Discrete Fourier Transform

Robert Hines

February 10, 2025

Contents

1 DFT

Let k be a field with a primitive nth root of unity ζ, and C_n the cyclic group of order n. The group ring is isomorphic to k^n (the Chinese remainder theorem)

$$k[C_n] = \frac{k[x]}{(x^n-1)} \cong \prod_{i=0}^{n-1} k[x]/(x-\zeta^i) \cong k^n$$

and the (D)iscrete (F)ourier (T)ransform realizes this isomorphism. From left to right, a polynomial $f = \sum_{i=0}^{n-1} f_i x^i$ of degree less than n can be evaluated at the roots of unity

$$f = (f_0, \ldots, f_{n-1}) \mapsto (f(\zeta^0), f(\zeta^1), \ldots, f(\zeta^{n-1}))) := \hat{f}$$

which is given by matrix multiplication against the coefficients of f,

$$\hat{f} = \left(\sum_i f_i(\zeta^0)^i), \ldots \sum_i f_i(\zeta^{n-1})^i \right) = \left(\zeta^{ij}\right)_{i,j=0}^{n-1} f.$$

The inverse to this evaluation is the interpolation problem: find the polynomial $f = \sum_{i=0}^{n-1} f_i x^i$ that takes the values $\hat{f}_i$ at the roots of unity ζ^i. More generally, if a polynomial p of degree $k-1$ takes the values y_i at distinct x_i then the coefficients of p are given by the unique solution to the linear system

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^{k-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{k-1} & x_{k-1}^2 & \ldots & x_{k-1}^{k-1} \end{pmatrix} \begin{pmatrix} p_0 \\ \vdots \\ p_{k-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ \vdots \\ y_{k-1} \end{pmatrix}.$$

When the x_i are the nth roots of unity, the matrix above takes the form $(\zeta^{ij})_{i,j=0}^{n-1}$, i.e. the DFT. The inverse of this matrix is $\frac{1}{n}(\zeta^{-ij})_{i,j=0}^{n-1}$ (exercise), giving the inverse DFT and solving the interpolation problem, giving the unique polynomial f of degree $n-1$ taking the values $\hat{f}_i$ at ζ^i.

In summary

$$\text{DFT} : f \xrightarrow{\text{evaluation at } \{\zeta^i\}} \hat{f}$$

$$\text{Inverse DFT} : \hat{f} \xrightarrow{\text{interpolation over } \{\zeta^i\}} f.$$

Algebraically, we linearly exchange convolution (polynomial muliplication modulo $x^n - 1$) for pointwise multiplication via the ring homomorphism above.

2 FFT

To speed up the discrete Fourier transform, we can work recursively (especially with $n = 2^N > 1$) to get the so-called (F)ast (F)ourier (T)ransform. There are various ways of performing the recursion. We outline two versions: decimation in time (DIT) and decimation in frequency (DIF).

2.1 Decimation in time

Thinking of f as a time series, we will repeatedly break f into to two pieces (hence "decimation in time"). Denote by $\mathcal{F}_n^{(i)}(f)$ the ith component of the DFT for $f \in k[C_n]$, and let $\zeta_n \in k$ be a primitive nth root of unity. Note that ζ_n^2 is a primitive $\frac{n}{2}$th root of unity.

We have, for $0 \leq i < n/2$:

$$\begin{aligned}
\mathcal{F}_n^{(i)}(f) &= \sum_{j=0}^{n-1} f_j (\zeta_n^i)^j \\
&= \sum_{j=2k} f_{2k} (\zeta_n^{2i})^k + \zeta_n^i \sum_{j=2k+1} f_{2k+1} (\zeta_n^{2i})^k \\
&= \mathcal{F}_{n/2}^{(i)}(f_{even}) + \zeta_n^i \mathcal{F}_{n/2}^{(i)}(f_{odd}).
\end{aligned}$$

For the second half of the coefficients, we have

$$\begin{aligned}
\mathcal{F}_n^{(i+n/2)}(f) &= \sum_{j=0}^{n-1} f_j (\zeta_n^{i+n/2})^j \\
&= \sum_{j=2k} f_{2k} (\zeta_n^{2i})^k - \zeta_n^i \sum_{j=2k+1} f_{2k+1} (\zeta_n^{2i})^k \\
&= \mathcal{F}_{n/2}^{(i)}(f_{even}) - \zeta_n^i \mathcal{F}_{n/2}^{(i)}(f_{odd}).
\end{aligned}$$

Above we used $\zeta_n^{n/2} = -1$ and $\zeta_n^{nk} = 1$. To summarize, we have the base case $\mathcal{F}_1(f) = f$ and

$$\mathcal{F}_n^{(i)}(f) = \begin{cases} \mathcal{F}_{n/2}^{(i)}(f_{even}) + \zeta_n^i \mathcal{F}_{n/2}^{(i)}(f_{odd}) & 0 \leq i < n/2 \\ \mathcal{F}_{n/2}^{(i-n/2)}(f_{even}) - \zeta_n^i \mathcal{F}_{n/2}^{(i-n/2)}(f_{odd}) & n/2 \leq i < n \end{cases}.$$

The "twiddle factors" $\zeta_{n/2^j}^i$, $0 \leq i < n/2^{j+1}$, can be precomputed to speed things up even further. Also, the recursion depth can be limited and a larger DFT base case can be used. The FFT has complexity $n \log n$ as opposed to the n^2 naive matrix multiplication of the DFT.

2.2 Decimation in frequency

Instead of considering even and odd indexed times, we consider even and odd indexed frequencies. Using the same notation as before, we have

$$\mathcal{F}_n^{(2i)}(f) = \sum_{j=0}^{n-1} f_j(\zeta_n^{2i})^j$$

$$\begin{aligned}
&= \sum_{j=0}^{n/2-1} f_j(\zeta_n^{2i})^j + \sum_{j=0}^{n/2-1} f_{j+n/2}(\zeta_n^{2i})^{j+n/2} \\
&= \sum_{j=0}^{n/2-1} (f_j + f_{j+n/2})(\zeta_n^{2i})^j \\
&= \mathcal{F}_{n/2}^{(i)}(f_{front} + f_{back})
\end{aligned}$$

and similarly

$$\mathcal{F}_n^{(2i+1)}(f) = \sum_{j=0}^{n-1} f_j(\zeta_n^{2i+1})^j$$

$$\begin{aligned}
&= \sum_{j=0}^{n/2-1} \zeta_n^j f_j(\zeta_n^{2i})^j + \sum_{j=0}^{n/2-1} \zeta_n^{j+n/2} f_{j+n/2}(\zeta_n^{2i})^{j+n/2} \\
&= \sum_{j=0}^{n/2-1} (f_j - f_{j+n/2})(\zeta_n^{2i})^j \\
&= \mathcal{F}_{n/2}^{(i)}((\zeta_n^j)_{j=0}^{n/2-1} \cdot (f_{front} - f_{back}))
\end{aligned}$$

with a pointwise product of $f_{front} - f_{back}$ with the first $n/2$ powers of ζ in the last line.

3 DIT FFT implementations: recursive, iterative, parallel

Here we look at pseudocode/circuits for recursive, iterative, and parallel implementations

3.1 Recursion

The recursion was outlined in the previous section. Below is some pseudocode. Managing powers of ζ could be done more efficiently, e.g. squaring and passing it along with each call.

3.2 Iteration

Instead of considering recursion through half-sized even/odd subproblems, we can start at the end with the 2×2 DFT and work backwards to obtain an iterative algorithm. If we want the output in natural order, we must start in bit-reversed order (which happens with the repeated

Algorithm 1 Recursive $FFT(f, i, n)$ ith coefficient of the DFT on nth roots of unity.

Note: ζ_n is a primitive nth root of unity, n a power of 2.

if $n = 1$ **then**
  return f
end if
$f_{even} = (f_0, f_2, \ldots, f_{n-2})$
$f_{odd} = (f_1, f_3, \ldots, f_{n-1})$
if $n > 1$ and $0 \leq i < n/2$ **then**
  return $FFT(f_{even}, n/2, i) + \zeta_n^i FFT(f_{odd}, n/2, i)$
end if
if $n > 1$ and $n/2 \leq i < n$ **then**
  return $FFT(f_{even}, n/2, i) - \zeta_n^i FFT(f_{odd}, n/2, i)$
end if

even/odd switcheroos), i.e. $br(f) = (f_{br(0)}, f_{br(1)}, \ldots, f_{br(n-1)})$ where $br(i)$ reads the $\log_2(n) - 1$ bit binary expansion of i backwards. E.g. when $n = 8$ we have

i	0	1	2	3	4	5	6	7
i_2	000	001	010	011	100	101	110	111
$br(i)_2$	000	100	010	110	001	101	011	111
$br(i)$	0	4	2	6	1	5	3	7

One way to think about this is as a matrix factorization of the DFT as a product of $\log_2(n) - 1$ block diagonal matrices, where the diagonal blocks at the ith stage are of the form

$$B_i = \begin{pmatrix} I & Z \\ I & -Z \end{pmatrix}, \; Z = \mathrm{diag}(\omega_i^k)_{k=0}^{2^i-1}, \; \omega_i = \zeta_n^{n/2^{i+1}}.$$

For example, when $n = 8$, we have $\hat{f} = (A_2 A_1 A_0) br(f)$ where the A_i are block diagonal with repeated diagonal blocks B_i

$$B_0 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \; B_1 = \left(\begin{array}{cc|cc} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & \omega_1 \\ \hline 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -\omega_1 \end{array}\right), \; B_2 = \left(\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & \omega_2 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & \omega_2^2 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & \omega_2^3 \\ \hline 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -\omega_2 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -\omega_2^2 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -\omega_2^3 \end{array}\right).$$

The pseudocode below has three for-loops: outermost for the depth (chain of As above), middle along the blocks of each A, and innermost for the Bs. (This order and other details could be changed around some.)

4 An application: polynomial multiplication

To multiply two polynomials $a(x)$, $b(x) \in k[x]$ whose degrees sum to less than n, we can use convolution in $k[C_n]$, i.e. the reduction modulo $x^n - 1$ won't affect the product ab. The naive

Algorithm 2 Iterative $FFT(f)$ on nth roots of unity (n a power of 2).

Note: ζ_n is a primitive nth root of unity, $\omega_i = \zeta_n^{n/2^{i+1}}$.

$f \leftarrow br(f)$ ▷ Bit-reversal
$m = 1$ ▷ Initialization, doubles in the outer loop
for $i = 0$ to $\log_2(n) - 1$ **do**
  $M = 2 \cdot m$
  for $j = 0$ to n by M **do**
    $\omega = 1$ ▷ Diagonal initialization, upper left
    for $k = 0$ to $m - 1$ **do**
      $a = f_{j+k}$
      $b = \omega \cdot f_{j+k+m}$
      $f_{j+k} \leftarrow a + b$
      $f_{j+k+m} \leftarrow a - b$
      $\omega \leftarrow \omega_i \cdot \omega$ ▷ Increase power along diagonal
    end for
  end for
  $m = M$ ▷ Doubling
end for
return f ▷ $f = \hat{f}$ in natural order

multiplication sums deg(a)deg(b) partial products: if $ab = c$ then the kth coefficient of the product is $c_k = \sum_{i+j=k} a_i b_j$. We can instead convert this to n pointwise multiplications, at the cost of performing DFTs:

$$ab = DFT^{-1}(DFT(a)DFT(b)).$$

In other words, the DFT diagonalizes polynomial multiplication in $k[x]/(x^n - 1)$ (with some overhead), which can be used for regular polynomial multiplication and is asymptotically faster.