程序设计思路

DIT-FFT:

按时间抽取基 2 FFT 算法，主要基于以下公式：

$$\text{原序列 x[n],定义}\begin{cases} e[n] = x[2n] \\ f[n] = x[2n+1] \end{cases}, \quad 0 \le n \le \frac{N}{2} - 1$$

$$\Rightarrow \begin{cases} X[k] = E[k] + W_N^k F[k] \\ X\left[k + \frac{N}{2}\right] = E[k] - W_N^k F[k] \end{cases}, 0 \le k \le \frac{N}{2} - 1$$

此时可将 N 个点的 DFT,分解成 2 次 N/2 个点的 DFT。递归的往下分解，即可实现 DIT-FFT 算法。

程序中未使用递归，由循环实现。首先编写函数 invert，将原信号转化为**倒位序**。

同时编写 DIT 下的蝶形运算函数 dit_butterfly，核心代码如下，该函数可实现同址运算。

```
void dit_butterfly(complex<double> &x1, complex<double> &x2, complex<double>
&rotation_factor, complex<double> &y1, complex<double> &y2) {
    complex<double> t = x2 * rotation_factor;
    y2 = x1 - t;
    //可同址运算
    y1 = x1 + t;
}
```

首先得到 N/2 个 2 点 DFT 的结果，再逐渐往上“合并”，得到 4，8，…，完整 N 个点的 DFT。核心代码如下：

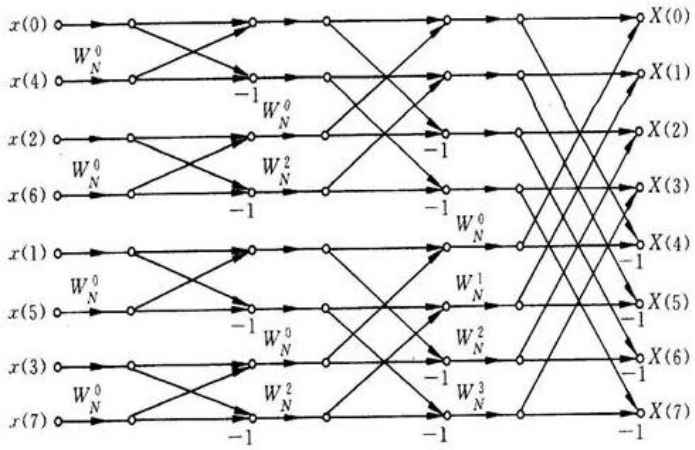
```
    //获得倒位序
    invert(xn, n, Xk);

    //m层
    for (int i = 0; i < m; i++) {
        int N = 1 << (i + 1);
        complex<double> Wn(cos(-2 * PI / N), sin(-2 * PI / N));

        for (int j = 0; j < n / N; j++) {
            //旋转因子
            complex<double> Wr = 1.0;

            for (int k = 0; k < N / 2; k++) {
                //同址运算
                dit_butterfly(Xk[j*N + k], Xk[j*N + k + N / 2], Wr, Xk[j*N + k],
Xk[j*N + k + N / 2]);
                Wr *= Wn;
            }
        }
    }
```

以 8 个点的 DFT 为例，分解流图如下：



DIF-FFT:

按频率抽取基 2 FFT 算法主要基于以下公式：

$$\text{定义}\begin{cases} e[n] = x[n] + x[n+\frac{N}{2}] \\ f[n] = [x[n] - x[n+\frac{N}{2}]]W_N^n \end{cases}, \quad 0 \le n \le \frac{N}{2} - 1$$

$$\Rightarrow \begin{cases} X[2r] = E[r] \\ X[2r+1] = F[r]' \end{cases} \quad 0 \le r \le \frac{N}{2} - 1$$

同样的，此时可将 N 个点的 DFT,分解成 2 次 N/2 个点的 DFT。递归的往下分解，即可实现 DIT-FFT 算法。

程序中未使用递归，由循环实现。

蝶形运算单元核心代码如下：

```
void dif_butterfly(complex<double> &x1, complex<double> &x2, complex<double>
&rotation_factor, complex<double> &y1, complex<double> &y2) {
    //临时变量，用于同址运算
    complex<double> t = x1;
    y1 = t + x2;
    y2 = (t - x2) * rotation_factor;
```

DIT-FFT 算法核心代码如下：

```
    complex<double> *inverted_Xk = new complex<double>[n];
    for (int i = 0; i < n; i++) {
        inverted_Xk[i] = xn[i];
    }

    for (int i = 0; i < m; i++) {
        int N = n >> i;
        complex<double> Wn(cos(-2 * PI / N), sin(-2 * PI / N));
```

```
        for (int j = 0; j < n / N; j++) {
            //旋转因子
            complex<double> Wr = 1.0;

            for (int k = 0; k < N / 2; k++) {
                //同址运算
                dif_butterfly(inverted_Xk[j*N + k], inverted_Xk[j*N + k + N/2], Wr,
inverted_Xk[j*N + k], inverted_Xk[j*N + k + N/2]);
                Wr *= Wn;
            }
        }
    }

    //倒位序操作
    invert(inverted_Xk, n, Xk);
```

运行结果

分别对 L = 2^10,11,12,13,14,15,16 的序列 x[n]做 DFT 与 FFT.

在不影响结果的情况下，我选择了全 1 序列作为输入，并输出了 FFT 结果的前三个分量。运行结果如下（时间以秒为单位）：

N = 2048:
DFT time cost: 1.285, X[0], X[1], X[3]: (2048,0), (-3.84487e-07,-5.6831e-10), (-3.84485e-07,-1.14172e-09)
DIT-FFT time cost: 0.005, X[0], X[1], X[3]: (2048,0), (0,0), (0,0)
DIT-FFT time cost: 0.007, X[0], X[1], X[3]: (2048,0), (0,0), (0,0)

N = 4096:
DFT time cost: 5.074, X[0], X[1], X[3]: (4096,0), (-7.68968e-07,-6.62056e-10), (-7.68966e-07,-1.2901e-09)
DIT-FFT time cost: 0.012, X[0], X[1], X[3]: (4096,0), (0,0), (0,0)
DIT-FFT time cost: 0.013, X[0], X[1], X[3]: (4096,0), (0,0), (0,0)

N = 8192:
DFT time cost: 20.234, X[0], X[1], X[3]: (8192,0), (-1.53795e-06,-5.16295e-10), (-1.53794e-06,-1.32201e-09)
DIT-FFT time cost: 0.026, X[0], X[1], X[3]: (8192,0), (0,0), (0,0)
DIT-FFT time cost: 0.027, X[0], X[1], X[3]: (8192,0), (0,0), (0,0)

N = 16384:
DFT time cost: 80.974, X[0], X[1], X[3]: (16384,0), (-3.07589e-06,-2.02522e-11), (-3.07592e-

06,-1.02677e-09)
DIT-FFT time cost: 0.056, X[0], X[1], X[3]: (16384,0), (0,0), (0,0)
DIT-FFT time cost: 0.059, X[0], X[1], X[3]: (16384,0), (0,0), (0,0)

N = 32768:
DFT time cost: 323.783, X[0], X[1], X[3]: (32768,0), (-6.15182e-06,5.05221e-09), (-6.15178e-06,-1.09848e-10)
DIT-FFT time cost: 0.124, X[0], X[1], X[3]: (32768,0), (0,0), (0,0)
DIT-FFT time cost: 0.125, X[0], X[1], X[3]: (32768,0), (0,0), (0,0)

N = 65536:
DFT time cost: 1300.44, X[0], X[1], X[3]: (65536,0), (-1.23035e-05,1.24504e-08), (-1.23036e-05,1.0251e-08)
DIT-FFT time cost: 0.27, X[0], X[1], X[3]: (65536,0), (0,0), (0,0)
DIT-FFT time cost: 0.266, X[0], X[1], X[3]: (65536,0), (0,0), (0,0)

N = 131072:
DFT time cost: 5281.37, X[0], X[1], X[3]: (131072,0), (-2.46037e-05,2.50001e-08), (-2.46034e-05,1.03041e-08)
DIT-FFT time cost: 0.54, X[0], X[1], X[3]: (131072,0), (0,0), (0,0)
DIT-FFT time cost: 0.54, X[0], X[1], X[3]: (131072,0), (0,0), (0,0)

可以发现：直接 DFT，DIT-FFT，DIF-FFT 的输出结果相同（排除数值计算的误差后），但 DFT 的运算时间 >> FFT 的运算时间。

可以看到，N 每增大 2 倍，DFT 的运算时间增加 4 倍，而 FFT 的时间仅仅增加 1 倍。特别是当 N >= 32768 时，DFT 的运算时间长达数分钟至一个多小时，而 FFT 仅需执行不到 1 秒，差别极大。由此可见，FFT 在 N 很大时的效率远高于直接 FFT。