Number Theoretic Transform (NTT)

January 20, 2025

Distributed Lab

zkdl-camp.github.io

github.com/ZKDL-Camp

Plan

Recap on Interpolation

Polynomial Interpolation

Notice

All the previous protocols use the idea that polynomials are **universal data encoders**. We can encode any set of scalars $(a_0, \ldots, a_{N-1}) \in \mathbb{F}^N$ using **interpolation**:

$$p(x_j) = a_j, \quad j = 0, \ldots, N-1, \quad \{x_j\}_{j \in [N]} \text{ are fixed}$$

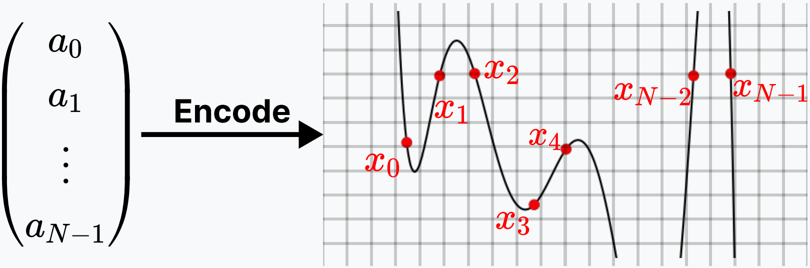

Figure: Polynomial Interpolation as a universal encoder.

Polynomial Interpolation

Example

In Groth16, we used interpolation of $3n$ polynomials:

$$L_j(i) = \ell_{i,j}, \quad R_j(i) = r_{i,j}, \quad O_j(i) = o_{i,j},$$

where $\ell_{i,j}, r_{i,j}, o_{i,j}$ are the elements of constraint matrices L, R, O (left, right, and output).

However, in PlonK we have witnessed $a(\omega^j) = A_j$ where A_j are the elements of the left trace vector A.

Question

What the heck is this ω? Why do we need it? How it helps?

A	B	C
2	3	6
6	3	9
9	✗	8

$a(1) = 2$ $\quad b(1) = 3$ $\quad c(1) = 6$

$a(\omega) = 6$ $\quad b(\omega) = 3$ $\quad c(\omega) = 9$

$a(\omega^2) = 9$ $\quad b(\omega^2) = 0$ $\quad c(\omega^2) = 8$

$\downarrow \quad \downarrow \quad \downarrow$

$a(x)$ $\quad b(x)$ $\quad c(x)$

Why we need something advanced?

Recall

The interpolation formula in given by:

$$p(x) = \sum_{i=0}^{N-1} a_i \cdot \ell_i(x), \quad \ell_i(x) = \prod_{j=0, j\neq i}^{N-1} \frac{x - x_j}{x_i - x_j}$$

Question

What is the naive complexity of this interpolation implementation?

Observation

Through careful choice of $\{x_j\}_{j\in[N]}$, we can reduce the complexity of interpolation, multiplication, or other complex operations to $\mathcal{O}(N \log N)$. **Spoiler:** we will use the nth roots of unity domain $\Omega = \{\omega^j\}_{j\in[N]}$. Let us see why it helps.

Roots of Unity

Multiplicative Subgroup.

We know that $\mathbb{F}_p$ is a **field**: we have a usual arithmetic $+, \times$.

Question

Does $(\mathbb{F}_p, \times)$ form a group?

No, since 0 does not have an inverse. But, if we consider $(\mathbb{F}_p \setminus \{0\}, \times)$, we do have a group structure!

Definition

A **multiplicative group** of a finite field $\mathbb{F}$, denoted as $\mathbb{F}^\times$, is a multiplicative group $(\mathbb{F} \setminus \{0\}, \times)$.

Number of Elements

The number of elements in $\mathbb{F}_p^\times$ is $p - 1$.

Primitive Root

Theorem

*Multiplicative group of a finite field $\mathbb{F}_p^\times$ is **cyclic**. The generators ω of this group are called **primitive roots**.*

Example

$\omega = 3$ is the primitive root of $\mathbb{F}_7$. Indeed,

$$3^1 = 3, \quad 3^2 = 2, \quad 3^3 = 6, \quad 3^4 = 4, \quad 3^5 = 5, \quad 3^6 = 1.$$

Clearly, $\langle \omega \rangle = \mathbb{F}_7^\times$.

The set $\mathbb{F}_p^\times$ is not useful on its own. However, we can consider the following set, called r**-th roots of unity**:

$$\Omega_r = \{\omega \in \mathbb{F}_p^\times \mid \omega^r = 1\} \subset \mathbb{F}_p^\times.$$

Question. When such cyclic group exists?

Roots of Unity

Theorem (Lagrange Theorem)

If $\mathbb{H} \leq \mathbb{G}$ is a subgroup of any finite group $\mathbb{G}$, then $ord(\mathbb{H}) \mid ord(\mathbb{G})$.

Corollary

If Ω_r is a subgroup of $\mathbb{F}_p^\times$, then $r \mid (p-1)$.

Some other Notes

Moreover, one might prove in the opposite direction:

- If $r \mid (p-1)$, then there exists a subgroup $\Omega_r \leq \mathbb{F}_p^\times$.
- Its generator is given by $\omega = g^{(p-1)/r}$ where $\langle g \rangle = \mathbb{F}_p^\times$.

Yet another note

Typically, we would need r to be the power of two. We will see why in the NTT section.

Complex Analysis Interpretation

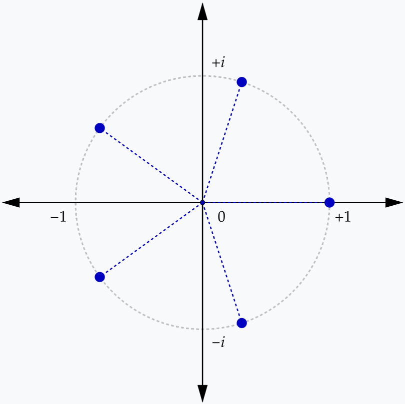

Figure: Visualization of the roots of unity $\Omega_5 = \{z \in \mathbb{C} : z^5 = 1\}$.

On the complex plane, the generator of the r-th roots of unity Ω_r is given by $\zeta_r = e^{2\pi i/r}$. In a finite field, we do not have such a luxury.

Vanishing Polynomial

Definition

The **vanishing polynomial** $z_D(x)$ of a set $D \subset \mathbb{F}_p$ is a polynomial satisfying $z_D(d) = 0$ for all $d \in D$.

Vanishing polynomials are always of form $z_D(x) = c \cdot \prod_{d \in D}(x - d)$.

The interesting question is: what is the vanishing polynomial of the r-th roots of unity Ω_r? For simplicity, assume $c = 1$.

Lemma

The vanishing polynomial of Ω_r is $z_\Omega(x) = x^r - 1$.

Proof Idea. Since for any $\zeta \in \Omega_r$ we have $\zeta^r = 1$, or, equivalently, $\zeta^r - 1$. Thus, any $\zeta \in \Omega_r$ is a root of $z_\Omega(x) = x^r - 1$.

Vanishing Polynomial over $\mathbb{R}$

Figure: Vanishing polynomial $p(x) = (x-1)(x-2)(x-4)$ of $D = \{1, 2, 4\}$

Barycentric Interpolation

Now, let us come back to the interpolation problem $p(x_j) = a_j$ for $j \in [N]$. Introduce $\gamma(x) = \prod_{j=0}^{N-1}(x - x_j)$.

Proposition

The Lagrange basis polynomial ℓ_j can be rewritten as:

$$\ell_j(x) = \gamma(x) \cdot \frac{w_j}{x - x_j}, \quad w_j = \frac{1}{\sum_{k=0, k \neq j}^{N-1}(x_j - x_k)}.$$

Let us substitute it into the interpolation formula:

$$p(x) = \sum_{j=0}^{N-1} a_j \ell_j(x) = \sum_{j=0}^{N-1} a_j \gamma(x) \frac{w_j}{x - x_j} = \gamma(x) \sum_{j=0}^{N-1} \frac{w_j}{x - x_j} a_j.$$

Barycentric Interpolation (Cont.)

$$\textbf{Barycentric Formula:}\ p(x) = \gamma(x) \sum_{j=0}^{N-1} \frac{w_j}{x - x_j} a_j$$

Proposition

- Computing $\{w_j\}_{j \in [N]}$ costs $\mathcal{O}(N^2)$ operations *before evaluation*.
- Both $\gamma(x)$ and sum requires $\mathcal{O}(N)$ operations.

But what happens if instead of x_j, we use $\omega^j \in \Omega_N$?

$$p(x) = \frac{x^N - 1}{N} \sum_{j \in [N]} \frac{\omega^j}{x - \omega^j} a_j$$

Takeaway: We can interpolate in $\mathcal{O}(N)$ operations.

Number Theoretic Transform

What is NTT?

Now suppose we want to find $m(x) = p(x)q(x)$. We'll use NTT!

Question

What does it mean that you *know* polynomial $p(x) \in \mathbb{F}^{(\leq N)}[x]$?

This means either of two (typically):

- You know the polynomial coefficients $p_0, \dots, p_{N-1}$.
- You know the polynomial values at some points $\{(x_j, a_j)\}_{j \in [N]}$.

Definition (NTT)

Suppose $p(x) = \sum_{j=0}^{N-1} p_j x^j$. The **Number Theoretic Transform** (NTT) of p is defined as evaluations of p at the N-th roots of unity:

$$\mathrm{NTT}(p) = \left(p(\omega^0), p(\omega^1), \dots, p(\omega^{N-1})\right).$$

What is the point of NTT?

Note: To denote the result of NTT, we use hat: $\hat{p} = \mathsf{NTT}(p)$.

Question: Given NTTs $\hat{p}$ and $\hat{q}$ of two polynomials p and q, how do we find the NTT of their product $m(x) = p(x)q(x)$?

Main NTT Property

Suppose $m(x) = p(x)q(x)$ is the product of p and q. Then,

$$\hat{\boldsymbol{m}} = \hat{\boldsymbol{p}} \odot \hat{\boldsymbol{q}}$$

Speaking more formally, $\mathsf{NTT} : (\mathbb{F}^{(\leq N)}[X], \times) \to (\mathbb{F}^N, \odot)$ is a homomorphism between a set of polynomials of degree up to N and their NTT domain. With certain appropriate technicalities, NTT can be extended to the isomorphism (namely, use $\mathbb{F}[X]/(X^N - 1)$).

Why? Well... $m(\omega^j) = p(\omega^j)q(\omega^j)$:/

Final Ingredient: Inverse NTT

Now, can we restore the polynomial $m(x)$ from its NTT $\hat{m}$? Of course!

Definition

Inverse NTT The **Inverse Number Theoretic Transform** (INTT) is a function that restores the polynomial $m(x)$ from its evaluations $\hat{m}$:

$$\mathrm{INTT}(\hat{m}) = (m_0, m_1, \dots, m_{N-1})$$

In its essence, we solve the interpolation problem:

$$m(\omega^j) = \hat{m}_j, \quad j \in [N], \quad \textbf{Goal}\text{: find coefficients } m_0, \dots, m_{N-1}$$

Punchline

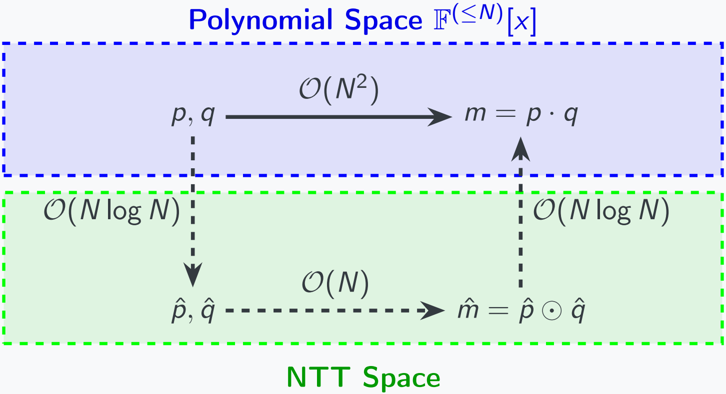

Figure: Illustration of the NTT Algorithm

Question

Does it resemble you one trick from Elliptic Curves?

Illustration

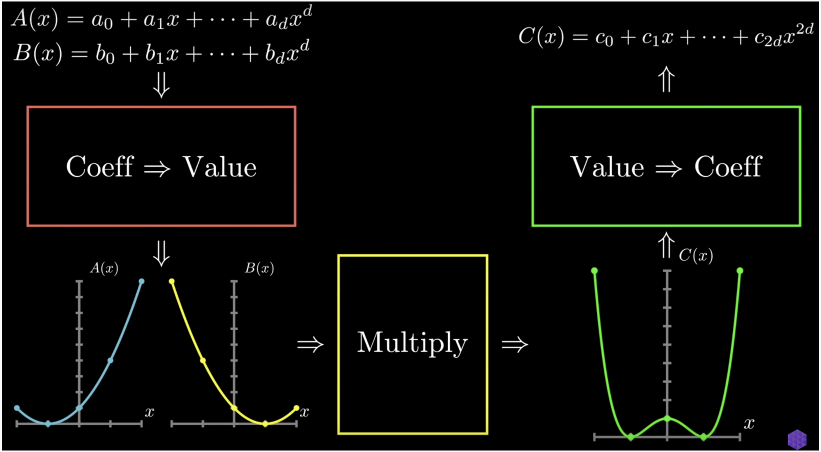

Figure: Illustration of the FFT Algorithm. Taken from "The Fast Fourier Transform (FFT): Most Ingenious Algorithm Ever?"

Details

When NTT works?

Note

For NTT to work, we will impose the following requirements on our setup:

1. The field $\mathbb{F}_p$ should have 2^k-roots of unity for sufficiently many k. In other words, $p = p' \cdot 2^m + 1$ with *small* $p' \in \mathbb{N}$.
2. The polynomial order is $N = 2^k$. Not a strict requirement, since we can always pad the polynomial with zeros.

Example

- **BabyBear prime** $p = 15 \cdot 2^{27} + 1$ is NTT-friendly: the order of multiplicative group is $15 \cdot 2^{27}$, so $2^k \mid 15 \cdot 2^{27}$ for all $k \leq 27$.
- **Mersenne prime** $p = 2^{31} - 1$ is not NTT-friendly: the order of multiplicative group is $2^{31} - 2 = 2 \times (2^{30} - 1)$.

Why NTT takes quasilinear complexity?

Recall that we need to evaluate N expressions:

$$p(\omega^j) = \sum_{i=0}^{N-1} p_i(\omega^j)^i = \sum_{i=0}^{N-1} p_i\omega^{ij}, \quad j \in [N].$$

Naive Complexity: $\mathcal{O}(N^2)$ operations. We need N evaluations, each of which requires N multiplications.

$$\begin{aligned} p(\omega^j) = \sum_{i=0}^{2^r-1} p_i\omega^{ij} &= \sum_{i=0}^{2^{r-1}-1} p_{2i}\omega^{2ij} + \sum_{i=0}^{2^{r-1}-1} p_{2i+1}\omega^{j(2i+1)} \\ &= \sum_{i=0}^{2^{r-1}-1} p_{2i}(\omega^{2j})^i + \omega^j \sum_{i=0}^{2^{r-1}-1} p_{2i+1}(\omega^{2j})^i. \end{aligned}$$

Folding

Denote $p_E(x) = \sum_{i=0}^{2^{r-1}-1} p_{2i}x^i$ and $p_O(x) = \sum_{i=0}^{2^{r-1}-1} p_{2i+1}x^i$.
Then,

$$p(\omega^j) = p_E(\omega^{2j}) + \omega^j p_O(\omega^{2j}).$$

Fact #1

We need only $N/2$ evaluations from Ω of p_E and p_O. Note that:

$$p(\omega^{j+N/2}) = p_E(\omega^{2j}) + \omega^j \omega^{N/2} p_O(\omega^{2j}).$$

Fact #2

- We need to evaluate two $N/2$-degree polynomials.
- We need to evaluate them at $N/2$ points.

Thus, we shrink the problem size by half at each step.

Algorithm Summarized

Algorithm 1: Number Theoretic Transform (NTT)

Input : Polynomial $p(x) = \sum_{j=0}^{N-1} p_j x^j$

Output : Vector of evaluations $\mathsf{NTT}(\boldsymbol{p}, \omega)$ at $\Omega = \{\omega\}_{j \in [N]}$

if $N = 1$ **then**

  Return : (p_0)

end

$H \leftarrow N/2$ /* Compute the domain half-size */

$\boldsymbol{p}_E \leftarrow (p_0, p_2, \ldots, p_{N-2})$ /* Find even-indexed coefficients */

$\boldsymbol{p}_O \leftarrow (p_1, p_3, \ldots, p_{N-1})$ /* Find odd-indexed coefficients */

$\boldsymbol{y}_E \leftarrow \mathsf{NTT}(\boldsymbol{p}_E, \omega^2)$ /* Compute NTT for even polynomial via $\frac{N}{2}$th primitive root ω^2 */

$\boldsymbol{y}_O \leftarrow \mathsf{NTT}(\boldsymbol{p}_O, \omega^2)$ /* Compute NTT for odd polynomial via $\frac{N}{2}$th primitive root ω^2 */

Return : $(y_0, \ldots, y_{N-1})$ with $y_j = y_{E, j \bmod H} + \omega^j y_{O, j \bmod H}$

Inverse NTT

Theorem

The Inverse NTT can be computed in the same way as NTT, but with the inverse primitive root ω^{-1}*:*

$$p_j = \frac{1}{N} \sum_{i=0}^{N-1} \omega^{-ij} \hat{p}_i$$

Thus, its complexity is also $\mathcal{O}(N \log N)$*.*

Conclusion

To compute $m(x) = p(x)q(x)$, simply use the following:

$$m(x) = \mathrm{INTT}(\mathrm{NTT}(p) \odot \mathrm{NTT}(q))$$

The total complexity remains $\mathcal{O}(N \log N)$.

Thank you for your attention

♥

zkdl-camp.github.io
github.com/ZKDL-Camp