Decimation-in-time (DIT) Radix-2 FFT

Abstract

Decimation-in-time (DIT) Radix-2 FFT

\[ X(k) = \sum_{n=0}^{N-1} x(n) e^{-\frac{i 2\pi nk}{N}} \]

\[ = \sum_{n=0}^{\frac{N}{2}-1} x(2n) e^{-\frac{i 2\pi (2n)k}{N}} + \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) e^{-\frac{i 2\pi (2n+1)k}{N}} \]

\[ = \sum_{n=0}^{\frac{N}{2}-1} x(2n) e^{-\frac{i \pi nk}{2}} + e^{-i \frac{\pi k}{2}} \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) e^{-\frac{i \pi nk}{2}} \]

\[ = \text{DFT}_{\frac{N}{2}}[[x(0), x(2), \ldots, x(N-2)]] + W_k^N \text{DFT}_{\frac{N}{2}}[[x(1), x(3), \ldots, x(N-1)]] \]

New Operation Counts

- \(2\left(\frac{N}{2}\right)^2 + N = \frac{N^2}{2} + N\) Complex Multiplies
- \(2 \frac{N}{2} \left(\frac{N}{2} - 1\right) + N = \frac{N^2}{2}\)

Reduction by almost a factor of two over direct DFT computation

1 Additional Simplification

A basic "butterfly" operation is shown in Figure 2, which results in only \(\frac{N}{2}\) "twiddle factor" multiplies per stage.

The same trick can be applied recursively to the two length \(\frac{N}{2}\) DFT's to save computation. The result is the radix-2 DIT FFT algorithm (Figure 3).

Computational cost of radix-2 DIT FFT

\(^*\text{http://creativecommons.org/licenses/by/1.0}\)
\(^1\text{http://cnx.rice.edu/content/m12032/latest/#DFTequation}\)
\(^2\text{http://cnx.rice.edu/content/12032/latest/#DFTequation}\)
Figure 1
Figure 2: Both operations are equivalent
\( M = \log_2 N \) stages; \( \frac{N}{2} \) butterflies/stage; 1 complex multiply, 2 adds/butterfly

\( \frac{N}{2} \log_2 N \) complex multiplies

\( N \log_2 N \) complex adds

Using special butterflies for \( W_N^0, W_N^N, W_N^N, W_N^N, W_N^{3N} \), the cost is

- \( 2N\log_2 N - 7N + 12 \) real multiplies
- \( 3N\log_2 N - 3N + 4 \) real additions

**NOTE:** Input is in "bit-reversed" order (hence "decimation-in-time"). That is, if \( n \) is written as a binary number, the order is that binary number reversed.

**Example 1: N=8**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>000</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
<td>010</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>011</td>
<td>110</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>001</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>101</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>101</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
<td>111</td>
<td>7</td>
</tr>
</tbody>
</table>
Example 2: Example FFT Code

/******************************************************************************
/* fft.c */
/* Douglas L. Jones */
/* University of Illinois at Urbana-Champaign */
/* January 19, 1992 */
/* */
/* fft: in-place radix-2 DIT DFT of a complex input */
/* */
/* input: */
/* n: length of FFT: must be a power of two */
/* m: n = 2**m */
/* input/output */
/* x: double array of length n with real part of data */
/* y: double array of length n with imag part of data */
/* */
/* Permission to copy and use this program is granted */
/* as long as this header is included. */
/******************************************************************************
fft(n,m,x,y)
int n,,m;
double x[],y[];
{
int i,,j,,k,,n1,,n2;
double c,,s,,e,,a,,t1,,t2;

j = 0; /* bit-reverse */
n2 = n/2;
for (i=1; i < n - 1; i++)
{
    n1 = n2;
    while ( j > = n1 )
    {
        j = j - n1;
        n1 = n1/2;
    }
    j = j + n1;

    if (i < j)
    {
        t1 = x[i];
x[i] = x[j];
x[j] = t1;
        t1 = y[i];
y[i] = y[j];
y[j] = t1;
    }
}
n1 = 0; /* FFT */
n2 = 1;

for (i=0; i < m; i++)
{
    n1 = n2;
    n2 = n2 + n2;
    e = -6.283185307179586/n2;
    a = 0.0;

    for (j=0; j < n1; j++)
    {
        c = cos(a);
        s = sin(a);
        a = a + e;

        for (k=j; k < n; k=k+n2)
        {
            t1 = c*x[k+n1] - s*y[k+n1];
            t2 = s*x[k+n1] + c*y[k+n1];
            x[k+n1] = x[k] - t1;
            y[k+n1] = y[k] - t2;
            x[k] = x[k] + t1;
            y[k] = y[k] + t2;
        }
    }
}

return;

Comments

- Simple, short, elegant
- Three-loop structure
- For efficiency, a cosine-sine lookup table should be added