<u>利用 FFT 计算 LC3 1.0 式 86 的方法</u>

起草：季文骢，版本：6

<u>第一部分：概述</u>

式 86 如下（只保留必要的数学记号，其中 $k_{min} \le k \le k_{max}$，$L > 0$，k、$k_{min}$、$k_{max}$ 和 L 均为整数，x(n) 为实序列）：

$$R(k) = \sum_{n=0}^{L-1} x(n)x(n-k)$$

为方便起见定义 $k_{width} = k_{max} - k_{min} + 1$，直接按公式进行计算需要进行 $L \cdot k_{width}$ 次迭代，按 $k_{min} = 17$、$k_{max} = 114$、$\max(L) = 64$ 来估计，最差的情况需要 6272 次迭代。因此需要引入快速求解算法以降低迭代次数，下面将给出一种这样的算法及其推导过程。

<u>第二部分：算法流程及正确性证明</u>

定义 $k_s = k - k_{min}$（$0 \le k_s < k_{width}$），$w(n) = x(-n - k_{min})$，构造序列 $S(k_s)$：

$$S(k_s) = \sum_{n=0}^{L-1} x(n)w(k_s - n)$$

那么：

$$\begin{aligned} S(k_s) &= \sum_{n=0}^{L-1} x(n)x(n - k_s - k_{min}) \\ &= \sum_{n=0}^{L-1} x(n)x(n-k) \\ &= R(k) \end{aligned}$$

因此有 $S(k - k_{min}) = R(k)$，也就是说序列 $R(k_{min} \dots k_{max})$ 和 $S(0 \dots k_{width} - 1)$ 是相同的，若要得到序列 $R(k_{min} \dots k_{max})$ 中各项的值，只需计算序列 $S(0 \dots k_{width} - 1)$ 中各项的值即可。

由于恒有 $L + k_{width} - 1 > 0$，故必能找到一个正整数 s，使得 $L + k_{width} - 1$ 夹在 2^{s-1} 和 2^s 之间，即：

$$2^{s-1} < L + k_{width} - 1 \le 2^s$$

显然 $s = \lceil \log_2(L + k_{width} - 1) \rceil$。

定义序列 $p_1(n)$、$p_2(n)$（其中 $0 \le n < 2^s$）：

$$p_1(n) = \begin{cases} x(n) & (0 \le n < L) \\ 0 & (L \le n < 2^s) \end{cases}$$

$$p_2(n) = \begin{cases} w(n) & (0 \le n < k_{width}) \\ 0 & \left(k_{width} \le n < 2^s - (L-1)\right) \\ w(n - 2^s) & (2^s - (L-1) \le n < 2^s) \end{cases}$$

定义序列 $p_1(n)$、$p_2(n)$ 的循环卷积 $P(u)$（其中 $0 \le u < 2^s$）并将其求和范围缩小至 $0 \le n < L$：

$$
\begin{aligned}
P(u) &= \sum_{n=0}^{2^s-1} p_1(n) p_2\big((u-n) \bmod 2^s\big) \\
&= \sum_{n=0}^{L-1} p_1(n) p_2\big((u-n) \bmod 2^s\big) + \sum_{n=L}^{2^s-1} p_1(n) p_2\big((u-n) \bmod 2^s\big) \\
&= \sum_{n=0}^{L-1} p_1(n) p_2\big((u-n) \bmod 2^s\big)
\end{aligned}
$$

考虑 $0 \le u < L-1$ 的情况：

$$
P(u) = \sum_{n=0}^{u} p_1(n) p_2(u-n) + \sum_{n=u+1}^{L-1} p_1(n) p_2(u-n+2^s)
$$

由于恒有 $n \le L-1 \le u+L-1$，因此 $u-n+2^s \ge 2^s-(L-1)$ 恒成立，故：

$$
\begin{aligned}
P(u) &= \sum_{n=0}^{u} x(n) w(u-n) + \sum_{n=u+1}^{L-1} x(n) w(u-n+2^s-2^s) \\
&= \sum_{n=0}^{u} x(n) w(u-n) + \sum_{n=u+1}^{L-1} x(n) w(u-n) \\
&= \sum_{n=0}^{L-1} x(n) w(u-n) \\
&= S(u)
\end{aligned}
$$

再考虑 $L-1 \le u < 2^s$ 的情况：

$$
\begin{aligned}
P(u) &= \sum_{n=0}^{L-1} p_1(n) p_2(u-n) \\
&= \sum_{n=0}^{L-1} x(n) w(u-n) \\
&= S(u)
\end{aligned}
$$

因此可得结论，对于任意 $0 \le u < 2^s$，总是有 $S(u) = P(u)$。

最后，根据循环卷积的离散傅立叶变换（DFT）的相关性质，有：

$$
\begin{aligned}
\{S(0 \dots 2^s-1)\} &= \{P(0 \dots 2^s-1)\} \\
&= \mathrm{IDFT}\big\{\mathrm{DFT}\{P(0 \dots 2^s-1)\}\big\} \\
&= \mathrm{IDFT}\left\{\mathrm{DFT}\left\{\sum_{n=0}^{2^s-1} p_1(n) p_2\big((u-n) \bmod 2^s\big) \qquad (u = 0, 1 \dots 2^s-1)\right\}\right\} \\
&= \mathrm{IDFT}\big\{\mathrm{DFT}\{p_1(0 \dots 2^s-1)\} \cdot \mathrm{DFT}\{p_2(0 \dots 2^s-1)\}\big\}
\end{aligned}
$$

综合以上推导过程，得到计算 $R(k)$ 的算法如下：

```
//  Find (2 ^ s):
sz_min = L + k_width - 1;
sz = 1
while (sz < sz_min) {
    sz *= 2;
}
```

```
// p1[0...2 ^ s - 1]:
for (n = 0; n < L; ++n) {
   p1[n] = x[n];
}
for (n = L; n < sz; ++n) {
   p1[n] = 0;
}

// p2[0... 2 ^ s - 1]:
for (n = 0; n < k_width; ++n) {
   p2[n] = x[-n - k_min];
}
for (n = k_width; n < sz - (L - 1); ++n) {
   p2[n] = 0;
}
for (n = sz - (L - 1); n < sz; ++n) {
   p2[n] = x[-(n - sz) - k_min];
}

// DFT{p1[0...2 ^ s - 1]}:
p1 = DFT(p1);

// DFT{p2[0...2 ^ s - 1]}:
p2 = DFT(p2);

// Convolve p1[0...2 ^ s - 1] with p2[0...2 ^ s - 1] in frequency domain:
for (n = 0; n < sz; ++n) {
   r[n] = p1[n] * p2[n];
}

// IDFT{DFT{p1[0...2 ^ s - 1]} * DFT{p2[0...2 ^ s - 1]}}:
r = IDFT(r);

// Output.
R[k_min...k_max] = r[0...k_width - 1];
```

算法中的离散傅立叶变换（DFT）过程可以通过快速傅立叶变换（FFT）来实现。

第三部分：基于三次 DFT 的变种算法

先考虑长度为 N 的任意复序列 $X(k = 0,1 \ldots N-1)$ 的 IDFT 序列 $x(n)$（$0 \le n < N$）：

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{j\frac{2\pi}{N}kn}$$

于是：

$$x(n) = \left(x^*(n)\right)^* = \left(\left(\frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{j\frac{2\pi}{N}kn}\right)^*\right)^* = \frac{1}{N}\left(\sum_{k=0}^{N-1} X^*(k)e^{-j\frac{2\pi}{N}kn}\right)^*$$

由此得到 IDFT 的一个性质：

$$\mathrm{IDFT}\{X(0 \ldots N)\} = \frac{1}{N}\left(\mathrm{DFT}\{X^*(0 \ldots N)\}\right)^*$$

在实际的实现中，可以将系数 $\frac{1}{N}$ 乘在 $X^*(k)$ 上以避免不必要的后处理，即：

$$\mathrm{IDFT}\{X(0 \ldots N)\} = \left(\mathrm{DFT}\left\{\frac{1}{N}X^*(0 \ldots N)\right\}\right)^*$$

根据这一性质及其它相关的离散傅立叶变换（DFT）性质，对 $\{S(0 \ldots 2^s-1)\}$ 进行变形：

$$\begin{aligned}\{S(0 \ldots 2^s-1)\} &= \mathrm{IDFT}\left\{\mathrm{DFT}\{p_1(0 \ldots 2^s-1)\} \cdot \mathrm{DFT}\{p_2(0 \ldots 2^s-1)\}\right\} \\ &= \frac{1}{N}\left(\mathrm{DFT}\left\{\left(\mathrm{DFT}\{p_1(0 \ldots 2^s-1)\} \cdot \mathrm{DFT}\{p_2(0 \ldots 2^s-1)\}\right)^*\right\}\right)^*\end{aligned}$$

$$= \frac{1}{N}\left(\mathrm{DFT}\left\{(\mathrm{DFT}\{p_1(0 \dots 2^s - 1)\})^* \cdot (\mathrm{DFT}\{p_2(0 \dots 2^s - 1)\})^*\right\}\right)^*$$

$$= \frac{1}{N}\Big(\mathrm{DFT}\Big\{(\mathrm{DFT}\{p_1(0 \dots 2^s - 1)\})^*$$

$$\cdot\, \mathrm{DFT}\left\{p_2^*\left((-n) \bmod 2^s\right) \quad (0 \le n < 2^s)\right\}\Big\}\Big)^*$$

注意，上述推导的最后一步只对 $(\mathrm{DFT}\{p_2(0 \dots 2^s - 1)\})^*$ 这一部分进行了变换，这是因为 $p_1(n)$ 的构造已足够简单而且构造的过程（假定按 n 从小到大的顺序构造）对 $x(n)$ 的访问也是顺序的（按 n 从小到大的顺序进行访问）。

考察序列 $p_2^*\left((-n) \bmod 2^s\right)$（$0 \le n < 2^s$），不妨定义 $p_3(n) = p_2^*\left((-n) \bmod 2^s\right)$：

$$\begin{aligned}
p_3(n) &= p_2^*\left((-n) \bmod 2^s\right) \\
&= \begin{cases} w(-n) & (0 \le n < L) \\ 0 & \left(L \le n < 2^s - (k_{width} - 1)\right) \\ w(2^s - n) & \left(2^s - (k_{width} - 1) \le n < 2^s\right) \end{cases} \\
&= \begin{cases} x(n - k_{min}) & (0 \le n < L) \\ 0 & \left(L \le n < 2^s - (k_{width} - 1)\right) \\ x(n - k_{min} - 2^s) & \left(2^s - (k_{width} - 1) \le n < 2^s\right) \end{cases}
\end{aligned}$$

不难发现，在 $p_3(n)$ 的构造过程中（假定按 n 从小到大的顺序构造），对 $x(n)$ 的访问在区间 $0 \le n < L$ 以及 $2^s - (k_{width} - 1) \le n < 2^s$ 内也是顺序的(按 n 从小到大的顺序进行访问)。

综合以上推导，可将算法伪代码修改为：

```
//  Find (2 ^ s):
sz_min = L + k_width - 1;
sz = 1
while (sz < sz_min) {
    sz *= 2;
}

//  p1[0...2 ^ s - 1], p3[0...2 ^ s - 1]:
for (n = 0; n < L; ++n) {
    p1[n] = x[n];
    p3[n] = x[n - k_min];
}
for (n = L; n < sz - (k_width - 1); ++n) {
    p1[n] = p3[n] = 0;
}
for (n = sz - (k_width - 1); n < sz; ++n) {
    p1[n] = 0;
    p3[n] = x[n - k_min - sz];
}

//  DFT{p1[0...2 ^ s - 1]}:
p1 = DFT(p1);

//  DFT{p3[0...2 ^ s - 1]}:
p3 = DFT(p3);

//  Convolve in frequency domain:
for (n = 0; n < sz; ++n) {
    r[n] = p1[n].conjugate() * p3[n] / sz;
}

//  IDFT:
r = DFT(r);

//  Output.
R[k_min...k_max] = r[0...k_width - 1];
```