of unity, which satisfy the condition $\omega^4 \equiv 1 \mod 7681$ are $3383, 4298, 7680$. Out of three roots, 7680 is not a primitive n-th root of unity, as there exists k = 2 < d that satisfy $\omega^d \equiv 1 \mod 7691$ Therefore ω = 3383 or ω = 4298 are the primitive 4-th root of unity in $\mathbb{Z}_{7681}$. The value of ω will be important in calculating NTT and positive-wrapped convolution.

6.3 Primitive 2n-th Root of Unity

Let $\mathbb{Z}_q$ be an integer ring modulo q, and $d-1$ is the polynomial degree of g(x) and h(x) and ω is a primitive n-th root. ψ is a primitive 2n-th root of unity in $\mathbb{Z}_q$ if and only if: $\psi^2 \equiv \omega \mod q$ and $\psi^d \equiv -1 \mod q$.

6.3.1 Example

In a ring $\mathbb{Z}_{7681}$, $d = 4$, and ω = 3383, the 2n-th root of unity, which satisfies the above conditions, are 1925 and 5756 as $1925^2 = 5756^2 \equiv 3383 \bmod 7681$ and 1925^4 = 5756^4 = $7680 \equiv -1 \bmod 7681$. The value of ψ will be significant in calculating NTT and negative-wrapped convolution.

6.4 Modular multiplicative inverse

In mathematics, particularly in arithmetic, a modular multiplicative inverse of an integer a is an integer x such that the product a*x is congruent to 1 to the modulus q. In the standard notation of modular arithmetic, this congruence is written as a*x ≡ 1 mod q which is the shorthand way of writing the statement that m divides (evenly) the quantity ax - 1, or, put another way, the remainder after dividing ax by the integer m is 1. You can also use the pow()function from Python to calculate modular multiplicative inverse i.e. 5761 = pow(4, -1, 7681), 4298 = pow(3383, -1, 7681), 1213 = pow(1925, -1, 7681)

6.4.1 Example

Let $\mathbb{Z}_q$ be an integer ring modulo q = 7681, d is the polynomial degree with d =4, ω is primitive n-th Root of Unity with ω = 3383 and ψ is a primitive 2n-th root of unity with ψ = 1925. Following are the modular multiplicative inverse for d, ω and ψ: d^{-1} = 5761, ω^{-1} = 4298 and ψ^{-1} = 1213.

7 NTT-Based Positive-Wrapped Convolution

The NTT of a polynomial does not have any physical meaning, unlike Discrete Fourier Transform (DFT) which represents a signal in the frequency domain. However, NTT preserves one of the essential properties of DFT: the convolution theorem, which is valuable in calculating polynomial multiplication.

7.1 NTT Based on ω

The Number Theoretic Transform (NTT) of a vector of polynomial coefficients a is defined as $\hat{a}$ = NTT(a), where:

$$\hat{a_j} = \sum_{i=0}^{d-1} \omega^{ij} a_i \bmod q$$

and j = 0, 1, 2, . . . , d - 1

7.1.1 Example

If $g(x) = 1 + 2x + 3x^2 + 4x^3$ or in vector notation: $g = [1, 2, 3, 4]$. In a ring $\mathbb{Z}_{7681}$, $d = 4$ and $\omega = 3383$. Thus, the NTT of g can be calculated by the following matrix multiplication:

$$\hat{g} = \begin{bmatrix} \omega^{0\times0} & \omega^{0\times1} & \omega^{0\times2} & \omega^{0\times3} \\ \omega^{1\times0} & \omega^{1\times1} & \omega^{1\times2} & \omega^{1\times3} \\ \omega^{2\times0} & \omega^{2\times1} & \omega^{2\times2} & \omega^{2\times3} \\ \omega^{3\times0} & \omega^{3\times1} & \omega^{3\times2} & \omega^{3\times3} \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

$$\hat{g} = \begin{bmatrix} \omega^{0} & \omega^{0} & \omega^{0} & \omega^{0} \\ \omega^{0} & \omega^{1} & \omega^{2} & \omega^{3} \\ \omega^{0} & \omega^{2} & \omega^{4} & \omega^{6} \\ \omega^{0} & \omega^{3} & \omega^{6} & \omega^{9} \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

$$\hat{g} = \begin{bmatrix} \omega^{0} & \omega^{0} & \omega^{0} & \omega^{0} \\ \omega^{0} & \omega^{1} & \omega^{2} & \omega^{3} \\ \omega^{0} & \omega^{2} & \omega^{0} & \omega^{2} \\ \omega^{0} & \omega^{3} & \omega^{2} & \omega^{1} \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

$$\hat{g} = \begin{bmatrix} 3383^{0} & 3383^{0} & 3383^{0} & 3383^{0} \\ 3383^{0} & 3383^{1} & 3383^{2} & 3383^{3} \\ 3383^{0} & 3383^{2} & 3383^{0} & 3383^{2} \\ 3383^{0} & 3383^{3} & 3383^{2} & 3383^{1} \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

$$\hat{g} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 3383 & 7680 & 4298 \\ 1 & 7680 & 1 & 7680 \\ 1 & 4298 & 7680 & 3383 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

$$\hat{g} = \begin{bmatrix} 10 \\ 913 \\ 7679 \\ 6764 \end{bmatrix}$$

Figure 8: NTT calculations example

Therefore, the NTT(g) = $\hat{g}$ = [10, 913, 7679, 6764] in $\mathbb{Z}_{7681}$, $d = 4$ and $\omega = 3383$. Note that the NTT of a particular polynomial is not always unique. It depends on the choice of ω. The NTT result will differ if one uses $\omega = 4298$ instead of $\omega = 3383$.

7.2 INTT Based on ω

The Inverse of Number Theoretic Transform (INTT) of an NTT vector $\hat{a}$ is defined as a = NTT($\hat{a}$), where:

$$a_i = d^{-1} \sum_{i=0}^{d-1} \omega^{-ij} \hat{a}_j \text{ mod q}$$

and j = 0, 1, 2, . . . , d - 1

Note that the INTT has a formula that is very similar to NTT. The only differences are ω replaced by its inverse in $\mathbb{Z}_q$. It always holds that a = INTT(NTT(a)).

7.2.1 Example

If NTT(g) = $\hat{g}$ = $[10, 913, 7679, 6764]$. In a ring $\mathbb{Z}_{7681}$, $d = 4$ and $\omega = 3383$. Thus, the INTT of $\hat{g}$ can be calculated by the following matrix multiplication:

$$g = n^{-1} \begin{bmatrix} \omega^{-0\times0} & \omega^{-0\times1} & \omega^{-0\times2} & \omega^{-0\times3} \\ \omega^{-1\times0} & \omega^{-1\times1} & \omega^{-1\times2} & \omega^{-1\times3} \\ \omega^{-2\times0} & \omega^{-2\times1} & \omega^{-2\times2} & \omega^{-2\times3} \\ \omega^{-3\times0} & \omega^{-3\times1} & \omega^{-3\times2} & \omega^{-3\times3} \end{bmatrix} \begin{bmatrix} 10 \\ 913 \\ 7679 \\ 6764 \end{bmatrix}$$

$$g = n^{-1} \begin{bmatrix} \omega^{0} & \omega^{0} & \omega^{0} & \omega^{0} \\ \omega^{0} & \omega^{-1} & \omega^{-2} & \omega^{-3} \\ \omega^{0} & \omega^{-2} & \omega^{-4} & \omega^{-6} \\ \omega^{0} & \omega^{-3} & \omega^{-6} & \omega^{-9} \end{bmatrix} \begin{bmatrix} 10 \\ 913 \\ 7679 \\ 6764 \end{bmatrix}$$

$$g = n^{-1} \begin{bmatrix} \omega^{0} & \omega^{0} & \omega^{0} & \omega^{0} \\ \omega^{0} & \omega^{-1} & \omega^{-2} & \omega^{-3} \\ \omega^{0} & \omega^{-2} & \omega^{-0} & \omega^{-2} \\ \omega^{0} & \omega^{-3} & \omega^{-2} & \omega^{-1} \end{bmatrix} \begin{bmatrix} 10 \\ 913 \\ 7679 \\ 6764 \end{bmatrix}$$

$$g = 5761 \begin{bmatrix} 4298^{0} & 4298^{0} & 4298^{0} & 4298^{0} \\ 4298^{0} & 4298^{1} & 4298^{2} & 4298^{3} \\ 4298^{0} & 4298^{2} & 4298^{0} & 4298^{2} \\ 4298^{0} & 4298^{3} & 4298^{2} & 4298^{1} \end{bmatrix} \begin{bmatrix} 10 \\ 913 \\ 7679 \\ 6764 \end{bmatrix}$$

$$g = 5761 \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 4298 & 7680 & 3383 \\ 1 & 7680 & 1 & 7680 \\ 1 & 3383 & 7680 & 4298 \end{bmatrix} \begin{bmatrix} 10 \\ 913 \\ 7679 \\ 6764 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

Figure 9: INTT calculations example

In the figure above $n^{-1} = d^{-1} = 4^{-1} = 5761$. Therefore, the INTT($\hat{g}$) = g = [1, 2, 3, 4], which is the initial polynomial coefficient given

7.3 Using NTT to Calculate Positive-Wrapped Convolutions

Because NTT is a variant of DFT in the polynomial ring, one can apply DFT convolution theorem to calculate positive-wrapped convolution. Let g and h are the multiplicand vectors of polynomial coefficients. The positive-wrapped convolution of g and h, PWC can be calculated by: PWC = INTT(NTT(g) ∘ NTT(h))

where ∘ is an element-wise vector multiplication in $\mathbb{Z}_q$.

7.3.1 Example

Let g = [1,2,3,4] and h = [5,6,7,8]. The NTT of them are $\mathbb{Z}_{7681}$ are ($\hat{g}$) = $[10, 913, 7679, 6764]$ and ($\hat{h}$) = $[10, 913, 7679, 6764]$ when $\omega = 3383$. We can calculate their positive-wrapped convolution by:

$$INTT(\begin{bmatrix}10\\913\\7679\\6764\end{bmatrix} \circ \begin{bmatrix}26\\913\\7679\\6764\end{bmatrix}) = INTT(\begin{bmatrix}260\\4021\\4\\3660\end{bmatrix}) = 5761 \begin{bmatrix}1 & 1 & 1 & 1\\1 & 4298 & 7680 & 3383\\1 & 7680 & 1 & 7680\\1 & 3383 & 7680 & 4298\end{bmatrix} \begin{bmatrix}260\\4021\\4\\3660\end{bmatrix} = \begin{bmatrix}66\\68\\66\\60\end{bmatrix}$$

Figure 10: Using NTT to Calculate PWC

PWC = $[66, 68, 66, 60]$, the same result as calculated by schoolbook multiplication and long division

8 NTT-Based Negative-Wrapped Convolution

This section explains the definition of Number Theoretic Transform (NTT) and its inverse (INTT) based on the 2n-th root of unity, ψ.

8.1 NTT Based on ψ

The Number Theoretic Transform (NTT^{ψ}) of a vector of polynomial coefficients a is defined as $\hat{a}$ = NTT^{ψ}(a), where:

$$\hat{a_j} = \sum_{i=0}^{d-1} \psi^{2ij+i} a_i \text{ mod q}$$

8.1.1 Example

If $g(x) = 1 + 2x + 3x^2 + 4x^3$ or in vector notation: $g = [1, 2, 3, 4]$. In a ring $\mathbb{Z}_{7681}$, $d = 4$ and $\psi = 1925$. Thus, the matrix multiplication using the above equation can calculate the NTT^{ψ} of g.

NTT^{ψ}(g) = $\hat{g}$ = $[1467, 2807, 3471, 7621]$.

8.2 INTT Based on ψ

The Inverse of the Number Theoretic Transform (INTT) of an NTT vector $\hat{a}$ is defined as a = $INTT^{\psi}(\hat{a})$, where:

$$a_i = d^{-1} \sum_{i=0}^{d-1} \psi^{-(2ii+j)j} \hat{a}_j \text{ mod q}$$

and j = 0, 1, 2, . . . , d - 1

8.2.1 Example

Let NTT^{ψ}(g) = $\hat{g}$ = $[1467, 2807, 3471, 7621]$. In a ring $\mathbb{Z}_{7681}$, $d = 4$, $\omega = 3383$ and $\psi = 1925$. Thus, the INTT of $\hat{g}$ can be calculated by matrix multiplication. In the equation d^{-1} = 4^{-1} = 5761 and $/psi^{-1}$ = 1925^{-1} = 1213. Therefore, the INTT($\hat{g}$) = g = [1, 2, 3, 4], which is the initial polynomial coefficient given

8.3 Using NTT to Calculate Negative-Wrapped Convolutions

Let g and h are the multiplicands vectors of polynomial coefficients. The negative-wrapped convolution of g and h, NWC can be calculated by: NWC = $INTT^{\psi}(NTT^{\psi}$(g) $\circ$ NTT^{ψ}(h))

where $\circ$ is an element-wise vector multiplication in $\mathbb{Z}_q$.

```
void test_hexl_ntt_nwc(uint64_t *result, const uint64_t *p1, const uint64_t *p2,
                       uint64_t degree, uint64_t modulus) {
    auto ntt : unique_ptr<NTT>  = hexl_ntt::create_ntt( N: degree, modulus);
    std::vector<uint64_t> output( n: degree);
    std::vector<uint64_t> p1_output( n: degree);
    std::vector<uint64_t> p2_output( n: degree);
    hexl_ntt::compute_forward( result: p1_output.data(),  operand: p1,  &: ntt);
    hexl_ntt::compute_forward( result: p2_output.data(),  operand: p2,  &: ntt);
    hexl_ntt::eltwise_mult_mod( result: output.data(),  p1: p1_output.data(),
                               p2: p2_output.data(), degree, modulus);
    hexl_ntt::compute_inverse(result,  operand: output.data(),  &: ntt);
}
```

Figure 11: Using HEXL NTT to Calculate NWC in C++

8.3.1 Example

Let g = [1,2,3,4] and h = [5,6,7,8]. The NTT^{ψ} of them are $(\hat{g}) = [1467, 2807, 3471, 7621]$ and $(\hat{h}) = [2489, 7489, 6478, 6607]$. In a ring$\mathbb{Z}_{7681}$, $\omega = 3383$ and $\psi = 1925$. NWC = $[7625, 7645, 2, 60]$, the same result calculated by schoolbook multiplication and long division when written with negative numbers [-56, -36, 2, 60].