NTT 201 - Foundations of NTT Hardware Design

Yuval Domb
yuval@ingonyama.com

Chapter 1

1.1 Introduction

NTT, or Number Theoretic Transform. is the term used to describe a Discrete Fourier Transform (DFT) over finite fields. In general, there's nothing unique about NTT that does not apply to DFT and visa-versa. However, the uses of NTT, its typical operating point, and some subtleties make it a very different beast. This note is twofold. In this chapter, we describe NTT from the use-case point-of-view, emphasizing what it is good for and what it is not. In the next chapter, we show how an NTT of a specific operating point can be optimally implemented in hardware.

1.2 From FT to DFT

1.2.1 FT

Fourier Transform (FT) [1] converts a continuous signal[1] from the time domain to the frequency domain and is defined as

$$X(\varepsilon) = \int_{-\infty}^{\infty} x(t) e^{-i2\pi\varepsilon t} dt \tag{1.1}$$

FT exists under complicated necessary conditions, but a generally sufficient condition is that $x(t) \in \mathcal{L}_1$ where $\mathcal{L}_1$ is the set of all absolutely integrable functions.[2] Other than that, $x(t)$ is an infinite, non-periodic function of time. FT is a unitary linear transformation and obeys Parseval's theorem [2]. One can think of $X(\varepsilon_0)$ as the quantity of signal energy at frequency ε_0 since it is exactly the correlation between the signal and the discrete tone $e^{-i2\pi\varepsilon_0 t}$.

1.2.2 DTFT

The Discrete-Time FT (DTFT) [3] is an FT for discrete-time signals and is defined as

$$X(f) = \sum_{-\infty}^{\infty} x[n] e^{-i2\pi f n} \tag{1.2}$$

[1] We tend to refer to the time-domain function as a signal in the context of FT.

[2] a.k.a. Lebesgue integrable functions.

The sampled discrete tone $e^{-i2\pi fn}$ is periodic, leading to a periodic frequency-domain function $X(f)$ with maximum frequency at $f = 0.5$. The DTFT can be thought of as a folded version of FT where the domain $\varepsilon \in (-\infty, \infty)$ is folded onto $f \in (-0.5, 0.5]$. DTFT can still be thought of in the sense of signal energy per frequency, but now the frequency f contains the accumulated energy of $X(\varepsilon)$ at frequencies $\{..., \varepsilon-2, \varepsilon-1, \varepsilon, \varepsilon+1, \varepsilon+2, ...\}$. The folding phenomenon in the frequency domain, often referred to as aliasing, is associated with time-domain sampling. In that sense, we can think of $x[n]$ as a sampled version of $x(t)$ (i.e. $x[n] = x(nT)$ where T is the sampling period).[3]

1.2.3 DFT

But what about periodic signals? On the one hand, a periodic signal has infinite energy. On the other hand, a single period is deterministically sufficient to represent it. So how about a transformation that represents a single period of a periodic signal? As it turns out, this is possible. For continuous-time periodic signals, this is called a Fourier Series (FS) [4]. For discrete-time periodic signals, it is called a DFT [5] and is defined as

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-i2\pi \frac{k}{N} n} \tag{1.3}$$

The frequency equivalent for DFT is $\frac{k}{N}$ for $k \in [0, N-1]$. DFT can be viewed as a normalized DTFT of a periodic signal. It is for that reason that for DFT we typically refer to signal power rather than energy.

The following table summarises the relationship between different transforms:

	FT	DTFT	FS	DFT
Periodic TD (Sampled FD)			✓	✓
Periodic FD (Sampled TD)		✓		✓

Note that it may seem that DFT is finite-time and finite-frequency. In fact, that is not the case but it is rather periodic in both time and frequency. This, as we shall see, has much influence on some of its main properties.

1.3 Properties of DFT

The set of discrete tones used in the DFT can be rewritten as

$$e^{-i2\pi \frac{k}{N} n} = \left(e^{-i\frac{2\pi}{N}}\right)^{kn} = \omega^{kn} \tag{1.4}$$

where ω is termed the N'th root-of-unity, since N is the smallest integer such that $\omega^N = 1$. We can now rewrite the DFT in its generalized form

$$X[k] = \sum_{n=0}^{N-1} x[n] \omega^{kn} \tag{1.5}$$

where ω^{kn} is a generalized discrete tone.

[3] An exact comparison between FT and DTFT requires normalization by the sampling period $T = f^{-1}$.

Replacing $x[n]$ with coefficients x_n of and substituting z for ω^k, results in the z-transform of the sequence $\{x_n\}_{n=0}^{N-1}$

$$X(z) = \sum_{n=0}^{N-1} x_n z^n \tag{1.6}$$

which is simply a polynomial of order $N-1$ in z.[4] With that, we can think of the transform $X[k]$ as polynomial evaluations of $X(z)$ over the root-of-unity powers[5] (i.e. $X[k] = X(\omega^k)$). This perspective is extremely useful in understanding some of the most useful properties of DFT.

1.3.1 Convolution

Linear convolution of two length-N sequences f_n and g_n is defined as

$$(f * g)_n = \sum_{m=0}^{N-1} f_m g_{n-m} \tag{1.7}$$

where both sequences are set zero for all $n \notin [0, N-1]$. In the polynomial form, this is described as

$$\begin{aligned}
(F \cdot G)(z) &= F(z)G(z) && (1.8)\\
&= \sum_{m=0}^{N-1} f_m z^m \cdot \sum_{k=0}^{N-1} g_k z^k && (1.9)\\
&= \sum_{m=0}^{N-1} \sum_{k=0}^{N-1} f_m \cdot g_k z^{m+k} && (1.10)\\
&= \sum_{m=0}^{N-1} \sum_{n=m}^{m+N-1} f_m \cdot g_{n-m} z^n && (1.11)\\
&= \sum_{n=0}^{2N-1} \left(\sum_{m=0}^{N-1} f_m \cdot g_{n-m} \right) z^n && (1.12)
\end{aligned}$$

where equality (1.12) is obtained by using the previous definition that $g_n \equiv 0$ for all $n \notin [0, N-1]$. Note that the term in parenthesis is equivalent to our previous definition of linear convolution (1.7).

So how can we use this? Imagine you have two length-N sequences and you would like to calculate their linear convolution of length $2N-1$, or you have coefficients of two degree-$(N-1)$ univariate polynomials and you want to calculate the coefficients of their product polynomial of degree-$(2N-2)$. The above claims that the point-wise multiplication of two polynomials' evaluations results in the product polynomial. The nice thing is that the product of evaluations can be used to calculate the corresponding coefficients by Inverse DFT (IDFT).[6] This is nearly sufficient and only missing one crucial ingredient. This

[4] The Region of Convergence (RoC) is obviously $z \leqslant 1$ so it consists of $[\omega]$

[5] Note how the root-of-unity powers are an equidistant partition of the unit circle.

[6] DFT is a unitary transformation and is thus bijective (i.e. invertible). This property is discussed hereafter.

ingredient is an outcome of the *Fundamental Theorem of Algebra* [6] which states that any polynomial of degree-$(L-1)$ is uniquely defined by any L distinct evaluations. This means that any $2N-1$ evaluations of the product polynomial $(F \cdot G)(z)$ suffice to uniquely define the coefficients $(f * g)_n$. With that, let us define the following convolution algorithm for two sequences of length-N

$$(f * g)_{n=0}^{2N-1} = \mathrm{IDFT}_{2N-1}\left(\mathrm{DFT}_{2N-1}(f_n) \circ \mathrm{DFT}_{2N-1}(g_n)\right) \tag{1.13}$$

where $\circ$ denotes the Hadamard element-wise product, subscript $\square_{2N-1}$ depicts a transform's length, and all sequences are assumed to be zero-padded to the required length as needed. The advantage of the r.h.s. of (1.13), as discussed later in this work, is that its typical computational complexity is $\mathcal{O}(N \log N)$, whereas the complexity of the naive linear convolution is $\mathcal{O}(N^2)$.

Finally, one could ask what would result from the following algorithm with all transforms shortened to length N

$$(f \circledast g)_{n=0}^{N-1} = \mathrm{IDFT}_N\left(\mathrm{DFT}_N(f_n) \circ \mathrm{DFT}_N(g_n)\right) \tag{1.14}$$

As it turns out, and somewhat hinted by the l.h.s. notation of (1.14), the above is an algorithm for calculating the cyclic convolution of two length-N sequences or the polynomial product of two polynomials in a polynomial ring $P[z]/(z^N-1)$. To see why this is so let us reexamine the polynomial product in the context of the ring $P[z]/(z^N-1)$ (i.e. $F(z)G(z)/(z^N-1)$). Clearly, the modulo restriction adds the cyclic constraint $z^{k+N} = z^k$ which when plugged into (1.12) leads to the following

$$\begin{aligned}
F(z)G(z)/(z^N-1) &= \sum_{n=0}^{2N-1}\left(\sum_{m=0}^{N-1} f_m \cdot g_{n-m}\right) z^n && (1.15)\\
&= \sum_{n=0}^{N-1}\left(\sum_{m=0}^{N-1} f_m \cdot g_{n-m}\right) z^n + \sum_{n=N}^{2N-1}\left(\sum_{m=0}^{N-1} f_m \cdot g_{n-m}\right) z^n && (1.16)\\
&= \sum_{n=0}^{N-1}\left(\sum_{m=0}^{N-1} f_m \cdot g_{n-m}\right) z^n + \left(\sum_{m=0}^{N-1} f_m \cdot g_{n-m+N}\right) z^n && (1.17)\\
&= \sum_{n=0}^{N-1}\left(\sum_{m=0}^{N-1} f_m \cdot (g_{n-m} + g_{n-m+N})\right) z^n && (1.18)\\
&= \sum_{n=0}^{N-1}\left(\sum_{m=0}^{N-1} f_m \cdot \widetilde{g}_{n-m}\right) z^n && (1.19)
\end{aligned}$$

where $\widetilde{g}_n$ is the N-periodic concatenation of g_n. The resulting polynomial (1.19) is of maximum degree $N-1$ and is thus uniquely defined by N distinct evaluations of the product $F(z)G(z)$ which immediately leads to algorithm (1.14).

1.3.2 Interpolation

Interpolation is the process of calculating the value of an evaluation of a polynomial for some point in its domain. Since the polynomial coefficients (or time domain signal) are well-defined, the interpolated value is unique. For a single sample, this can be achieved

by directly evaluating the polynomial from its coefficients or by using a form of *Lagrange Interpolation* such as the *Barycentric Formula* [7] over its available evaluations. In both cases, this is achievable with computational complexity $\mathcal{O}(N)$. It is often the case in data processing that we are interested in interpolating $L \geqslant N$ points located on a regularly-spaced grid. The naive implementation would require $\mathcal{O}(LN)$ complexity, but as it turns out, this can be performed by simple manipulation of the DFT, lowering the complexity to at most $\mathcal{O}(L \log L)$.

Suppose we have a sequence f_n of length N. One can easily use (1.6) to evaluate $F(z)$ anywhere in its domain. With a DFT of length N one can evaluate $\Omega = \{F(z) : z \in [\omega]\}$ where ω is the N'th root-of-unity and $[\omega]$ is the cyclic multiplicative subgroup generated by ω. With a DFT of length $L > N$ one can evaluate $\Upsilon = \{F(z) : z \in [\upsilon]\}$ where υ is the L'th root-of-unity. One may ask how a length-L DFT can be performed over the sequence f_n of length N. Since we treat the time domain sequence as polynomial coefficients, the only requirement is to zero-pad it to length-L and proceed as if it were a length-L sequence. This is equivalent to an order-$L-1$ univariate polynomial whose $L-N$ most significant monomials are zero. This leads to the following algorithm for arbitrary interpolation from subdomain $[\omega] \to [\upsilon]$

$$F([\upsilon]) = \mathrm{DFT}_L(\mathrm{IDFT}_N(F([\omega]))) \tag{1.20}$$

where $F([\omega]) \equiv \{F(z) : z \in [\omega]\}$ and zero-padding is assumed implicitly.

When $L = \lambda N$ for some $\lambda \in \mathbb{N}$ then $[\omega] = [\upsilon^\lambda] \leqslant [\upsilon]$ which means that

$$[\upsilon] = \bigcup_{l=0}^{\lambda-1} \upsilon^l[\omega] \tag{1.21}$$

where $\upsilon^l[\omega]$ are cosets of $[\omega]$. The key observation is that $F([\omega]) = F([\upsilon^\lambda]) \subset F([\upsilon])$ and it seems that in algorithm (1.20) the evaluations $F([\omega])$ would appear in both the input and output (i.e. some computational effort would be spent on recomputing $F([\omega])$). Can we somehow avoid this unnecessary effort? As it turns out, we can and this is particularly beneficial when λ is small (e.g. $\lambda = 2$ as is often the case). To see how this can be done, let us plug the k'th element from coset $\upsilon^l[\omega]$ into (1.6)

$$F(\upsilon^l\omega^k) = \sum_{n=0}^{N-1} f_n(\upsilon^l\omega^k)^n \tag{1.22}$$

$$= \sum_{n=0}^{N-1} \upsilon^{ln} f_n \omega^{kn} \tag{1.23}$$

In the context of the transform of length L (i.e. with υ as the root of unity) the above is exactly the formula for $F[\lambda k + l]$, the evaluations of coset l. This leads to the following algorithm for evaluating coset l from an original sequence of N evaluations

$$F(\upsilon^l[\omega]) = \mathrm{DFT}_N([\upsilon^l] \circ \mathrm{IDFT}_N(F([\omega]))) \tag{1.24}$$

where the Hadamard multiplication by $[\upsilon^l]$ is called modulation. Note how modulation of the coefficients by $[\upsilon^l]$ translates to a shift of the evaluations by υ^l (i.e. $F([\omega]) \to F(\upsilon^l[\omega])$). As an example, for $\lambda = 2$ the cost of interpolation reduces from a length-N IDFT and a length-$2N$ DFT to two length-N transforms.

Note that we used the frequency-shift/time-modulation property. The dual of this property (i.e. time-shift/frequency-modulation) holds in very much the same way and is extremely common in signal processing applications.

1.4 Fast DFT (FFT)

Fast Fourier Transform (FFT) is the common terminology for efficient methods for calculating the DFT. There are two unique methods.

The less common method, that can be shown to achieve near-linear complexity, is due to Winograd [8]. The problem with the Winograd method is that it is difficult to construct for arbitrary sizes. Constructions are known for fairly small-size transforms and as a result, it is not widely used. Although the Winograd FFT is utilized in Part 2 of this work, we will not explore it further here.

The common method for constructing FFT is typically associated with a 1965 paper due to Cooley and Tukey [9] even though it incorporates ideas that date back to Gauss as early as 1805 [10]. The method is often referred to as CT-FFT. The general idea is that when the transform length is a composite integer, it can be partitioned into smaller transforms whose results are combined to provide the desired calculation. Imagine we are interested in a length-N transformation where $N = LM$ with $L, M \in \mathbb{N}$. With CT-FFT the complexity can be reduced from $\mathcal{O}(N^2) = \mathcal{O}(L^2M^2)$ to approximately $\mathcal{O}(LM^2 + ML^2)$. The recursive application of the above method reduces the overall complexity to $\mathcal{O}(N \log N)$.

There are two main methods to construct recursive partitioning, Decimation-In-Time (DIT) and Decimation-In-Frequency (DIF). Both methods are essentially one and it can be shown that they both produce identical results up to permutation. Starting with the following DFT for a length-N sequence f_n

$$F(\omega^k) = \sum_{n=0}^{N-1} f_n \omega^{kn} \tag{1.25}$$

we can construct a DIT by partitioning the sequence to L sets $\left\{\{f_{mL+l}\}_{m=0}^{M-1}\right\}_{l=0}^{L-1}$ and proceeding to calculate the partial sums of (1.25) accordingly.[7]

$$\begin{aligned} F(\omega^k) &= \sum_{l=0}^{L-1} \sum_{m=0}^{M-1} f_{mL+l} \omega^{k(mL+l)} && (1.26)\\ &= \sum_{l=0}^{L-1} \omega^{kl} \sum_{m=0}^{M-1} f_{mL+l} (\omega^L)^{km} && (1.27)\\ &= \sum_{l=0}^{L-1} (\omega^M)^{ql} \omega^{\rho l} \sum_{m=0}^{M-1} f_{mL+l} (\omega^L)^{\rho m} && (1.28)\\ &= \sum_{l=0}^{L-1} \mu_L^{ql} \omega^{\rho l} \sum_{m=0}^{M-1} f_{mL+l} \cdot \mu_M^{\rho m} && (1.29) \end{aligned}$$

[7] The interleaved manner of the inner sets in the partition is often referred to as reverse-bit-ordering for DFTs of lengths that are powers of two.

$$= \sum_{l=0}^{L-1} \omega^{\rho l} \cdot \left(\sum_{m=0}^{M-1} f_{mL+l} \cdot \mu_M^{\rho m} \right) \mu_L^{ql} \tag{1.30}$$

where $\mu_L = \omega^M$ and $\mu_M = \omega^L$ are the L-th and M-th roots of unity respectively and $k = qM + \rho$ where $\rho < M$. Note that the internal sum in parenthesis represents L length-M DFTs. The outputs of these DFTs are multiplied element-wise by the twiddle-factors $\omega^{\rho l}$ and the products are fed into M length-L DFTs. An intuitive way to think about this algorithm is as follows:

1. Organise the sequence f_n of length N in an $L \times M$ matrix, where the construction is done column-wise.
2. Perform an independent length-M DFT per each row of the matrix.
3. Multiply all matrix elements such that element (l, ρ) is multiplied by the twiddle-factor $\omega^{\rho l}$.
4. Perform an independent length-L DFT per each column of the matrix.

The DIF alternative is to partition the transform $\{F(\omega^k)\}_{k=0}^{N-1}$ to $\left\{ \{F(\omega^{mL+l})\}_{m=0}^{M-1} \right\}_{l=0}^{L-1}$ and proceed to calculate the partial transforms. Recursing the FFT partitioning is easy and will not be discussed here. As mentioned above, CT-FFT partitioning only relies on the ability to factorize N to integral factors. As such, the finest partition is limited by the prime factors of N.[8] It is worth noting that DFTs with lengths that are powers of two are extremely popular in practice.

1.5 NTT

NTT is a DFT over a finite field and is typically used in cryptography. Although one can give a notion of frequency and power to the transform, it is typically used as a mathematical tool for efficiently operating with polynomials in polynomial rings. An NTT for a sequence will always exist, provided a root-of-unity of appropriate order exists. An interesting property of finite fields is that the elements of its multiplicative group all lie on the unit circle and are all roots of unity.

A question that often arises is, when is NTT a beneficial tool? In general, NTT should be used for efficiently performing convolutions and for interpolating many points that lie on regularly-spaced grids. As a rule of thumb, one should be concerned when the number of output values being calculated is smaller than the size of the NTTs being used. In those cases, there are usually more efficient methods for calculation that often do not require NTT at all.

1.5.1 Example: Groth16

Groth16 [11] is a Zero Knowledge Proving (ZKP) system [12]. As part of the system, the prover is required to calculate the following quotient

$$Q(x) = \frac{A(x)B(x) - C(x)}{x^N - 1} \tag{1.31}$$

[8] This is not the case for Winograd FFT.

where $A(x)$, $B(x)$, and $C(x)$ are of order $N-1$, and the denominator factorizes as

$$x^N - 1 = \prod_{n=0}^{N-1} (x - \omega^n) \tag{1.32}$$

where ω is an N'th root of unity. When the prover is honest, $x^N - 1$ divides the nominator, and $Q(x)$ is a polynomial of order $N-1$. The prover begins with the following evaluation sequences $A([\omega])$, $B([\omega])$, and $C([\omega])$. Since $[\omega]$ are all roots of both the nominator and denominator, simply plugging those evaluations in (1.31) would result in $\frac{0}{0}$ for evaluations of $Q([\omega])$ which is practically useless.

We may deduce that since the nominator is of maximum degree $2N-2$, we would be required to interpolate $A(x)$, $B(x)$, and $C(x)$ to at least $2N-1$ distinct evaluations before proceeding. As it turns out, we can do a little better by recognizing that the result polynomial $Q(x)$ is of order $N-1$. This means that N distinct evaluations suffice to represent it. An efficient method to get these N evaluations is by sampling all polynomials in N non-zero locations. This sampling is achievable by algorithm (1.24) using two length-N NTTs per source polynomial. Altogether the complexity is six length-N NTTs and some linear-time processing.

Bibliography

[1] Fourier transform. https://en.wikipedia.org/wiki/Fourier_transform.

[2] Parseval's theorem. https://en.wikipedia.org/wiki/Parseval%27s_theorem.

[3] Discrete-time fourier transform. https://en.wikipedia.org/wiki/Discrete-time_Fourier_transform.

[4] Fourier series. https://en.wikipedia.org/wiki/Fourier_series.

[5] Discrete fourier transform. https://en.wikipedia.org/wiki/Discrete_Fourier_transform.

[6] Fundamental theorem of algebra. https://en.wikipedia.org/wiki/Fundamental_theorem_of_algebra.

[7] Lagrange polynomial. https://en.wikipedia.org/wiki/Lagrange_polynomial.

[8] S. Winograd. On computing the discrete fourier transform. *Mathematics of Computation*, 32(141):175–199, 1978.

[9] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19:297–301, 1965. URL: http://cr.yp.to/bib/entries.html#1965/cooley.

[10] M. Heideman, D. Johnson, and C. Burrus. Gauss and the history of the fast fourier transform. *IEEE ASSP Magazine*, 1(4):14–21, 1984.

[11] Jens Groth. On the size of pairing-based non-interactive arguments. Cryptology ePrint Archive, Paper 2016/260, 2016. https://eprint.iacr.org/2016/260.

[12] Zero-knowledge proof. https://en.wikipedia.org/wiki/Zero-knowledge_proof.