FFT 仿真报告

吴彦成

2023 年 3 月 13 日

1 概述

本次仿真的最终目标是在 matlab 中完成实现一个 64 位的 FFT 的运算的函数。该函数的输入是 1×64 的向量，输出是 1×64 的向量。相较于 DFT 算法，该算法可以将乘法的使用次数从 64^2 降低到 64×8。

该算法的核心思想是运用原文中的公式 (3):

$$\begin{aligned} F(k) &= \sum_{n=0}^{63} x(n) W_{64}^{nk} \\ &= \sum_{n_2=0}^{7} W_8^{n_2 k_1} [W_{64}^{n_2 k_2} \sum_{n_1=0}^{7} x(8n_1 + n_2) W_8^{n_1 k_2}] \end{aligned}$$

其中，$F(k)$ 是 FFT 输出的 64 位向量的第 k 项，x 是 FFT 输入的 64 位原始数据向量，$W_N = e^{-2\pi i/n}$.

公式 (3) 利用三角函数的周期性，将公式第一行的 64 次运算简化为了第二行的 16 次运算。在后续的操作中，我们会进一步利用此特性将 16 次运算简化到 8 次。

2 进一步化简以及对公式错误的修正

为了实现进一步的化简，我们注意到形如:

$$\sum_{m=0}^{7} x(m) W_8^{mk}$$

的结构多次出现。

在原文公式 (4) 中也对此结构进行了描述，但有一些符号上的错误，具体如下：

$$\begin{aligned}\alpha(k) &= \sum_{m=0}^{7} x(m) W_8^{mk} \\ &= \left\{ \sum_{m=0}^{7} [x_{RE}(m)\cos(mk\pi/4) + x_{IM}(m)\sin(mk\pi/4)] \right. \\ &\quad \left. + j\sum_{m=0}^{7} [x_{RE}(m)\sin(mk\pi/4) - x_{IM}(m)\cos(mk\pi/4)] \right\}\end{aligned} \tag{4}$$

我们对其修正如下：

$$\begin{aligned}\alpha(k) &= \sum_{m=0}^{7} x(m) W^{mk} \\ &= \sum_{m=0}^{7} [x_{RE}(m)cos(\frac{mk\pi}{4}) + x_{IM}(m)sin(\frac{mk\pi}{4})] \\ &+ j\sum_{m=0}^{7} [-x_{RE}(m)sin(\frac{mk\pi}{4}) + x_{IM}(m)cos(\frac{mk\pi}{4})]\end{aligned}$$

利用三角函数的基本性质：

$$\sin(\pi+\phi) = -\sin(\phi), \text{ and } \cos(\pi+\phi) = -\cos(\phi) \tag{5}$$

文中对公式 (4) 进行了如下的进一步化简：

$$\begin{aligned}\alpha(k)_{RE} &= \sum_{m=0}^{3} [(x_{RE}(m) - x_{RE}(4+m))\cos(mk\pi/4) \\ &\quad + (x_{IM}(m) - x_{IM}(4+m))\sin(mk\pi/4)] \\ \alpha(k)_{IM} &= j\sum_{m=0}^{3} [(x_{RE}(m) - x_{RE}(4+m))\sin(mk\pi/4) \\ &\quad - (x_{IM}(m) - x_{IM}(4+m))\cos(mk\pi/4)]\end{aligned} \tag{6}$$

此番操作是在公式 (4) 错误的基础上再误认为 $cos((4+m)k\pi/4) = -cos(mk\pi/4)$ 对一切的 k 都成立。然而事实上，$cos((4+m)k\pi/4) = cos(mk\pi/4 + k\pi) = (-1)^k cos(mk\pi/4)$。我们对公式 (6) 修正如下：

$$\alpha(k)_{RE} = \sum_{m=0}^{3}[(x_{RE}(m) + (-1)^k x_{RE}(4+m))cos(\frac{mk\pi}{4}) \\ + (x_{IM}(m) + (-1)^k x_{IM}(4+m))sin(\frac{mk\pi}{4})]$$

$$\alpha(k)_{IM} = j\sum_{m=0}^{3}[-(x_{RE}(m) + (-1)^k x_{RE}(4+m))sin(\frac{mk\pi}{4}) \\ + (x_{IM}(m) + (-1)^k x_{IM}(4+m))cos(\frac{mk\pi}{4})]$$

通过修正后的 α 操作，我们将需要的乘法次数从原来的 8 次缩减到现在的 4 次。

3 仿真的流程

我们注意到在公式 (3) 中括号内的部分：

$$W_{64}^{n_2k_2}\sum_{n_1=0}^{7}x(8n_1+n_2)W_8^{n_1k_2}$$

只与 n_2 和 k_2 有关，并且当 n_2 和 k_2 遍历 $0\sim7$ 时，其也会遍历 64 个值，所以也可以把这一部分看作一个 64 位的中间向量。我们仿真的第一步就是要得到这个中间向量。

得到中间向量的方法其实就是遍历 n_2 和 k_2，对每一个确定的 n_2 和 k_2 通过上述公式计算对应的值。当然，对于

$$\sum_{n_1=0}^{7}x(8n_1+n_2)W_8^{n_1k_2}$$

我们可以利用之前讨论过的 α 操作将循环的次数从 8 次降到 4 次。这样一来，我们便顺利地得到了中间向量 y。

下一步我们便可以直接通过中间向量 y 得到最终的结果：

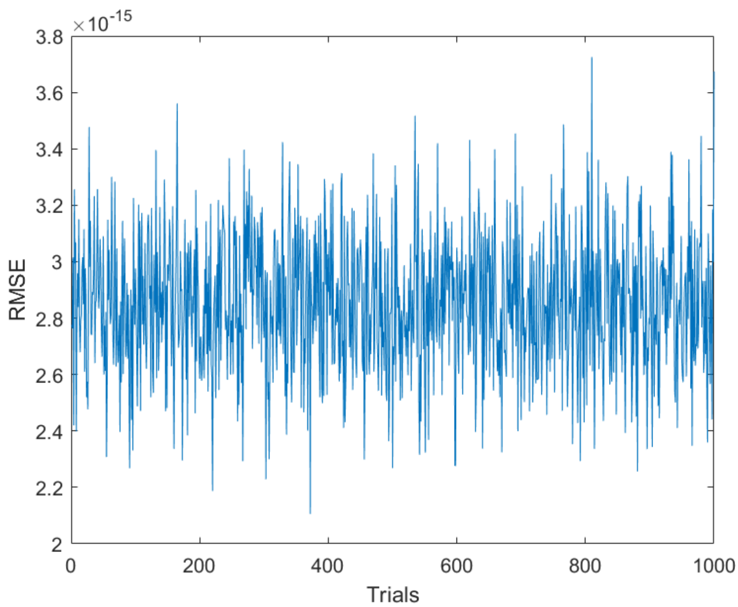
$$F(k) = \sum_{n2=0}^{7}W_8^{n_2k_1}y(8k_2+n_2)$$

注意到这一部分的和只与 k_1 和 k_2 有关，并且这样的结构同样可以使用 α 操作将循环的次数从 8 次降到 4 次。

至此我们便顺利得到了 FFT 的结果，一个 64 位的向量。

4 误差分析

为了验证仿真的正确性，我们用均方根进行了误差估计。具体的操作是每次生成一组 64 位随机数组成的向量作为函数的输入，将得到的结果与 matlab 内置的 FFT 得到的结果进行均方差估计，并重复 1000 次，最终结果如下：



如果将 1000 组 64 位随机向量视为一组 64000 位的数据，其最终的均方根误差为 2.8628×10^{-15}。误差在可接受范围之类。

5 总结

本次仿真成功地完成了对原文中除去 LUTs 以外核心算法的论证，为下一步工作奠定了基础。