Understanding the Number Theoretic Transform (NTT)

Step-by-Step Matrix Examples, Optimizations, and Bit-Reversal

Prepared for Educational Purposes

June 5, 2025

Why NTT?

- NTT is used for efficient polynomial multiplication over finite fields.
- Replaces $O(n^2)$ naive multiplication with $O(n \log n)$.
- Crucial in lattice-based post-quantum cryptography (e.g., Kyber).

Polynomial Convolutions

- **Linear convolution:** Full multiplication, length $2n - 1$.
- **Cyclic convolution:** Wraps modulo $x^n - 1$.
- **Negacyclic convolution:** Wraps modulo $x^n + 1$ (used in Kyber).

NTT: Mathematical Definition

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

- ω: Primitive nth root of unity in $\mathbb{Z}_q$
- Operates like DFT but in modular arithmetic.

NTT Setup

- Polynomial: $a = [1, 2, 3, 4] = 1 + 2x + 3x^2 + 4x^3$
- Modulus: $q = 17$, length: $n = 4$
- Primitive root: $\omega = 4$
- Powers of ω: $\omega^0 = 1$, $\omega^1 = 4$, $\omega^2 = 16$, $\omega^3 = 13$

NTT Matrix

$$\hat{a} = \begin{bmatrix} \omega^{0\cdot 0} & \omega^{0\cdot 1} & \omega^{0\cdot 2} & \omega^{0\cdot 3} \\ \omega^{1\cdot 0} & \omega^{1\cdot 1} & \omega^{1\cdot 2} & \omega^{1\cdot 3} \\ \omega^{2\cdot 0} & \omega^{2\cdot 1} & \omega^{2\cdot 2} & \omega^{2\cdot 3} \\ \omega^{3\cdot 0} & \omega^{3\cdot 1} & \omega^{3\cdot 2} & \omega^{3\cdot 3} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \mod 17$$

$$\Downarrow$$

$$\hat{a} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 4 & 16 & 13 \\ 1 & 16 & 1 & 16 \\ 1 & 13 & 16 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} \mod 17$$

NTT Row By Row Calculation

$$\hat{a}_0 = 1 + 2 + 3 + 4 = 10 \mod 17 = 10$$

$$\hat{a}_1 = 1 + 8 + 48 + 52 = 109 \Rightarrow 109 \mod 17 = 7$$

$$\hat{a}_2 = 1 + 32 + 3 + 64 = 100 \Rightarrow 100 \mod 17 = 15$$

$$\hat{a}_3 = 1 + 26 + 48 + 16 = 91 \Rightarrow 91 \mod 17 = 6$$

$$\text{final result: } \hat{a} = [10,\ 7,\ 15,\ 6]$$

Inverse NTT (INTT)

$$a_i = n^{-1} \sum_{j=0}^{n-1} \hat{a}_j \cdot \omega^{-ij} \mod q$$

- $n = 4 \Rightarrow n^{-1} = 13$ in $\mathbb{Z}_{17}$
- $\omega^{-1} = 13$ in $\mathbb{Z}_{17}$ (inverse of 4)
- Use ω^{-1} powers: $1, 13, 16, 4$

Inverse NTT Matrix Setup

- $\omega^{-1} = 13$, $\omega^{-2} = 16$, $\omega^{-3} = 4$, $n^{-1} = 13$
- INTT matrix formula:

$$a = n^{-1} M^{-1} \hat{a}$$

$$\Downarrow$$

$$a = 13 \begin{bmatrix} \omega^{-0\cdot0} & \omega^{-0\cdot1} & \omega^{-0\cdot2} & \omega^{-0\cdot3} \\ \omega^{-1\cdot0} & \omega^{-1\cdot1} & \omega^{-1\cdot2} & \omega^{-1\cdot3} \\ \omega^{-2\cdot0} & \omega^{-2\cdot1} & \omega^{-2\cdot2} & \omega^{-2\cdot3} \\ \omega^{-3\cdot0} & \omega^{-3\cdot1} & \omega^{-3\cdot2} & \omega^{-3\cdot3} \end{bmatrix} \begin{bmatrix} \hat{a}_0 \\ \hat{a}_1 \\ \hat{a}_2 \\ \hat{a}_3 \end{bmatrix}$$

$$\Downarrow$$

$$a = 13 \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 13 & 16 & 4 \\ 1 & 16 & 1 & 16 \\ 1 & 4 & 16 & 13 \end{bmatrix} \begin{bmatrix} 10 \\ 7 \\ 15 \\ 6 \end{bmatrix} = [1,\ 2,\ 3,\ 4]$$

Optimizing NTT to $O(n \log n)$

- Avoid full matrix-vector multiplication.
- Use divide-and-conquer: Cooley-Tukey algorithm.
- Operates in stages with "butterfly" operations combining pairs of inputs.
- Uses symmetry and periodicity of roots of unity.

Negacyclic NTT Overview

- Negacyclic convolution computes $(a * b) \bmod (x^n + 1)$.
- Used in cryptosystems like Kyber (for noise-resilient polynomial multiplication).
- Input/output: degree-$(n-1)$ polynomials over $\mathbb{Z}_q$.
- Requires ψ such that $\psi^2 \equiv \omega \mod q$ and $\psi^n \equiv -1 \mod q$.

Why Negacyclic?

- Regular cyclic NTT wraps around $x^n - 1$: creates aliasing in mod-q arithmetic.
- Negacyclic $x^n + 1$ enforces sign flipping at boundaries.
- Important for error-correcting behavior and security in LWE-based crypto.

Negacyclic NTT Example Setup

Let:

- $q = 17$, $n = 4$
- $\zeta = 9$ a $2n$-th primitive root of unity ($\zeta^8 \equiv 1$, $\zeta^4 \equiv -1$)
- $\omega = \zeta^2 = 13$ (primitive 4th root of unity with $\omega^2 = -1$)
- Polynomial $a(x) = 1 + 2x + 3x^2 + 4x^3$

Negacyclic NTT via Twisted Multiplication

Step 1: Multiply input by powers of ζ (twisting)

$$a'(i) = a(i) \cdot \zeta^i \quad \text{for } i = 0, 1, 2, 3$$

$$a' = [1, 2 \cdot 9, 3 \cdot 13, 4 \cdot 15] = [1, 18, 39, 60] \mod 17 = [1, 1, 5, 9]$$

Step 2: Perform standard NTT on a' using $\omega = 13$

NTT of Twisted Input

Use:

$$\omega = 13, \quad \omega^0 = 1,\ \omega^1 = 13,\ \omega^2 = 16,\ \omega^3 = 4$$

NTT matrix:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 13 & 16 & 4 \\ 1 & 16 & 1 & 16 \\ 1 & 4 & 13 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 5 \\ 9 \end{bmatrix} \quad \text{mod } 17 \Rightarrow [16,\ 12,\ 13,\ 2]$$

Negacyclic Output

Step 3: Multiply NTT result by powers of ζ^i again (twist back)

$$\zeta^i = [1, 9, 13, 15], \quad \text{componentwise mult}$$

$$[16, 12, 13, 2] \circ [1, 9, 13, 15] = [16, 108, 169, 30] \mod 17 = [16, 6, 16, 13]$$

Negacyclic NTT output: $[16,\ 6,\ 16,\ 13]$

Fast NTT Steps

- Recursively split inputs into even and odd indexed parts.
- Combine using:

$$F[k] = F_e[k] + \omega^k F_o[k], \quad F[k + n/2] = F_e[k] - \omega^k F_o[k]$$

- Complexity: $O(n \log n)$

Bit-Reversal Ordering

- Bit-reversal: reversing binary representation of indices.
- Example: $3_{10} = 011_2 \rightarrow 110_2 = 6_{10}$
- Input data is reordered to align with recursive structure of FFT/NTT.

Why Bit-Reversal is Used

- Ensures correct pairwise operations in butterfly stages.
- Enables in-place computation and efficient memory access.
- Avoids overhead of extra permutations between stages.

Final Notes

- NTT is essential for fast polynomial arithmetic in cryptography.
- Matrix form shows the structure; Cooley-Tukey brings speed.
- Bit-reversal ensures algorithm efficiency on both CPU and hardware.