Fast Fourier Transform

1 DFT vs FFT

1.1 Discrete Fourier Transform (DFT)

$$\begin{bmatrix} P(\omega^0) \\ P(\omega^1) \\ P(\omega^2) \\ \vdots \\ P(\omega^{n-1}) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega^1 & \omega^2 & \dots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \dots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \dots & \omega^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

$$P(\vec{\omega}) = M(\omega)\vec{a}$$

Figure 1: Discrete Fourier Transform

DFT transforms a sequence of complex numbers (or real numbers in some applications) into another sequence of complex numbers. For example, as shown in the picture above, we want to transfer $n \times 1$ vector $\vec{a}$ to $\vec{P}$. Now the matrix $\vec{M}(\omega)$ in the image is called the **DFT matrix**, with element as:

$$M[i, j] = \omega^{ij} \tag{1}$$

Here, ω is the primitive n^{th} root of unity:

$$\omega = e^{-\frac{2\pi i}{n}} \tag{2}$$

The time complexity of DFT is $O(n^2)$.

1.2 Fast Fourier Transform (FFT)

FFT is Cooley and Tukey's fast polynomial evaluation based algorithm to compute the matrix vector product in $O(n \log n)$ time.

2 Polynomial Multiplication

2.1 Serial Case

Suppose we have $P(x)$ as a polynomial of degree $m-1$:

$$P(x) = a_0 + a_1 x + a_2 x^2 + ... + a_{m-1} x^{m-1} \tag{3}$$

And we have $Q(x)$ is a polynomial of degree $n-1$:

$$Q(x) = b_0 + b_1 x + b_2 x^2 + ... + b_{n-1} x^{n-1} \tag{4}$$

Now we want to compute:

$$R(x) = P(x) \times Q(x) \tag{5}$$

Then we have $R(x)$ as a polynomial of degree as $m+n-2$ and its $m+n-1$ coefficients $c_0, c_1, ..., c_{m+n-2}$ are given by:

$$c_i = \sum_{j=\max(0,i-n+1)}^{\min(i,m-1)} a_j \times b_{i-j} \tag{6}$$

And the time complexity is $O(mn)$.

2.2 Parallel Case

Here.