arXiv:2603.29129v1 [cs.MS] 31 Mar 2026

使用低精度快速傅里叶变换计算目标精度快速傅里叶变换

SHOTA KAWAKAMI，筑波大学理工学研究科，日本DAISUKE TAKAHASHI，筑波大学计算科学中心，日本

现代处理器执行低精度算术的吞吐量高于执行高精度算术。对于矩阵乘法，Ozaki 方案通过将输入分割成低精度分量，并将计算交由经过优化的低精度例程处理，利用了这一性能差距。然而，快速傅里叶变换（FFT）尚无类似方法。本文提出一种方法，将 Ozaki 方案应用于布鲁斯坦快速傅里叶变换中的循环卷积，使用低精度快速傅里叶变换计算目标精度快速傅里叶变换。拆分分量卷积通过数论变换（NTT）精确计算；数论变换是在有限域上进行的快速傅里叶变换，并结合中国剩余定理，取代浮点快速傅里叶变换。我们给出了分割数的上界，并提出一种数论变换域累加策略，以减少 NTT 的调用次数。作为具体实现，我们使用 32 位数论变换实现双精度快速傅里叶变换，并证实，与基于 FFTW 和三单精度算术的快速傅里叶变换相比，该方法的相对误差更低，且在不同 FFT 长度下误差稳定；最多需要调用 96 次 NTT，采用数论变换域累加时则最多调用 64 次。在 Intel Xeon Platinum 8468 上，对于长度为 $n = 2^{10}$–2^{18}, 的变换，执行时间约为 FFTW 双精度快速傅里叶变换的 107–1315×，其中 NTT 占总时间的约 80%。

CCS概念： • **计算数学** → **变换计算**；**数学软件**。

附加关键词和短语：快速傅里叶变换、数论变换、布鲁斯坦快速傅里叶变换、Ozaki 方案、中国剩余定理

1 引言

快速傅里叶变换（FFT） [Cooley和Tukey 1965], 是一种用于计算离散傅里叶变换（DFT）的高效算法，广泛应用于信号处理、科学计算和任意精度算术。

现代处理器的低精度算术吞吐量通常远高于高精度算术，而且这一差距还在持续扩大。例如，从NVIDIA的H100到B200数据中心GPU，峰值双精度吞吐量仅从34 TFLOPS提高到37 TFLOPS，而半精度Tensor Core吞吐量则从990 TFLOPS提高到2.2 PFLOPS，8位Tensor Core吞吐量则从2.0 PFLOPS提高到4.5 PFLOPS [NVIDIA2023, 2024]。在近期的消费级GPU（例如NVIDIA的GeForce RTX 5090）上，双精度吞吐量仅为单精度吞吐量的1/64 [NVIDIA 2025b], , 因为图形工作负载很少需要双精度。此外，大多数处理器不提供双精度以上浮点算术或64位以上整数算术的硬件支持。因此，四倍精度（binary128）等高精度格式 [IEEE 2008], Double-Double和Quad-Double [Hida等人 2001], 以及MPFR等任意精度库 [Fousse等人 2007]和GMP [GNU 2023], 因此必须通过软件实现，性能会显著降低。

对于矩阵乘法，Ozaki 方案 [Ozaki 等人 2012] 为弥合这一精度–性能差距提供了一种可移植的解决方案：通过分裂输入，并将核心计算交由经过优化的低精度基础线性代数子程序（BLAS）例程执行，以目标精度计算矩阵-矩阵乘积。由于达到目标精度只需使用现有的优化基础线性代数子程序库，无需实现或优化新内核。NVIDIA 的

作者联系信息： Shota Kawakami，kawakami@hpcs.cs.tsukuba.ac.jp，筑波大学理工学研究科，筑波，茨城，日本； Daisuke Takahashi，daisuke@cs.tsukuba.ac.jp，计算科学中心，筑波大学，筑波，茨城，日本。

新一代 Rubin 架构利用 Ozaki 方案，在 R200 GPU 上实现了 200 TFLOPS 的双精度矩阵乘法性能，尽管其原生双精度吞吐量仅为 33 TFLOPS [NVIDIA 2025a]。鉴于双精度吞吐量增长停滞，而低精度吞吐量仍在持续提升，将这一方法从矩阵乘法扩展到其他基础运算是一个重要方向。

使用低精度运算以目标精度计算快速傅里叶变换的研究，主要集中于实现高于双精度的精度 [Van Der Hoeven 和 Lecerf 2015； Sorna 等人 2018年； Frewer 2021； Gong等人 2022年； Tolmachev 2023年； Kawakami 和 Takahashi 2024]；然而，所有现有方法都针对每种目标精度从头实现快速傅里叶变换内核。目前还没有提出类似 Ozaki 方案的方法，即通过重用现有的优化低精度快速傅里叶变换，以目标精度计算快速傅里叶变换。

本文提出了这样一种方法。我们的关键观察是，布鲁斯坦快速傅里叶变换 [Bluestein 1970] 将离散傅里叶变换化为循环卷积，因此可以应用 Ozaki 方案。随后，使用数论变换（NTT）精确计算拆分分量卷积 [Pollard 1971], ——即在有限域上的快速傅里叶变换，而非浮点快速傅里叶变换，并结合中国剩余定理（CRT），从而避免卷积阶段的舍入误差。

本文的贡献如下：

- 我们提出一种方法，通过将 Ozaki 方案应用于 Bluestein 快速傅里叶变换中的循环卷积，并利用数论变换和中国剩余定理确保分量拆分卷积的精确性，从而使用低精度快速傅里叶变换计算达到目标精度的快速傅里叶变换。
- 我们提出分割数的上界，并引入数论变换域累加策略，二者均可减少所需的数论变换调用次数。
- 我们使用 32 位数论变换实现了双精度布鲁斯坦快速傅里叶变换，并评估其精度和计算成本，结果表明，基于 Ozaki 方案的循环卷积能够有效降低快速傅里叶变换的相对误差。

本文侧重于建立算法框架，并评估其精度和计算成本。

本文其余部分安排如下。第 2 节回顾相关工作。第 3 节介绍记号，并概述快速傅里叶变换、数论变换、中国剩余定理和 Ozaki 方案的背景。第 4节介绍所提方法。第 5 节报告实验结果。第 6 节讨论实现竞争力性能的条件、向任意长度快速傅里叶变换的扩展、可移植性，以及对其他精度的适用性。第 7 节总结全文。

2 相关工作

2.1 矩阵乘法的 Ozaki 方案

Ozaki 方案 [Ozaki 等人 2012]通过低精度矩阵-矩阵乘积，以目标精度计算矩阵-矩阵乘积。该方法将输入矩阵拆分为多个分量，使每个分裂分量乘积都能由标准BLAS例程精确计算，然后累加结果。这种方法已应用于各种场景，以利用硬件性能差距。Mukunoki 等人 [2020] 在 NVIDIA GPU 上使用半精度 Tensor Core 矩阵乘法来加速单精度和双精度矩阵乘积，而 Ootomo 等人 [2024] 则使用 8位整数Tensor Core运算实现相同目的。对于高于双精度的精度，Mukunoki 等人 [2021] 使用双精度BLAS加速四倍精度矩阵乘法，而 Utsugiri 和 Kouya [2022] 则分别使用双精度和单精度BLAS例程，加速三双精度和三单精度（TS）矩阵-矩阵乘积。

2.2 使用低精度算术计算目标精度快速傅里叶变换

使用低精度算术以目标精度计算快速傅里叶变换的方法，主要是为双精度以上的精度而开发的。VkFFT [Tolmachev 2023], 这一跨平台 GPU 快速傅里叶变换库提供 Double-Double 精度的快速傅里叶变换。Gong 等人 [2022]实现了 Double-Double 精度的区间算术快速傅里叶变换，而 Kawakami 和 Takahashi [2024] 实现了 Quad-Double 精度的快速傅里叶变换。Hoeven 和 Lecerf [2015] 使用定点算术实现了字长为 4×、 6×和 8×比特的快速傅里叶变换。Frewer [2021] 开发了 mpFFT，这是一个基于 MPFR 构建的任意精度快速傅里叶变换库。

Sorna 等人 [2018]在 NVIDIA GPU 上使用 Tensor Core 半精度运算，采用了类似的方法，其计算快速傅里叶变换的精度高于 cuFFT 的半精度快速傅里叶变换。所有这些方法都采用一种共同思路：先用低精度运算实现目标精度算术，然后从头编写专用的快速傅里叶变换内核。我们的方法则不同：通过将 Ozaki 方案应用于 Bluestein 快速傅里叶变换中的循环卷积，使用 NTT（即有限域上的快速傅里叶变换）作为低精度快速傅里叶变换例程，来计算目标精度快速傅里叶变换。

3 预备知识

3.1 记号

我们用 $\mathbb{F}_{32}$ 和 $\mathbb{F}_{64}$ 分别表示 IEEE 754-2008 [IEEE 2008], 所定义的单精度（二进制32）和双精度（二进制64）浮点数集合；无需指定浮点类型时，记作 $\mathbb{F}$。我们用 u 表示单位舍入误差；对于单精度，$u_{32} = 2^{-24}$，对于双精度， $u_{64} = 2^{-53}$。我们用 $\mathbf{fl}_{32}(\cdot)$ 和 $\mathbf{fl}_{64}(\cdot)$ 分别表示用单精度和双精度算术计算参数所得的结果；无需指定浮点类型时，记作$\mathbf{fl}(\cdot)$。

对于向量 $\boldsymbol{x}$，我们用 $x(j)$ 表示其第 j 个元素。我们分别用 $\mathrm{Re}(\cdot)$ 和 $\mathrm{Im}(\cdot)$ 表示实部和虚部。对于两个向量 $\boldsymbol{x}$和 $\boldsymbol{y}$，其逐元素乘积 $\boldsymbol{z}$满足 $z(j) = x(j)\ y(j)$，记作 $\boldsymbol{z} = \boldsymbol{x} \odot \boldsymbol{y}$。我们用 $\mathbf{1}_n$表示 n 维全一向量。

整数 a 模 m 的余数按绝对值最小的方式计算：

$$a \bmod m = a - m \cdot \left\lfloor \frac{a}{m} + \frac{1}{2} \right\rfloor.$$

也就是说，它满足

$$-\frac{m}{2} \le a \bmod m < \frac{m}{2}.$$

3.2 快速傅里叶变换

对于复向量 $\boldsymbol{x} \in \mathbb{C}^n$，离散傅里叶变换定义为

$$y(k) = \sum_{j=0}^{n-1} x(j)\, \omega_n^{jk} \quad (0 \le k < n), \tag{1}$$

其中 $y(k) \in \mathbb{C}$、 $\omega_n = e^{-2\pi i/n}$，且 $i = \sqrt{-1}$。逆离散傅里叶变换定义为

$$x(j) = \frac{1}{n} \sum_{k=0}^{n-1} y(k)\, \omega_n^{-jk} \quad (0 \le j < n). \tag{2}$$

我们将 $\boldsymbol{x}$ 的离散傅里叶变换和逆离散傅里叶变换分别记为 $\mathrm{DFT}(\boldsymbol{x})$ 和 $\mathrm{DFT}^{-1}(\boldsymbol{x})$。根据定义，$\mathrm{DFT}^{-1}$ 是DFT的逆变换；也就是说，$\mathrm{DFT}^{-1}(\mathrm{DFT}(\boldsymbol{x})) = \boldsymbol{x}$。直接计算式（1）和式（2）需要$O\left(n^2\right)$ 次运算。

对于合数长度，离散傅里叶变换可以分解为较小的离散傅里叶变换，由此得到广为人知的基数快速傅里叶变换算法。通过递归应用此类分解，Cooley和Tukey [Cooley和Tukey 1965] 提出的算法以及斯托克汉姆快速傅里叶变换 [Cochran等人 1967] 等算法，能够在适合进行基数分解的长度下，于 $O(n\log n)$ 时间内计算 n 点离散傅里叶变换。下文中，如果通过此类 $O(n\log n)$ 快速傅里叶变换程序及其逆变换来计算 $\boldsymbol{x}$ 的离散傅里叶变换和逆离散傅里叶变换，我们分别将其记为 $\mathrm{FFT}(\boldsymbol{x})$ 和 $\mathrm{FFT}^{-1}(\boldsymbol{x})$；需要明确给出长度时，则记为 $\mathrm{FFT}_n(\boldsymbol{x})$ 和 $\mathrm{FFT}_n^{-1}(\boldsymbol{x})$。

然而，对于素数长度或含有较大素因子的长度，基数快速傅里叶变换无法直接高效地应用，因此通常会采用其他方法。一种经典方法可以在 $O(n\log n)$ 时间内计算任意长度的离散傅里叶变换，即布鲁斯坦快速傅里叶变换 [Bluestein1970] （也称为啁啾z 变换）。布鲁斯坦快速傅里叶变换将一个 n 点离散傅里叶变换重写为卷积，并通过对适当选取长度的基于快速傅里叶变换的循环卷积来求值 N。

在布鲁斯坦快速傅里叶变换中，(1) 转换为

$$y(k) = \sum_{j=0}^{n-1} x(j)\omega_n^{jk} = \omega_{2n}^{k^2} \sum_{j=0}^{n-1} \left(x(j)\omega_{2n}^{j^2}\right) \omega_{2n}^{-(k-j)^2} \quad (0 \le k < n).$$

设 $\boldsymbol{\omega} \in \mathbb{C}^n$ 满足 $\omega(j) = \omega_{2n}^{j^2}$。那么，$\boldsymbol{y} =$ 离散傅里叶变换$(\boldsymbol{x})$ 可按如下方式计算：

步骤 1. 计算逐元素乘积：$\boldsymbol{x}' = \boldsymbol{x} \odot \boldsymbol{\omega}$。

步骤 2. 计算卷积：

$$y'(k) = \sum_{j=0}^{n-1} x'(j)\, \omega_{2n}^{-(k-j)^2} \quad (0 \le k < n). \tag{3}$$

步骤 3. 计算逐元素乘积：$\boldsymbol{y} = \boldsymbol{y}' \odot \boldsymbol{\omega}$。

为了高效计算 (3)，将其卷积和嵌入到长度为$N \ge 2n-1$的循环卷积中。为实现这一嵌入，我们使用两种填充算子。零填充$(\cdot\ N)$ 表示标准零填充，即将长度为n的序列扩展至长度 N。零填充$_{\pm}(\cdot\ N)$ 表示带索引反转（环绕）的零填充。具体而言，将零填充后的序列记作带波浪号的形式，我们定义

$$\widetilde{\boldsymbol{x}'} = \mathrm{ZeroPad}(\boldsymbol{x}', N) \in \mathbb{C}^N,$$
$$\widetilde{\boldsymbol{\omega}^*} = \mathrm{ZeroPad}_{\pm}(\boldsymbol{\omega}^*, N) \in \mathbb{C}^N,$$

where $\boldsymbol{\omega}^*$ 是 $\boldsymbol{\omega}$的逐元素复共轭，其中 $\omega^*(j) = \omega_{2n}^{-j^2}$。用分量表示

$$\widetilde{x'}(j) = \begin{cases} x'(j) & (0 \le j < n), \\ 0 & (n \le j < N), \end{cases}$$

$$\widetilde{\omega^*}(j) = \begin{cases} \omega^*(j) & (0 \le j < n), \\ 0 & (n \le j < N-n+1), \\ \omega^*(N-j) & (N-n+1 \le j < N). \end{cases}$$

然后，循环卷积可以通过长度为 N 的快速傅里叶变换计算为

$$\widetilde{\boldsymbol{y}'} = \mathrm{FFT}_N^{-1}\left(\mathrm{FFT}_N\left(\widetilde{\boldsymbol{x}'}\right) \odot \mathrm{FFT}_N\left(\widetilde{\boldsymbol{\omega}^*}\right)\right),$$

$\widetilde{\ }$所需的值 $\boldsymbol{y}'$ 可作为 $\boldsymbol{y}'$的前 n 项得到。选择 N为一个可执行（$N\log\ N$）快速傅里叶变换的长度后，步骤 2 中的循环卷积可在

算法 1 布鲁斯坦快速傅里叶变换

要求： $\boldsymbol{x} \in \mathbb{C}^n$，$\boldsymbol{\omega} \in \mathbb{C}^n$满足 $\omega\ (j)\ = \omega_{2n}^{j^2}$，快速傅里叶变换$_N$，快速傅里叶变换$_N^{-1}$满足 $N \geq 2n-1$**确保：** $\boldsymbol{y}$ =离散傅里叶变换$(\boldsymbol{x})\ \in \mathbb{C}^n$

1: $\boldsymbol{x}' \leftarrow \boldsymbol{x} \odot \boldsymbol{\omega}$ ⌢2: $\boldsymbol{x}' \leftarrow$ 零填充$(\boldsymbol{x}', N)$ ⌢3: $\boldsymbol{\omega}^* \leftarrow$ 零填充$_{\pm}(\boldsymbol{\omega}^*, N)$4: $\widetilde{\boldsymbol{y}'} \leftarrow$ 快速傅里叶变换$_N^{-1}\left($快速傅里叶变换$_N\left(\widetilde{\boldsymbol{x}'}\right) \odot$ 快速傅里叶变换$_N\left(\widetilde{\boldsymbol{\omega}^*}\right)\right)$ ⌢5: $\boldsymbol{y}' \leftarrow$ 前 n 个条目 $\boldsymbol{y}'$6: $\boldsymbol{y} \leftarrow \boldsymbol{y}' \odot \boldsymbol{\omega}$

$O(N$对数 $N)$时间。计入 $O(n)$步骤 1 和 3 中的逐元素乘积后，布鲁斯坦快速傅里叶变换的总体复数域复杂度为 $O(N$对数 $N+n)\ = O(n$对数$\,n)$。布鲁斯坦快速傅里叶变换的算法总结于算法 1。

当 n 是 2 的幂时，通常可以使用库利-图基快速傅里叶变换或 Stockham 快速傅里叶变换来计算离散傅里叶变换，因此通常不会使用布鲁斯坦快速傅里叶变换。不过，如果对这样的长度使用布鲁斯坦快速傅里叶变换，式（3）中的卷积会变成循环卷积，因此无需进行零填充，FFT 长度 N可以保持为 n。在这种情况下，布鲁斯坦快速傅里叶变换通过循环卷积来计算离散傅里叶变换，需要进行两次长度为n 的正向快速傅里叶变换和一次长度为n的逆快速傅里叶变换，此外还需要计算逐元素乘积。尽管渐近复杂度仍为$O\ (n\log n)$，但其常数因子约为库利-图基或斯托克汉姆算法等基数快速傅里叶变换的三倍。

3.3 数论变换与中国剩余定理

数论变换 [Pollard 1971] 是在有限域 $\mathbb{Z}/p\mathbb{Z} = \mathbb{Z}_p$ 上针对素数模数 p定义的离散傅里叶变换。对于一个长度为n 的向量 $\boldsymbol{x} \in \mathbb{Z}_p^n$，数论变换定义为

$$y(k) = \sum_{j=0}^{n-1} x(j)\,\omega_n^{jk} \bmod p \quad (0 \leq k < n), \tag{4}$$

其中 $\omega_n \in \mathbb{Z}_p$是一个本原的 n 次单位根；也就是说，对于$0 < j < n$，有 $\omega_n^n \equiv 1\ (\text{模}\ p)$ 且 $\omega_n^j \not\equiv 1\ (\text{模}\ p)$。逆数论变换定义为

$$x(j) = n^{-1} \sum_{k=0}^{n-1} y(k)\,\omega_n^{-jk} \bmod p \quad (0 \leq j < n), \tag{5}$$

其中 n^{-1}和 ω_n^{-jk}分别表示 n和 ω_n^{jk}在 $\mathbb{Z}_p$中的乘法逆元。我们将模 p 下的数论变换和逆数论变换分别记作 $\text{NTT}_p(\boldsymbol{x})$ 和 $\text{NTT}_p^{-1}(\boldsymbol{x})$。当模数可由上下文确定时，我们省略下标 p，记作 $\text{NTT}(\boldsymbol{x})$ 和 $\text{NTT}^{-1}(\boldsymbol{x})$。根据定义，逆数论变换是数论变换的逆变换；也就是说，$\text{NTT}_p^{-1}(\text{NTT}_p(\boldsymbol{x}))\ = \boldsymbol{x}$。

直接计算（4）和（5）需要$O\left(n^2\right)$次运算。与复数域情形一样，当 n具有适当的因式分解时，可以使用库利-图基型或斯托克汉姆型分解，在 $O(n\log n)$时间内计算数论变换。下文中，数论变换$(\cdot)$和 数论变换$^{-1}(\cdot)$分别表示通过 $O(n\log n)$算法计算的数论变换和逆数论变换。

对于 $\boldsymbol{x}$，$\boldsymbol{y} \in \mathbb{Z}_p^n$，模 p 的循环卷积 $\boldsymbol{z} = \boldsymbol{x} \circledast \boldsymbol{y}$可根据卷积定理计算为

$$\boldsymbol{z} = \text{NTT}_p^{-1}\left(\text{NTT}_p(\boldsymbol{x}) \odot \text{NTT}_p(\boldsymbol{y})\right).$$

如果 $\boldsymbol{x}$、$\boldsymbol{y}$ 和 $\boldsymbol{x} \circledast \boldsymbol{y}$ 中的所有元素都位于 $[-p/2,\ p/2)$，则可以使用数论变换精确计算整数循环卷积 $\boldsymbol{x} \circledast \boldsymbol{y}$。

设 m_0、……、m_{d-1} 为两两互素整数，并设 $M = \prod_{j=0}^{d-1} m_j$。中国剩余定理指出，给定 $0 \le j < d$ 的剩余 $a \equiv a_j$(模 m_j) 后，解 a 在模 M 意义下唯一确定。一种显式重构方法为

$$a \equiv \sum_{j=0}^{d-1} a_j \, M_j \, M_j^{-1} \pmod{M},$$

其中，$M_j = M/m_j$ 和 M_j^{-1} 表示 M_j 模 m_j 的乘法逆元。

中国剩余定理可用于重构使用不同模数的 NTT 计算得到的整数循环卷积。设 p_0、……、p_{d-1} 为互异素数，并设 $P = \prod_{j=0}^{d-1} p_j$。为了计算整数向量 $\boldsymbol{x}$ 和 $\boldsymbol{y}$ 模 P 的循环卷积，我们首先计算各个 p_j 模数下的结果：

$$\boldsymbol{z}_j = \mathrm{NTT}_{p_j}^{-1}\Big(\mathrm{NTT}_{p_j}(\boldsymbol{x} \bmod p_j) \odot \mathrm{NTT}_{p_j}(\boldsymbol{y} \bmod p_j)\Big) \quad (0 \le j < d).$$

然后，通过中国剩余定理重构得到 $\boldsymbol{z} = \boldsymbol{x} \circledast \boldsymbol{y}$ 模 P：

$$\boldsymbol{z} \equiv \sum_{j=0}^{d-1} \boldsymbol{z}_j \, P_j \, P_j^{-1} \pmod{P}, \tag{6}$$

其中，$P_j = P/p_j$和 P_j^{-1}表示 P_j模 p_j的乘法逆元。

我们不直接计算 (6)，而是可以通过 Garner 算法逐步重构 $\boldsymbol{z}$ [Gar-ner 1959]。令 $P'_j = \prod_{k=0}^{j} p_k$ 并设 $\boldsymbol{z}'_0 = \boldsymbol{z}_0$。对于 $1 \le j < d$，定义

$$\boldsymbol{z}'_j = \boldsymbol{z}'_{j-1} + \left\{\left(\boldsymbol{z}_j - \boldsymbol{z}'_{j-1}\right)\left(P'_{j-1}\right)^{-1} \bmod p_j\right\} P'_{j-1},$$

其中，$\left(P'_{j-1}\right)^{-1}$表示 P'_{j-1}模 p_j的乘法逆元。于是，$\boldsymbol{z}'_{d-1} \equiv \boldsymbol{z}$(模 P)成立。

3.4 Ozaki 方案

Ozaki 方案是一种用于矩阵乘法的无误差变换。为便于阐述，本文介绍 Ozaki 方案在点积中的应用；同一思路也适用于矩阵-向量和矩阵-矩阵乘积。Ozaki 方案按以下三个步骤计算点积：

步骤 1. 拆分输入向量：

$$\boldsymbol{x} \to \boldsymbol{x}^{(0)} + \cdots + \boldsymbol{x}^{(k_x-1)}, \quad \boldsymbol{y} \to \boldsymbol{y}^{(0)} + \cdots + \boldsymbol{y}^{(k_y-1)}.$$

步骤 2。计算各分割数的点积：

$$z^{(s,t)} = \boldsymbol{x}^{(s)} \cdot \boldsymbol{y}^{(t)} \quad (0 \le s < k_x,\ 0 \le t < k_y).$$

步骤 3。累加结果：

$$z = \sum_{s=0}^{k_x-1} \sum_{t=0}^{k_y-1} z^{(s,t)}.$$

T基于 Ozaki 方案的点积算法总结见算法 2。

算法 2 使用 Ozaki 方案计算点积

要求： $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^n$确保： $z \simeq \boldsymbol{x} \cdot \boldsymbol{y} \in \mathbb{F}$*1:* $\boldsymbol{x}^{(0)}, \ldots, \boldsymbol{x}^{(k_x-1)} \in \mathbb{F}^n \leftarrow$ 拆分 $(\boldsymbol{x})$ *2:* $\boldsymbol{y}^{(0)}, \ldots, \boldsymbol{y}^{(k_y-1)} \in \mathbb{F}^n \leftarrow$ 拆分 $(\boldsymbol{y})$ *3:* 对于 $s = 0, \ldots, k_x - 1$ 执行 *4:* 对于 $t = 0, \ldots, k_y - 1$ 执行 *5:* $z^{(s,t)} \leftarrow \boldsymbol{x}^{(s)} \cdot \boldsymbol{y}^{(t)}$ *6:* $z \leftarrow \sum_{s=0}^{k_x-1} \sum_{t=0}^{k_y-1} z^{(s,t)}$

算法 3 Ozaki 方案中的分割过程

Require: $\boldsymbol{x} \in \mathbb{F}^n$
Ensure: $\boldsymbol{x}^{(0)}, \ldots, \boldsymbol{x}^{(k-1)} \in \mathbb{F}^n$ such that $\sum_{j=0}^{k-1} \boldsymbol{x}^{(j)} = \boldsymbol{x}$

function Split $(\boldsymbol{x})$
  $\rho \leftarrow \left\lceil \left(\log_2\left(u^{-1}\right) + \log_2(n)\right)/2 \right\rceil$
  $\boldsymbol{r}^{(0)} \leftarrow \boldsymbol{x}$
  $\mu^{(0)} \leftarrow \left\|\boldsymbol{r}^{(0)}\right\|_\infty$
  $k \leftarrow 0$
  while $\mu^{(k)} > 0$ **do**
    $\sigma^{(k)} \leftarrow 2^{\left\lceil \log_2\left(\mu^{(k)}\right)\right\rceil + \rho}$
    $\boldsymbol{x}^{(k)} \leftarrow \mathbf{fl}\left(\left(\boldsymbol{r}^{(k)} + \sigma^{(k)}\mathbf{1}_n\right) - \sigma^{(k)}\mathbf{1}_n\right)$
    $\boldsymbol{r}^{(k+1)} \leftarrow \boldsymbol{r}^{(k)} - \boldsymbol{x}^{(k)}$
    $\mu^{(k+1)} \leftarrow \left\|\boldsymbol{r}^{(k+1)}\right\|_\infty$
    $k \leftarrow k + 1$
  return $\boldsymbol{x}^{(0)}, \ldots, \boldsymbol{x}^{(k-1)}$

在步骤 1 中，算法 3 将浮点向量 $\boldsymbol{x}$、 $\boldsymbol{y} \in \mathbb{F}^n$ 拆分为以下向量之和： k_x和 k_y定点向量，每个向量含有 α 比特，其中

$$\alpha = \left\lfloor \frac{\log_2\left(u^{-1}\right) - \log_2(n)}{2} \right\rfloor. \tag{7}$$

更具体地说，使用算法 3中定义的缩放因子 $\sigma^{(k)}$， $\boldsymbol{x}$的拆分分量满足以下条件：

$$\left\|\boldsymbol{x}^{(k)}\right\|_\infty \le 2^\alpha u\sigma^{(k)}, \quad \boldsymbol{x}^{(k)} \in u\sigma^{(k)}\mathbb{Z}^n \quad (0 \le k < k_x). \tag{8}$$

$\boldsymbol{y}$的拆分分量也适用相同的条件。采用此 α后，构造出的拆分可确保步骤 2 中的点积不会产生舍入误差；也就是说，

$$\boldsymbol{x}^{(s)} \cdot \boldsymbol{y}^{(t)} = \mathbf{fl}\left(\boldsymbol{x}^{(s)} \cdot \boldsymbol{y}^{(t)}\right)$$

对于所有 (s, t)。

分割数取决于向量元素的分布以及 α。令 $|\boldsymbol{x}|$ 表示 $\boldsymbol{x}$的逐元素绝对值，则分割数 k_x 的上界为

$$k_x \le \left\lceil \frac{\log_2(\max|\boldsymbol{x}|) - \log_2(\min|\boldsymbol{x}|) + \log_2\left(u^{-1}\right) - 1}{\alpha - 1} \right\rceil. \tag{9}$$

因此， k_x 会随着 $\boldsymbol{x}$ 中元素的指数范围变大而增大。此外，当 α 较小时， k_x 往往会增大；也就是说，当 n 较大或尾数长度 $\log_2(u^{-1})$ 较小时。

当 $k_x = k_y = K$ 时，点积的步骤 1 至 3 的计算成本分别为 $O(Kn)$、 $O(K^2n)$ 和 $O(K^2)$。对于矩阵乘法，相应的成本分别为 $O(Kn^2)$、 $O(K^2n^3)$ 和 $O(K^2n^2)$。因此，步骤 2 是主要步骤，总体复杂度对于点积为 $O(K^2n)$，对于矩阵乘法为 $O(K^2n^3)$。由此可见，计算成本由分割数 K 决定，而分割数取决于拆分宽度 α 和输入值的动态范围，如上文所述。

4 所提方法

本节介绍一种使用低精度快速傅里叶变换，以目标精度计算快速傅里叶变换的方法。在整个讨论过程中，我们假设离散傅里叶变换规模 n 是 2 的幂。

4.1 动机：采用 Bluestein 快速傅里叶变换的理由

以高于硬件直接支持的精度计算快速傅里叶变换，通常采用两步方法：(1) 使用多个低精度值和运算来模拟高精度算术；(2) 使用这些模拟运算实现快速傅里叶变换算法。例如，Double-Double 和 Quad-Double 算术 [Hida等人 2001]使用成对或成组的四个双精度数，分别实现 106 比特或 212 比特的尾数。人们已使用这些数据类型实现快速傅里叶变换。

遵循这一范式，若要使用低精度算术计算双精度快速傅里叶变换，我们可以采用 TS 算术 [Fabiano等人 2019], 提出的表示方式，即使用三个单精度值表示一个浮点数。TS 算术可提供 $24 \times 3 = 72$ 比特的尾数，超过双精度的 53 比特尾数，不过其指数范围仍受限于单精度的范围。

一个三单精度数将真实值 x 表示为

$$x = x_0 + x_1 + x_2,$$

其中 x_0、 x_1、 $x_2 \in \mathbb{F}_{32}$ 均满足

$$x_{s-1} \neq 0 \implies |x_s| \leq \frac{1}{2}\,\mathrm{ulp}\,(x_{s-1}), \quad s = 1, 2.$$

这里，$\mathrm{ulp}(x)$ 表示 x 的末位单位。我们将 $x_{0:2}$ 简写为三元组（x_0、 x_1、 x_2），并称其为三单精度数。

现在，考虑使用 TS 算术计算双精度离散傅里叶变换。设 $\boldsymbol{x} \in \mathbb{F}_{64}^n + i\mathbb{F}_{64}^n$ 为一个双精度复向量， $\boldsymbol{\omega} \in \mathbb{F}_{64}^n + i\mathbb{F}_{64}^n$ 为旋转因子向量，其中 $\omega(j) = \mathbf{fl}_{64}\left(\omega_n^j\right)$。我们可以将它们各自拆分为三个单精度复向量 $\boldsymbol{x}_s, \boldsymbol{\omega}_t \in \mathbb{F}_{32}^n + i\mathbb{F}_{32}^n (s, t = 0, 1, 2)$。于是， $\boldsymbol{x}$ 的离散傅里叶变换可表示为

$$\begin{aligned} y(k) &= \sum_{j=0}^{n-1} x(j)\omega_n^{jk} \\ &\simeq \sum_{j=0}^{n-1} \left(\sum_{s=0}^{2} x_s(j)\right)\left(\sum_{t=0}^{2} \omega_t(jk \bmod n)\right) \quad (0 \leq k < n). \end{aligned} \tag{10}$$

算法 4 使用 TS 算术实现的双精度 Bluestein 快速傅里叶变换。这里，TS(·) 将一个双精度值转换为其 TS 表示，而 TSAdd、TSSub 和 TSMul 分别表示 TS 加法、TS 减法和 TS 乘法。

要求： $\boldsymbol{x}, \boldsymbol{\omega}, \boldsymbol{\omega}^* \in \mathbb{F}_{64}^n + i\mathbb{F}_{64}^n$确保：$\boldsymbol{y} \in \mathbb{F}_{64}^n + i\mathbb{F}_{64}^n$ 使得 $\boldsymbol{y} \simeq$ 离散傅里叶变换$(\boldsymbol{x})$
1. 将输入转换为 TS。 1: $\boldsymbol{x}_{0:2} \leftarrow \text{TS}(\boldsymbol{x})$ 2: $\boldsymbol{\omega}_{0:2} \leftarrow \text{TS}(\boldsymbol{\omega})$ 3: $\boldsymbol{\omega}^*_{0:2} \leftarrow \text{TS}(\boldsymbol{\omega}^*)$
2. 在 TS 中计算逐元素乘积： $\boldsymbol{x}' \leftarrow \boldsymbol{x} \odot \boldsymbol{\omega}$。 4: $\text{Re}(\boldsymbol{x}'_{0:2}) \leftarrow$ TSSub (TSMul (Re $(\boldsymbol{x}_{0:2})$, Re $(\boldsymbol{\omega}_{0:2})$) ,TSMul (Im $(\boldsymbol{x}_{0:2})$,Im $(\boldsymbol{\omega}_{0:2})$)) 5: $\text{Im}(\boldsymbol{x}'_{0:2}) \leftarrow$ TSAdd (TSMul (Re $(\boldsymbol{x}_{0:2})$, Im $(\boldsymbol{\omega}_{0:2})$) ,TSMul (Im $(\boldsymbol{x}_{0:2})$,Re $(\boldsymbol{\omega}_{0:2})$)) 3. 在 TS 中进行循环卷积： $\boldsymbol{y}' \leftarrow \boldsymbol{x}' \circledast \boldsymbol{\omega}^*$。 6: $\boldsymbol{y}'_{0:2} \leftarrow \boldsymbol{x}'_{0:2} \circledast \boldsymbol{\omega}^*_{0:2}$4. 在 TS 中计算逐元素乘积： $\boldsymbol{y} \leftarrow \boldsymbol{y}' \odot \boldsymbol{\omega}$。 7: $\text{Re}(\boldsymbol{y}_{0:2}) \leftarrow$ TSSub (TSMul (Re $(\boldsymbol{y}'_{0:2})$, Re $(\boldsymbol{\omega}_{0:2})$),TSMul (Im $(\boldsymbol{y}'_{0:2})$,Im $(\boldsymbol{\omega}_{0:2})$))8: $\text{Im}(\boldsymbol{y}_{0:2}) \leftarrow$ TSAdd (TSMul (Re $(\boldsymbol{y}'_{0:2})$, Im $(\boldsymbol{\omega}_{0:2})$), TSMul (Im $(\boldsymbol{y}'_{0:2})$,Re $(\boldsymbol{\omega}_{0:2})$))5. 将 TS 结果转换为双精度浮点数。 9: $\boldsymbol{y} \leftarrow \text{fl}_{64}(\boldsymbol{y}_0 + \boldsymbol{y}_1 + \boldsymbol{y}_2)$

我们可以使用 TS 算术实现库利-图基或斯托克汉姆快速傅里叶变换，以在10中计算$O(n \log n)$时间。然而，展开10可得：

$$\begin{aligned} &\sum_{j=0}^{n-1} \left(\sum_{s=0}^{2} x_s(j) \right) \left(\sum_{t=0}^{2} \omega_t(jk \bmod n) \right) \quad (0 \le k < n) \\ =&\sum_{s=0}^{2} \sum_{t=0}^{2} \sum_{j=0}^{n-1} x_s(j)\omega_t(jk \bmod n) \quad (0 \le k < n). \end{aligned}$$

每一项 $\sum_{j=0}^{n-1} x_s(j)\omega_t(\ jk \bmod\ n)$ 都不具备离散傅里叶变换结构，因此我们无法使用单精度 FFT 例程同时计算所有项。我们的目标是仅使用低精度快速傅里叶变换例程来计算目标精度快速傅里叶变换，从而利用高度优化的硬件实现。为实现这一目标，我们采用布鲁斯坦快速傅里叶变换算法，该算法将离散傅里叶变换重新表述为循环卷积。我们考虑使用 TS 算术和布鲁斯坦快速傅里叶变换方法计算双精度离散傅里叶变换。算法步骤如下：

步骤 1. 将双精度复向量 $\boldsymbol{x}$、 $\boldsymbol{\omega}$和 $\boldsymbol{\omega}^*$ 转换为 TS 复向量。步骤 2. 使用 TS 算术计算逐元素乘积 $\boldsymbol{x}' \leftarrow \boldsymbol{x} \odot \boldsymbol{\omega}$。步骤 3. 使用 TS 算术计算循环卷积 $\boldsymbol{y}' \leftarrow \boldsymbol{x}' \circledast \boldsymbol{\omega}^*$。步骤 4. 使用 TS 算术计算逐元素乘积 $\boldsymbol{y} \leftarrow \boldsymbol{y}' \odot \boldsymbol{\omega}$。步骤 5. 将结果 $\boldsymbol{y}$ 从 TS 复向量转换回双精度复向量。

上述过程总结于算法4中。

现在，考虑步骤 3 中两个 TS 向量之间的循环卷积 $\boldsymbol{x}_s$， $\boldsymbol{y}_t \in \mathbb{F}_{32}^n (s,\ t = 0、1、2)$。其形式如下：

$$\begin{aligned} z(k) &= \sum_{j=0}^{n-1} x(j)\, y((k-j) \bmod n) \\ &\simeq \sum_{j=0}^{n-1} \left(\sum_{s=0}^{2} x_s(j) \right) \left(\sum_{t=0}^{2} y_t((k-j) \bmod n) \right) \quad (0 \le k < n). \end{aligned} \tag{11}$$

我们可以使用库利-图基或斯托克汉姆快速傅里叶变换，在 TS 算术中通过基于 FFT 的卷积计算 (11)，时间复杂度为 $O(n \log n)$。然而，展开 (11) 可得：

$$\begin{aligned} &\sum_{j=0}^{n-1} \left(\sum_{s=0}^{2} x_s(j) \right) \left(\sum_{t=0}^{2} y_t((k-j) \bmod n) \right) \\ &= \sum_{s=0}^{2} \sum_{t=0}^{2} \sum_{j=0}^{n-1} x_s(j)\, y_t((k-j) \bmod n) \quad (0 \le k < n). \end{aligned}$$

关键在于，$\sum_{j=0}^{n-1} x_s(j)\ y_t((k-j) \bmod n)$ 是单精度向量 $\boldsymbol{x}_s$和 $\boldsymbol{y}_t$之间的循环卷积。因此，我们可以使用单精度 FFT 例程，在 $O(n \log n)$ 时间内计算它。

然而，使用单精度快速傅里叶变换的朴素实现对每一项只能达到单精度精度，因此，即使将所有项求和以形成 TS 精度结果，仍会引入误差。这正是 Ozaki 方案发挥关键作用之处：如果我们能将 Ozaki 方案应用于循环卷积，就可以将 TS 向量拆分为多个单精度向量，使每个分裂分量的循环卷积都能通过单精度快速傅里叶变换精确计算（没有舍入误差）。这就是所提方法的核心思想。

4.2 Ozaki 方案在循环卷积中的应用

对于浮点向量 $\boldsymbol{x},\ \boldsymbol{y} \in \mathbb{F}^n$，循环卷积 $\boldsymbol{z} = \boldsymbol{x} \circledast \boldsymbol{y}$ 可表示为矩阵-向量乘积 $\boldsymbol{z} = Y\boldsymbol{x}$，其中使用由 $\boldsymbol{y}$ 生成的循环矩阵：

$$Y = \left(y_{jk}\right) = \begin{pmatrix} y(0) & y(n-1) & \cdots & y(1) \\ y(1) & y(0) & \cdots & y(2) \\ \vdots & \vdots & \ddots & \vdots \\ y(n-1) & y(n-2) & \cdots & y(0) \end{pmatrix} \in \mathbb{F}^{n \times n}.$$

由于 $y_{jk} = y((j-k) \bmod n)$，$Y$的每一行都是 $\boldsymbol{y}$中各元素的循环移位。因此，可以将卷积视为矩阵-向量乘积，从而将 Ozaki 方案应用于循环卷积。

在算法 3 中，量 $\mu^{(k)}$和 $\sigma(k)$与元素的排列顺序无关。因此，如果对 $\boldsymbol{y}$应用算法 3得到分裂 $\boldsymbol{y} = \sum_{t=0}^{k_y-1} \boldsymbol{y}^{(t)}$，那么我们可以定义$Y = \sum_{t=0}^{k_y-1} Y^{(t)}$，其元素为 $y_{(t)jk} = y_{(t)}((j-k) \bmod n)$，其中每个 $Y_{(t)}$都是循环矩阵。因此，如果 $\boldsymbol{x}$通过算法 3分裂为 $\boldsymbol{x} = \sum_{s=0}^{k_x-1} \boldsymbol{x}_{(s)}$，那么循环卷积 $\boldsymbol{z} = \boldsymbol{x} \circledast \boldsymbol{y}$可计算为

$$\boldsymbol{z} = \sum_{s=0}^{k_x-1} \sum_{t=0}^{k_y-1} Y^{(t)} \boldsymbol{x}^{(s)} = \sum_{s=0}^{k_x-1} \sum_{t=0}^{k_y-1} \boldsymbol{x}^{(s)} \circledast \boldsymbol{y}^{(t)}. \tag{12}$$

直接计算 $Y_{(t)}\boldsymbol{x}_{(s)}$需要 $O(n^2)$ 的时间，而每个循环卷积项 $\boldsymbol{x}_{(s)} \circledast \boldsymbol{y}^{(t)}$使用基于 FFT 的卷积可在 $O(n \log n)$ 的时间内计算。

使用基于 FFT 的卷积时，(12) 中的每一项都计算为

$$\boldsymbol{x}^{(s)} \circledast \boldsymbol{y}^{(t)} = \mathrm{FFT}^{-1}\left(\mathrm{FFT}\left(\boldsymbol{x}^{(s)}\right) \odot \mathrm{FFT}\left(\boldsymbol{y}^{(t)}\right)\right).$$

这是一个实数循环卷积；因此，使用浮点快速傅里叶变换计算时会产生舍入误差。

相比之下，对于使用快速傅里叶变换计算的整数卷积，如果精确结果的绝对值不超过 u^{-1}，且数值误差严格小于 1/2，则可通过舍入到最近整数恢复精确整数结果。根据（8），每个分裂分量都可以乘以缩放因子 $\left(u\sigma_{(k)}\right)^{-1}$，转换为各元素绝对值不超过 2^{α} 的整数向量。因此，我们采用这种整数方法。设 $c_x^{(s)} = \left(u\sigma_{(s)}\right)^{-1}$ 和 $c_y^{(t)} = \left(u\sigma^{(t)}\right)^{-1}$ 分别为分裂 $\boldsymbol{x}$ 和 $\boldsymbol{y}$ 得到的缩放因子。那么，$\boldsymbol{x}^{(s)} \circledast \boldsymbol{y}^{(t)}$ 可计算为

$$\boldsymbol{x}^{(s)} \circledast \boldsymbol{y}^{(t)} = \left\lfloor \mathrm{FFT}^{-1}\left(\mathrm{FFT}\left(c_x^{(s)}\boldsymbol{x}^{(s)}\right) \odot \mathrm{FFT}\left(c_y^{(t)}\boldsymbol{y}^{(t)}\right)\right)\right\rceil \Big/ \left(c_x^{(s)}c_y^{(t)}\right),$$

其中 $\lfloor\cdot\rceil$ 表示舍入到最近整数。

然而，使用浮点快速傅里叶变换计算整数循环卷积时，误差不一定小于 1/2。基于 FFT 的卷积的最大误差上界由 [Percival 2003]

$$\|\boldsymbol{z}' - \boldsymbol{z}\|_\infty \le \left\|c_x^{(s)}\boldsymbol{x}^{(s)}\right\|_2 \cdot \left\|c_y^{(t)}\boldsymbol{y}^{(t)}\right\|_2 \cdot \left\{(1+u)^{3n}\left(1+u\sqrt{5}\right)^{3n+1}\left(1+\|\boldsymbol{\omega}' - \boldsymbol{\omega}\|_\infty\right)^{3n} - 1\right\},$$

其中，$\boldsymbol{z}'$ 是通过基于快速傅里叶变换的循环卷积计算出的结果，$\boldsymbol{z}$ 是精确值。这里，$\boldsymbol{\omega}'$ 表示快速傅里叶变换中使用的三角函数表，$\boldsymbol{\omega}$ 表示精确三角函数值；因此，$\|\boldsymbol{\omega}' - \boldsymbol{\omega}\|_\infty$ 是最大表误差。假设表项均正确舍入，则有$\|\boldsymbol{\omega}' - \boldsymbol{\omega}\|_\infty \le u/\sqrt{2}$。

如果 $\left\|c_x^{(s)}\boldsymbol{x}^{(s)}\right\|_\infty, \left\|c_y^{(t)}\boldsymbol{y}^{(t)}\right\|_\infty \le 2^{\alpha}$，那么 $\left\|c_x^{(s)}\boldsymbol{x}^{(s)}\right\|_2 \cdot \left\|c_y^{(t)}\boldsymbol{y}^{(t)}\right\|_2 \le n2^{2\alpha}$。为了确保正确舍入到精确结果，我们要求 $\|\boldsymbol{z}' - \boldsymbol{z}\|_\infty < 1/2$；也就是说，

$$\|\boldsymbol{z}' - \boldsymbol{z}\|_\infty \le n2^{2\alpha} \cdot \left\{(1+u)^{3n}\left(1+u\sqrt{5}\right)^{3n+1}\left(1+\frac{u}{\sqrt{2}}\right)^{3n} - 1\right\} < \frac{1}{2}.$$

解该不等式可得 α

$$\alpha < -\frac{1}{2}\left[1 + \log_2(n) + \log_2\left\{(1+u)^{3n}\left(1+u\sqrt{5}\right)^{3n+1}\left(1+\frac{u}{\sqrt{2}}\right)^{3n} - 1\right\}\right].$$

因此，只需选择

$$\alpha = \left\lfloor -\frac{1}{2}\left[1 + \log_2(n) + \log_2\left\{(1+u)^{3n}\left(1+u\sqrt{5}\right)^{3n+1}\left(1+\frac{u}{\sqrt{2}}\right)^{3n} - 1\right\}\right]\right\rfloor. \tag{13}$$

这个条件比（7）更严格；后者决定了 α 在原始 Ozaki 方案中的取值。

相比之下，使用数论变换可以精确计算整数循环卷积。采用模 p, $\boldsymbol{x}^{(s)} \circledast \boldsymbol{y}^{(t)}$ 的数论变换时，可以将其计算为

$$\boldsymbol{x}^{(s)} \circledast \boldsymbol{y}^{(t)} = \mathrm{NTT}_p^{-1}\left(\mathrm{NTT}_p\left(c_x^{(s)}\boldsymbol{x}^{(s)}\right) \odot \mathrm{NTT}_p\left(c_y^{(t)}\boldsymbol{y}^{(t)}\right)\right) \Big/ \left(c_x^{(s)}c_y^{(t)}\right).$$

由于 $\left\|c^{(s)}\boldsymbol{x}^{(s)} \circledast c^{(t)}\boldsymbol{y}^{(t)}\right\|_\infty \le n\left\|c^{(s)}\boldsymbol{x}^{(s)}\right\|_\infty \cdot \left\|c^{(t)}\boldsymbol{y}^{(t)}\right\|_\infty$，精确恢复的一个充分条件是

$$n\left\|c_x^{(s)}\boldsymbol{x}^{(s)}\right\|_\infty \cdot \left\|c_y^{(t)}\boldsymbol{y}^{(t)}\right\|_\infty < \frac{p}{2}.$$

此外，如果 $\left\|c_{(s)x}\ \boldsymbol{x}_{(s)}\right\|_\infty, \left\|c_{(y}{}^{t}\ \boldsymbol{y}^{(t)})\right\|_\infty \le 2^\alpha$，那么

$$n\,2^{2\alpha} < \frac{p}{2},$$

这意味着

$$\alpha < \frac{\log_2(p/2) - \log_2(n)}{2}.$$

因此，只需选择

$$\alpha = \left\lfloor \frac{\log_2(p/2) - \log_2(n)}{2} \right\rfloor. \tag{14}$$

取足够大的 p后，该约束相比基于快速傅里叶变换的约束（13）变得不那么严格。

为进一步放宽这一约束，我们可以使用中国剩余定理，将具有两个不同模数的 NTT 结合起来。为了使用模为 p_0 和 p_1 的 NTT 计算 $\boldsymbol{x}_{(s)} \circledast \boldsymbol{y}^{(t)}$，我们首先计算 $c_{(s)x}\ \boldsymbol{x}_{(s)}$ 以及 $c_{(y}{}^{t}\ \boldsymbol{y}^{(t)})$ 关于 p_0 和 p_1 的剩余：

$$\boldsymbol{x}_0^{(s)} = c_x^{(s)}\boldsymbol{x}^{(s)} \bmod p_0, \quad \boldsymbol{x}_1^{(s)} = c_x^{(s)}\boldsymbol{x}^{(s)} \bmod p_1,$$
$$\boldsymbol{y}_0^{(t)} = c_y^{(t)}\boldsymbol{y}^{(t)} \bmod p_0, \quad \boldsymbol{y}_1^{(t)} = c_y^{(t)}\boldsymbol{y}^{(t)} \bmod p_1.$$

然后，我们使用数论变换计算模 p_0 和 p_1 下的循环卷积：

$$\boldsymbol{z}_0'^{(s,t)} = \mathrm{NTT}_{p_0}^{-1}\left(\mathrm{NTT}_{p_0}\left(\boldsymbol{x}_0^{(s)}\right) \odot \mathrm{NTT}_{p_0}\left(\boldsymbol{y}_0^{(t)}\right)\right) \bmod p_0,$$
$$\boldsymbol{z}_1'^{(s,t)} = \mathrm{NTT}_{p_1}^{-1}\left(\mathrm{NTT}_{p_1}\left(\boldsymbol{x}_1^{(s)}\right) \odot \mathrm{NTT}_{p_1}\left(\boldsymbol{y}_1^{(t)}\right)\right) \bmod p_1.$$

最后，我们通过中国剩余定理重构 $\boldsymbol{x}_{(s)} \circledast \boldsymbol{y}^{(t)}$，得到

$$\boldsymbol{x}^{(s)} \circledast \boldsymbol{y}^{(t)} = \left(\boldsymbol{z}_0'^{(s,t)}\, p_1\, p_1^{-1} + \boldsymbol{z}_1'^{(s,t)}\, p_0\, p_0^{-1}\right) \Big/ \left(c_x^{(s)} c_y^{(t)}\right),$$

其中，p_1^{-1}表示 p_1 模 p_0的乘法逆元，p_0^{-1}表示 p_0 模 p_1的乘法逆元。由于分子对应于模 p_0p_1 和 $\left\|c_{(s)x}\ \boldsymbol{x}_{(s)} \circledast c_{(y}{}^{t}\ \boldsymbol{y}^{(t)})\right\|_\infty \le n \left\|c_{(s)x}\ \boldsymbol{x}_{(s)}\right\|_\infty \cdot \left\|c_{(y}{}^{t}\ \boldsymbol{y}^{(t)})\right\|_\infty$ 下的整数循环卷积，因此精确恢复的一个充分条件是

$$n \left\|c_x^{(s)}\boldsymbol{x}^{(s)}\right\|_\infty \cdot \left\|c_y^{(t)}\boldsymbol{y}^{(t)}\right\|_\infty < \frac{p_0p_1}{2}.$$

因此，与式（14）中一样，α的一个合适选择是

$$\alpha = \left\lfloor \frac{\log_2\left(p_0p_1/2\right) - \log_2\left(n\right)}{2} \right\rfloor. \tag{15}$$

图1比较了单精度下所考察方法得到的 α 值。在此设置下，$u = 2^{-24}$。我们还在式（14）中设定了 $p = 2{,}130{,}706{,}433 < 2^{31}$，并在式（15）中设定了 $p_0 = p$和 $p_1 = 2{,}113{,}929{,}217 < 2^{31}$。使用基于 FFT 的卷积时，对于 $n \ge 2^{10}$，式（13）中的 α变为负值，这意味着连 1 比特都无法分配。此外，对于原始 Ozaki 方案，在$n = 2^{20}$处，式（7）中的 α仅为 2 比特，这表明分割数可能会变得很大。使用单模数数论变换时，式（14）中的 α略大于原始 Ozaki 方案的对应值，但提升幅度有限。相比之下，使用两个不同模数的 NTT 并通过中国剩余定理进行重构时，在 $n = 2^{20}$处，式（15）中的 α达到 20 比特，远大于原始 Ozaki 方案的对应值。根据式（9），较大的 α 意味着分割数更少，因此我们可以预期计算成本会降低。

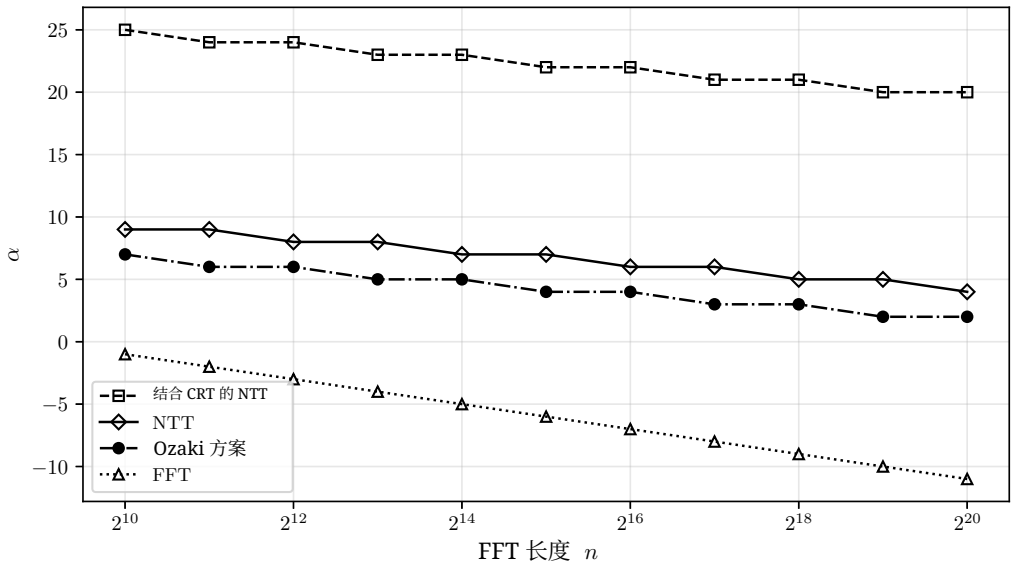

图 1. 单精度下各种卷积方法的拆分宽度 α（每个分裂分量的比特数）对比。α 越大，分割数越少，因此计算成本越低。

4.3 使用 Ozaki 方案进行 TS 精度循环卷积

根据上述讨论，算法5 介绍了一种使用 Ozaki 方案处理 TS 向量的循环卷积算法。给定 TS 向量 $\boldsymbol{x}_{0:2}$ 和 $\boldsymbol{y}_{0:2}$，该算法通过以下三个步骤计算循环卷积 $\boldsymbol{z}_{0:2} \simeq \boldsymbol{x}_{0:2} \circledast \boldsymbol{y}_{0:2}$：

步骤 1。应用 算法 6 将 $\boldsymbol{x}_{0:2}$ 和 $\boldsymbol{y}_{0:2}$ 拆分为多个分量，并计算每个分量模 p_0 和 p_1的 32 位数论变换。步骤 2。对于每一对拆分分量，通过 32 位逆数论变换和中国剩余定理重构计算其循环卷积。步骤 3。将所有成对卷积结果相加，得到 $\boldsymbol{z}_{0:2}$。

在步骤 1 中，对 $\boldsymbol{x}_{0:2}$和 $\boldsymbol{y}_{0:2}$应用算法 6，得到 $X(0)\,0, \ldots, X(k_x-1)\,0, X(0)\,1, \ldots, X(k_x-1)\,1, \boldsymbol{c}_x$；同理，得到 $Y(0)\,0, \ldots, Y(k_y-1)\,0, Y(0)\,1, \ldots, Y(k_y-1)\,1, \boldsymbol{c}_y$。这里，$p_0$和 p_1是互异素数，且 $p_0, p_1 < 2^{32}$，以便进行 32 位整数运算；p_0^{-1}, p_1^{-1}表示它们的乘法逆元，满足$p_0 p_0^{-1}$模 $p_1 = 1$ 和 $p_1 p_1^{-1}$模 $p_0 = 1$。这些量的构造方式使得

$$\boldsymbol{x}_{0:2} = \sum_{s=0}^{k_x-1} \left(\mathrm{NTT}_{p_0}^{-1}\left(X_0^{(s)}\right) p_1 p_1^{-1} + \mathrm{NTT}_{p_1}^{-1}\left(X_1^{(s)}\right) p_0 p_0^{-1} \right) \Big/ c_x^{(s)} ,$$

$$\boldsymbol{y}_{0:2} = \sum_{t=0}^{k_y-1} \left(\mathrm{NTT}_{p_0}^{-1}\left(Y_0^{(t)}\right) p_1 p_1^{-1} + \mathrm{NTT}_{p_1}^{-1}\left(Y_1^{(t)}\right) p_0 p_0^{-1} \right) \Big/ c_y^{(t)} .$$

算法 6中将 TS 向量分裂为单精度向量的方法，遵循 Mukunoki 等人为二进制 128 位向量提出的分裂算法。 [2021] 在算法6中，

算法 5 采用 Ozaki 方案进行 TS 精度循环卷积

Require: TS vectors $\boldsymbol{x}_{0:2}, \boldsymbol{y}_{0:2}$
Require: Distinct prime moduli $p_0, p_1 < 2^{32}$
Ensure: TS-vector cyclic convolution $\boldsymbol{z}_{0:2} \simeq \boldsymbol{x}_{0:2} \circledast \boldsymbol{y}_{0:2}$
$\boldsymbol{X}_0^{(0)}, \ldots, \boldsymbol{X}_0^{(k_x-1)}, \boldsymbol{X}_1^{(0)}, \ldots, \boldsymbol{X}_1^{(k_x-1)}, \boldsymbol{c}_x \leftarrow \mathrm{SplitTSAndCalcNTT}\left(\boldsymbol{x}_{0:2}, p_0, p_1\right)$
$\boldsymbol{Y}_0^{(0)}, \ldots, \boldsymbol{Y}_0^{(k_y-1)}, \boldsymbol{Y}_1^{(0)}, \ldots, \boldsymbol{Y}_1^{(k_y-1)}, \boldsymbol{c}_y \leftarrow \mathrm{SplitTSAndCalcNTT}\left(\boldsymbol{y}_{0:2}, p_0, p_1\right)$
for $s = 0, \ldots, k_x - 1$ **do**
  for $t = 0, \ldots, k_y - 1$ **do**
    $\boldsymbol{z}_0'^{(s,t)} \leftarrow \mathrm{NTT}_{p_0}^{-1}\left(\boldsymbol{X}_0^{(s)} \odot \boldsymbol{Y}_0^{(t)}\right)$; $\boldsymbol{z}_1'^{(s,t)} \leftarrow \mathrm{NTT}_{p_1}^{-1}\left(\boldsymbol{X}_1^{(s)} \odot \boldsymbol{Y}_1^{(t)}\right)$
    $\boldsymbol{z}'^{(s,t)} \leftarrow \boldsymbol{z}_0'^{(s,t)} + \left\{\left(\boldsymbol{z}_1'^{(s,t)} - \boldsymbol{z}_0'^{(s,t)}\right) p_0^{-1} \bmod p_1\right\} p_0$
    $\boldsymbol{z}_{0:2}^{(s,t)} \leftarrow \boldsymbol{z}'^{(s,t)} \Big/ \left(\boldsymbol{c}_x^{(s)} \boldsymbol{c}_y^{(t)}\right)$
$\boldsymbol{z}_{0:2} \leftarrow \boldsymbol{0}$
for $s = 0, \ldots, k_x - 1$ **do**
  for $t = 0, \ldots, k_y - 1$ **do**
    $\boldsymbol{z}_{0:2} \leftarrow \mathrm{TSAdd}\left(\boldsymbol{z}_{0:2}, \boldsymbol{z}_{0:2}^{(s,t)}\right)$

算法 6 算法中的分割过程和数论变换计算 5

Require: TS vector $\boldsymbol{x}_{0:2}$
Require: Distinct prime moduli $p_0, p_1 < 2^{32}$
Ensure: $\boldsymbol{X}_0^{(0)}, \ldots, \boldsymbol{X}_0^{(k-1)} \in \mathbb{Z}_{p_0}^n$, $\boldsymbol{X}_1^{(0)}, \ldots, \boldsymbol{X}_1^{(k-1)} \in \mathbb{Z}_{p_1}^n$, $\boldsymbol{c} \in \mathbb{F}_{32}^k$
function SplitTSAndCalcNTT $(\boldsymbol{x}_{0:2}, p_0, p_1)$
  $\rho \leftarrow \log_2\left(u_{32}^{-1}\right) - \left\lfloor \left(\log_2(p_0 p_1/2) - \log_2(n)\right)/2 \right\rfloor$
  $\boldsymbol{r}_{0:2}^{(0)} \leftarrow \boldsymbol{x}_{0:2}$
  $\mu^{(0)} \leftarrow \left\| \boldsymbol{r}_0^{(0)} \right\|_\infty$
  $k \leftarrow 0$
  while $\mu^{(k)} > 0$ **do**
    $\sigma^{(k)} \leftarrow 2^{\lceil \log_2(\mu^{(k)}) \rceil + \rho}$
    $\boldsymbol{x}^{(k)} \leftarrow \mathrm{fl}_{32}\left(\left(\boldsymbol{r}_0^{(k)} + \sigma^{(k)} \boldsymbol{1}_n\right) - \sigma^{(k)} \boldsymbol{1}_n\right)$
    $\boldsymbol{r}_0^{(k+1)}, \boldsymbol{r}_1^{(k+1)} \leftarrow \mathrm{FastTwoSum}\left(\boldsymbol{r}_0^{(k)} - \boldsymbol{x}^{(k)}, \boldsymbol{r}_1^{(k)}\right)$
    $\boldsymbol{r}_1^{(k+1)}, \boldsymbol{r}_2^{(k+1)} \leftarrow \mathrm{FastTwoSum}\left(\boldsymbol{r}_1^{(k)}, \boldsymbol{r}_2^{(k)}\right)$
    $c^{(k)} \leftarrow \left(u_{32}\sigma^{(k)}\right)^{-1}$; $\boldsymbol{x}'^{(k)} \leftarrow c^{(k)} \cdot \boldsymbol{x}^{(k)}$
    $\mu^{(k+1)} \leftarrow \left\| \boldsymbol{r}_0^{(k+1)} \right\|_\infty$
    $\boldsymbol{X}_0^{(k)} \leftarrow \mathrm{NTT}_{p_0}\left(\boldsymbol{x}'^{(k)} \bmod p_0\right)$; $\boldsymbol{X}_1^{(k)} \leftarrow \mathrm{NTT}_{p_1}\left(\boldsymbol{x}'^{(k)} \bmod p_1\right)$
    $k \leftarrow k + 1$
  return $\boldsymbol{X}_0^{(0)}, \ldots, \boldsymbol{X}_0^{(k-1)}, \boldsymbol{X}_1^{(0)}, \ldots, \boldsymbol{X}_1^{(k-1)}, \boldsymbol{c}$

我们首先计算 ρ 使用 α 由（15）得:

$$\rho = \log_2\left(u_{32}^{-1}\right) - \alpha = \log_2\left(u_{32}^{-1}\right) - \left\lfloor \frac{\log_2\left(p_0 p_1/2\right) - \log_2\left(n\right)}{2} \right\rfloor.$$

算法 7 FastTwoSum [Dekker 1971]

要求： $x,\ y \in \mathbb{F}$ 使得 $|x| \geq |y|$
确保： $a,\ b \in \mathbb{F}$ 使得 $a + b = x + y$
1: **函数** FastTwoSum($x,\ y$)
2: $a \leftarrow$ **浮点舍入**$(x + y)$
3: $b \leftarrow \mathbf{fl}(y - (a - x))$
4: **返回** $a\ , b$

然后我们迭代以下方程，直到 $\mu^{(k)} = 0$，以获得 $\boldsymbol{X}_{(0)0}$，……，$\boldsymbol{X}_{(k-1)0}$，$\boldsymbol{X}_{(0)1}$，……，$\boldsymbol{X}_{(k-1)1}$，以及 $\boldsymbol{c}$：

$$
\begin{aligned}
\mu^{(k)} &= \left\| \boldsymbol{r}_0^{(k)} \right\|_\infty \\
\sigma^{(k)} &= 2^{\lceil \log_2(\mu^{(k)}) \rceil + \rho} \\
\boldsymbol{r}_{0:2}^{(k)} &= \begin{cases} \boldsymbol{x}_{0:2} & (k = 0) \\ \boldsymbol{x}_{0:2} - \sum_{j=0}^{k-1} \boldsymbol{x}^{(j)} = \boldsymbol{r}_{0:2}^{(k-1)} - \boldsymbol{x}^{(k-1)} & (k > 0) \end{cases} \\
\boldsymbol{x}^{(k)} &= \mathbf{fl}_{32}\left(\left(\boldsymbol{r}_0^{(k)} + \sigma^{(k)} \mathbf{1}_n \right) - \sigma^{(k)} \mathbf{1}_n \right) \\
c^{(k)} &= \left(u_{32} \sigma^{(k)} \right)^{-1} \\
\boldsymbol{x}'^{(k)} &= c^{(k)} \cdot \boldsymbol{x}^{(k)} \\
\boldsymbol{X}_0^{(k)} &= \mathrm{NTT}_{p_0}\left(\boldsymbol{x}'^{(k)} \bmod p_0 \right) \\
\boldsymbol{X}_1^{(k)} &= \mathrm{NTT}_{p_1}\left(\boldsymbol{x}'^{(k)} \bmod p_1 \right).
\end{aligned}
\tag{16}
$$

对（16）中的三单精度数减去单精度数时，使用 FastTwoSum [Dekker 1971]（算法 7）。FastTwoSum 是一种无误差变换，它将两个浮点数 x和 y（其中 $|x| \geq |y|$）的精确和计算为一对数 （$a,\ b$），满足$s = \mathbf{浮点舍入}$（$x + y$）且 $a + b = x + y$。

在步骤 2 中，对于每个 (s, t)，我们计算 32 位逆 NTT，然后通过中国剩余定理（Garner 形式）进行重构，以获得拆分分量的循环卷积：

$$
\begin{aligned}
\boldsymbol{z}_0'^{(s,t)} &= \mathrm{NTT}_{p_0}^{-1}\left(\boldsymbol{X}_0^{(s)} \odot \boldsymbol{Y}_0^{(t)} \right), \\
\boldsymbol{z}_1'^{(s,t)} &= \mathrm{NTT}_{p_1}^{-1}\left(\boldsymbol{X}_1^{(s)} \odot \boldsymbol{Y}_1^{(t)} \right), \\
\boldsymbol{z}'^{(s,t)} &= \boldsymbol{z}_0'^{(s,t)} + \left\{ \left(\boldsymbol{z}_1'^{(s,t)} - \boldsymbol{z}_0'^{(s,t)} \right) p_0^{-1} \bmod p_1 \right\} p_0, \\
\boldsymbol{z}_{0:2}^{(s,t)} &= \boldsymbol{z}'^{(s,t)} \Big/ \left(c_x^{(s)} c_y^{(t)} \right).
\end{aligned}
$$

最后，在步骤 3 中，我们将所有成对卷积结果相加，得到 $\boldsymbol{z}_{0:2}$：

$$
\boldsymbol{z}_{0:2} = \sum_{s=0}^{k_x-1} \sum_{t=0}^{k_y-1} \boldsymbol{z}_{0:2}^{(s,t)}.
$$

现在我们分析算法5的时间复杂度。在步骤 1 中，我们执行 $2\left(k_x + k_y\right)$ 次 32 位正数论变换。由于每个长度为 n的数论变换需要 $O(n \log n)$时间，因此步骤 1 总共需要 $O((k_x + k_y) n \log n)$

时间。在步骤 2 中，我们执行 $2k_xk_y$ 32位逆数论变换。由于每次逆数论变换需要$O(n\log n)$时间，因此步骤 2 需要 $O\left(k_xk_yn\log n\right)$时间。在步骤 3 中，我们将 k_xk_y个长度为n的向量相加，这需要 $O\left(k_xk_yn\right)$ 时间。因此，步骤 2 是主导项，算法5 的总体时间复杂度为 $O\left(k_xk_yn\log n\right)$。此外，32 位数论变换（包括逆数论变换）的总次数为$2(k_x+k_y)+2k_xk_y=2(k_x+k_y+k_xk_y)$。

4.4 降低计算成本

在算法 6中，我们对 TS 向量进行分割，直到 $\mu^{(k)}$ 变为零。在算法 5中，随后我们计算每对拆分分量的循环卷积，并通过将所有结果项相加得到 $z_{0:2}$ 。然而在实际应用中，足够小的项 $z^{(s,t)}{}_{0:2}$可以从总和中省略，而几乎不会损失精度。因此，为了减少算法 6 中的正向数论变换次数以及算法 5中的逆数论变换次数，我们对分割数设置上界，并提前终止分割过程。Ozaki 方案在矩阵乘法中也采用了类似策略，以减少矩阵乘积的数量。

为了进一步减少逆数论变换的次数，考虑满足条件的索引 s， t， s'， t' satisfying $c_{(s)x}\ c_{(t)y}=c_{(s')x}\ c_{(t')y}$。在这种情况下，以下恒等式成立：

$$
\begin{aligned}
z_{0:2}^{(s,t)}+z_{0:2}^{(s',t')} &= z'^{(s,t)}\Big/\left(c_x^{(s)}c_y^{(t)}\right)+z'^{(s',t')}\Big/\left(c_x^{(s')}c_y^{(t')}\right)\\
&=\left(z'^{(s,t)}+z'^{(s',t')}\right)\Big/\left(c_x^{(s)}c_y^{(t)}\right)\\
&=\left\{\left(z_0'^{(s,t)}+z_0'^{(s',t')}\right)p_1p_1^{-1}+\left(z_1'^{(s,t)}+z_1'^{(s',t')}\right)p_0p_0^{-1}\right\}\Big/\left(c_x^{(s)}c_y^{(t)}\right)\\
&=\left\{\left(\mathrm{NTT}_{p_0}^{-1}\left(X_0^{(s)}\odot Y_0^{(t)}\right)+\mathrm{NTT}_{p_0}^{-1}\left(X_0^{(s')}\odot Y_0^{(t')}\right)\right)p_1p_1^{-1}\right.\\
&\quad\left.+\left(\mathrm{NTT}_{p_1}^{-1}\left(X_1^{(s)}\odot Y_1^{(t)}\right)+\mathrm{NTT}_{p_1}^{-1}\left(X_1^{(s')}\odot Y_1^{(t')}\right)\right)p_0p_0^{-1}\right\}\Big/\left(c_x^{(s)}c_y^{(t)}\right)\\
&=\left\{\left(\mathrm{NTT}_{p_0}^{-1}\left(X_0^{(s)}\odot Y_0^{(t)}+X_0^{(s')}\odot Y_0^{(t')}\right)\right)p_1p_1^{-1}\right.\\
&\quad\left.+\left(\mathrm{NTT}_{p_1}^{-1}\left(X_1^{(s)}\odot Y_1^{(t)}+X_1^{(s')}\odot Y_1^{(t')}\right)\right)p_0p_0^{-1}\right\}\Big/\left(c_x^{(s)}c_y^{(t)}\right).
\end{aligned}
$$

因此，当 $c(s)x\ c(t)y=c(s')x\ c(t')y$ 时，我们可以在 NTT 域中先累加 $X(s)0\odot Y(t)0$ 和 $X(s')0\odot Y(t')0$，以及 $X(s)1\odot Y(t)1$ 和 $X(s')1\odot Y(t')1$，再进行逆数论变换，从而减少逆数论变换的次数，而不必分别计算 $z^{(s,t)}0{:}2$ 和 $z(s',t')0{:}2$ 后再将它们相加。

例如，当 $s+t=s'+t'$ 时，等式 $c(s)x\ c(t)y=c(s')x\ c(t')y$ 成立。具体而言，如果分裂过程没有间隙，那么 $c_x^{(1)}=2^{-\alpha}c(0)x$ 和 $c(2)x=2^{-2\alpha}c(0)x$，类似地，$c(1)y$ 和 $c(2)y$ 也成立。在这种情况下，令 $s+t=s'+t'=k$，可得 $c(s)x\ c(t)y=c(s')x\ c(t')y=2^{-k\alpha}c(0)x\ c(0)y$。

另一方面，在逆数论变换之前累加项，可能会导致整数循环卷积结果超出范围 $[-p_0p_1/2,\ p_0p_1/2)$，此时中国剩余定理重构将失败。为避免这种情况，我们对逆数论变换之前累加的项数设定上界 L，并选择 α，使得即使最多进行L 次加法，重构仍然有效：

$$\alpha=\left\lfloor\frac{\log_2(p_0p_1/2)-\log_2(Ln)}{2}\right\rfloor.$$

因此，我们使用 α这一数值进行分裂。

算法8 给出了算法5 的一个修改版本，其中融入了这种数论变换域累加策略。这里，SplitTSAndCalcNTT($x_{0:2}$， p_0， p_1， L) 表示分割过程

算法 8 采用 Ozaki 方案和数论变换域累加的 TS 精度循环卷积

需要： TS 向量 $\boldsymbol{x}_{0:2}$，$\boldsymbol{y}_{0:2}$
需要： 互异素数模数 p_0，$p_1 < 2^{32}$
需要： 累加上限 $L \in \mathbb{N}$
确保： TS 向量 $z_{0:2} \simeq \boldsymbol{x}_{0:2} \circledast \boldsymbol{y}_{0:2}$
1: $\boldsymbol{X}_{(0)}0, \ldots, \boldsymbol{X}_{(k_x-1)}0, \boldsymbol{X}_{(0)}1, \ldots, \boldsymbol{X}_{(k_x-1)}1, \boldsymbol{c}_x \leftarrow$ SplitTSAndCalcNTT $(\boldsymbol{x}_{0:2}, p_0, p_1, L)$
2: $Y_0^{(0)}, Y_0^{(k_y-1)}, Y_1^{(0)}, \ldots, Y_1^{(k_y-1)}$ $\boldsymbol{c}_y \leftarrow$ SplitTSAndCalcNTT $(\boldsymbol{y}_{0:2}, p_0, p_1, L)$
3: $z_{0:2}^0, Z_0', Z_1' \leftarrow \mathbf{0}$
4: $c \leftarrow c_x^{(k_x-1)} \cdot c_y^{(k_y-1)}$
5: $l \leftarrow 0$
6: **对于** $k = k_x + k_y - 2, \cdots, 0$ **do**
7: **对于** $s =$ 最大 $(0, k - k_y + 1), \ldots,$ 最小 $(k_x - 1, k)$ **执行**
8: $t \leftarrow k - s$
9: **如果** $l \geq L$ 或 $c \neq c_x^{(s)} \cdot c_y^{(t)}$ **then**
10: $z_0' \leftarrow$ 数论变换$_{p_0}^{-1}(Z_0')$; $z_1' \leftarrow$ 数论变换$_{p_1}^{-1}(Z_1')$
11: $z' \leftarrow z_0' + \left\{\left(z_1' - z_0'\right) p_0^{-1}\right.$模 $\left.p_1\right\} p_0$
12: $z_{0:2}^{(k)} \leftarrow z'/c$
13: $z_{0:2} \leftarrow$ TSAdd $\left(z_{0:2}, z_0^{(k)}{}_{:2}\right)$
14: $Z_0', Z_1' \leftarrow \mathbf{0}$
15: $c \leftarrow c_x^{(s)} \cdot c_y^{(t)}$
16: $l \leftarrow 0$
17: $Z_0' \leftarrow \left(Z_0' + X^{(s)0} \odot Y_{(t}{}_0\right)$模 p_0
18: $Z_1' \leftarrow \left(Z_1' + X^{(s)1} \odot Y_{(t}{}_1\right)$模 p_1
19: $l \leftarrow l + 1$
20: $z_0' \leftarrow$ 数论变换$_{p_0}^{-1}(Z_0')$; $z_1' \leftarrow$ 数论变换$_{p_1}^{-1}(Z_1')$
21: $z' \leftarrow z_0' + \left\{\left(z_1' - z_0'\right) p_0^{-1}\right.$模 $\left.p_1\right\} p_0$
22: $z_{0:2}^{(0)} \leftarrow z'/c$
23: $z_{0:2} \leftarrow$ TSAdd $\left(z_{0:2}, z_0^{(0)}{}_{:2}\right)$

使用

$$\rho = \log_2\left(u_{32}^{-1}\right) - \alpha = \log_2\left(u_{32}^{-1}\right) - \left\lfloor \frac{\log_2(p_0 p_1/2) - \log_2(Ln)}{2} \right\rfloor$$

代替 ρ 在算法6中。

在算法 8 中，对于固定的 k，内层循环体（第 7 行）最多执行 $\min(k_x, k_y)$ 次。因此，如果我们将分割数上限设为 K，并设置累加上限使其满足 $L \geq K$，那么对于每个 k，逆数论变换的次数都可以减少到两次。此外，当 $k_x = k_y = K$ 时，外层循环（第 6 行）执行 $2K-1$ 次，因此逆数论变换的总次数至少为 $2(2K-1) = 4K-2$。加上分裂过程中所需的 $4K$ 次正向数论变换，算法 8 中数论变换的总次数（包括逆数论变换）至少为 $8K - 2$。在这种情况下，最佳时间复杂度为 $O(Kn\log n)$。

将 $L > 1$ 设置为 α，与 $L = 1$相比，约可减少 $\log_2(L)/2$ 个比特。每个分裂分量最多携带 α 比特的信息。当 K为有限值时，将 $L > 1$ 设置为可减少比特

每个分裂分量的宽度，相较于 $L = 1$ 在相同 K下，从而可能降低整体可达到的精度。将 $K = \infty$ 设置为可避免这种退化，但分割数可能变大，导致 NTT 的数量增加。不过，算法 8中的数论变换域累加策略可以缓解这一问题。

4.5 使用 32 位数论变换的双精度 Bluestein 快速傅里叶变换

我们通过计算算法 4 中的 TS 精度复数循环卷积来实现基于 TS 算术的双精度布鲁斯坦快速傅里叶变换8，该算法仅依赖 32 位数论变换。

在算法 4中，我们计算以下 TS 精度复数循环卷积：

$$\begin{aligned}\boldsymbol{x}'_{0:2} \circledast \boldsymbol{\omega}^*_{0:2} = &\operatorname{Re}\left(\boldsymbol{x}'_{0:2}\right) \circledast \operatorname{Re}\left(\boldsymbol{\omega}^*_{0:2}\right) - \operatorname{Im}\left(\boldsymbol{x}'_{0:2}\right) \circledast \operatorname{Im}\left(\boldsymbol{\omega}^*_{0:2}\right) \\ &+ i\left(\operatorname{Im}\left(\boldsymbol{x}'_{0:2}\right) \circledast \operatorname{Re}\left(\boldsymbol{\omega}^*_{0:2}\right) + \operatorname{Re}\left(\boldsymbol{x}'_{0:2}\right) \circledast \operatorname{Im}\left(\boldsymbol{\omega}^*_{0:2}\right)\right).\end{aligned}$$

由于右侧由四个实数循环卷积组成，因此可以通过调用算法 8 四次来完成此计算。然而，直接实现会产生冗余的分裂：每个 TS 向量都要分裂两次，因此总共需要调用算法 6八次。为避免这种冗余，我们缓存分裂结果，并在多次调用算法 8时重复使用，从而将分裂操作次数从八次减少到四次。

采用这种缓存策略后，我们现在估算算法 4 所需的 32 位变换总数。我们将四个 TS 向量 实部 $\left(\boldsymbol{x}'_{0:2}\right)$、虚部 $\left(\boldsymbol{x}'_{0:2}\right)$、实部 $\left(\boldsymbol{\omega}^*_{0:2}\right)$和 虚部$\left(\boldsymbol{\omega}^*_{0:2}\right)$分割，并总共调用算法 6 四次。设 K 为分割数的上界。每次调用算法 6 最多执行 $2K$ 次 32 位正数论变换（每个模数对应一次），因此 32 位正数论变换的总次数至多为 $8K$。

算法 8 中的逆数论变换次数取决于数论变换域累加界L （应用逆数论变换前累加项数的最大值）。在最坏情况下，当 $L = 1$ （不进行数论变换域累加）时，一次实数循环卷积最多需要$2K^2$ 次逆 32 位数论变换。由于一次复数循环卷积由四次实数循环卷积构成，因此逆 32 位数论变换的总次数至多为 $8K^2$。因此，算法 4总共最多执行 $8K + 8K^2$ 32 次比特变换（包括正变换和逆数论变换）。

在最佳情况下，当 $L \geq K$ 时，调用一次算法 8 只需要 $4K - 2$ 次逆向 32 位数论变换，因此总计至少需要 $8K + 4(4K - 2) = 24K - 8$ 次变换。此外，由于旋转因子向量 复部 $\left(\boldsymbol{\omega}^*_{0:2}\right)$ 和 虚部 $\left(\boldsymbol{\omega}^*_{0:2}\right)$ 与输入无关，其分裂以及相关的 $4K$正向数论变换可以离线预先计算，从而将运行时所需的变换次数减少到最佳情况下至少 $20K - 8$ 次，最坏情况下至多 $4K + 8K^2$ 次。

总之，由于每次长度为 n 的 32 位数论变换需要 $O(n \log n)$时间，而算法 4的总体时间复杂度在最佳情况下为 $O(Kn \log n)$，最坏情况下为 $O(K^2 n \log n)$。

5 实验

5.1 精度评估设置

本节中的所有精度实验均采用以下通用设置。对于所提方法，我们使用了素数模数 $p_0 = 2$、130、706、433 $< 2^{31}$ 和 $p_1 = 2$、113、929、217 $< 2^{31}$。我们计算了长度从 $n = 2^{10}$ 到 2^{18}的二的幂次快速傅里叶变换。

输入数据以双精度生成，具体如下

$$(\text{rand} - 0.5) \cdot \exp\left(\phi \cdot \text{randn}\right), \tag{17}$$

其中，rand 是在 $(0, 1]$ 上均匀分布的随机变量，而 randn 是标准正态随机变量。参数 ϕ 控制指数的范围：当 $\phi = 0$ 时，输入服从

在 $(-0.5, 0.5]$ 上的均匀分布；随着 ϕ 增大，更多取值聚集在零附近，而指数的分布范围也随之增大。我们评估了三种设置： $\phi = 0$、1.0 和 4.0。

我们以使用 MPFR 在 106 比特精度下计算的快速傅里叶变换结果（双精度有效数的 53 比特的两倍）作为参考，评估了最大相对误差和定义如下的相对 ℓ_2 误差

$$\max_{0 \le j < n} \left\{ \max \left(\left| \frac{\mathrm{Re}\,(y'(j)) - \mathrm{Re}\,(y(j))}{\mathrm{Re}\,(y(j))} \right|, \left| \frac{\mathrm{Im}\,(y'(j)) - \mathrm{Im}\,(y(j))}{\mathrm{Im}\,(y(j))} \right| \right) \right\},$$

and

$$\frac{\|\boldsymbol{y}' - \boldsymbol{y}\|_2}{\|\boldsymbol{y}\|_2},$$

其中， $\boldsymbol{y} \in \mathbb{C}^n$ 是 MPFR（106 比特）的参考输出， $\boldsymbol{y}' \in \mathbb{C}^n$ 是待评估快速傅里叶变换方法的输出。

5.2 与其他双精度快速傅里叶变换的精度比较

我们将所提出的双精度布鲁斯坦快速傅里叶变换（算法 4 使用算法 8）的精度与其他双精度快速傅里叶变换实现进行了比较。作为基准，我们考察了以下五种方法：

1. 使用 FFTW 的双精度快速傅里叶变换 [Frigo 和 Johnson 2005]
2. 使用双精度算术的 Stockham 快速傅里叶变换 3. 使用双精度算术的布鲁斯坦快速傅里叶变换 4. 使用 TS 算术的 Stockham 快速傅里叶变换 5. 使用 TS 算术的布鲁斯坦快速傅里叶变换

对于方法 4 和 5，在计算前先将输入从双精度转换为 TS 精度，并将结果舍入回双精度，与所提方法相同。对于所提方法，我们将分割数的上界设为 $K = \infty$ （无上限），并将数论变换域累加界设为 $L = 1$ （不进行数论变换域累加）。

图 2 显示了最大相对误差。虽然 Ozaki 方案往往能够降低矩阵乘法中的最大相对误差 [Ozaki 等人 2012], 但所提方法并不能始终使快速傅里叶变换达到最小的最大相对误差。

图 3 展示了相对误差。Stockham 快速傅里叶变换的相对误差小于布鲁斯坦快速傅里叶变换，这可能是因为 Stockham 快速傅里叶变换所需的算术运算更少，因此累积的舍入误差也更小。对于给定的快速傅里叶变换算法，基于 TS 的实现所产生的相对误差小于相应的双精度实现，这可归因于 TS 格式具有更长的有效数。此外，所提方法在大多数情况下都能达到最小的相对误差；唯一的例外是 $(\phi, n) = (4.0, 2_{10})$ 和 $(4.0, 2_{13})$，在这两种情况下，基于 TS 的 Stockham 快速傅里叶变换的精度略高。此外，与基于 TS 的布鲁斯坦快速傅里叶变换相比，所提方法显著降低了相对误差，二者的区别仅在于循环卷积例程。这些结果表明，将 Ozaki 方案应用于循环卷积阶段能够有效降低快速傅里叶变换的相对误差。

对于基线双精度快速傅里叶变换，相对误差往往会随 n增大。所提方法没有表现出这种趋势。通常，增大 n 会增加快速傅里叶变换中的运算次数，并加剧舍入误差的累积。相比之下，所提方法在算法4的步骤 3 中对循环卷积应用 Ozaki 方案，而循环卷积占据了主要计算成本；拆分后的循环卷积可以精确计算，因此与其他方法不同，这很可能抑制了相对误差的增长。

在此设置下， $\boldsymbol{x}'$ 和 $\boldsymbol{\omega}^*$ 的分割数，以及 32 位数论变换和逆数论变换的数量分别如图4 和5所示。随着 ϕ的增加， $\boldsymbol{x}'$ 的分割数往往会增加

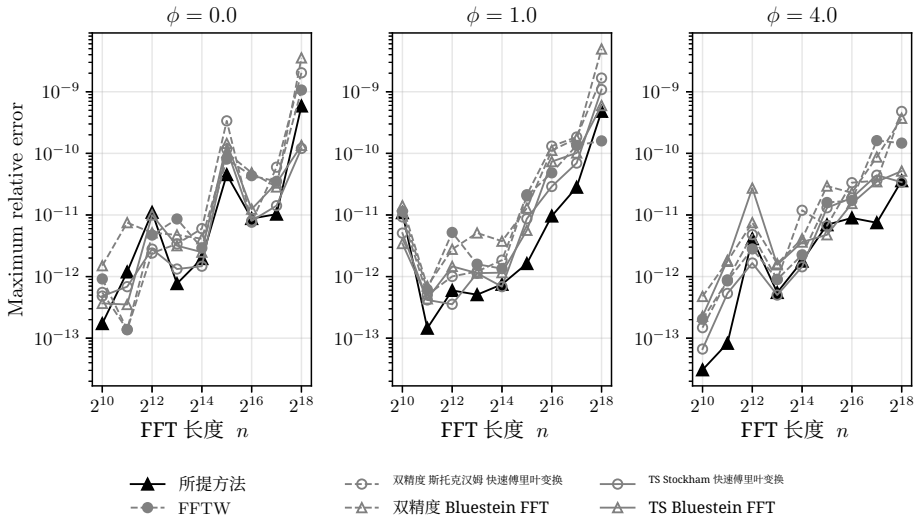

图 2. 所提出的双精度 Bluestein 快速傅里叶变换（$(K, L) = (\infty, 1)$）与其他双精度快速傅里叶变换实现的最大相对误差比较。

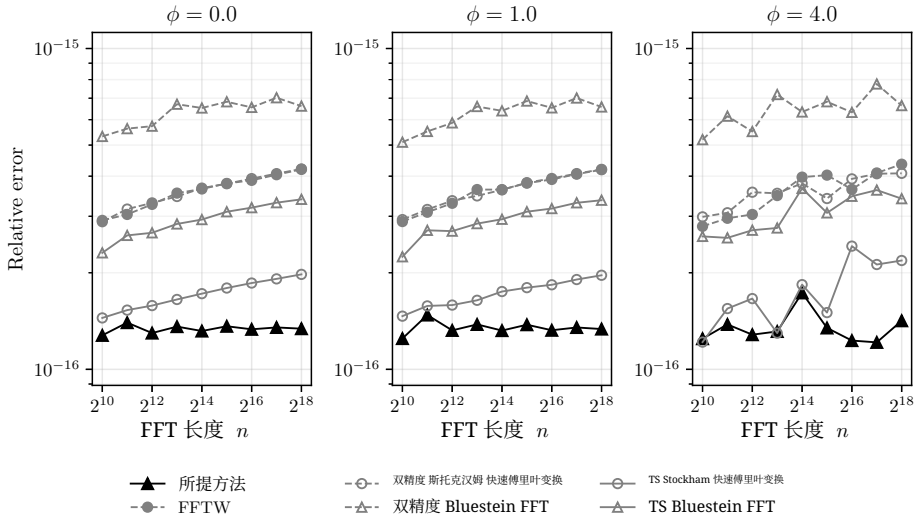

图 3. 所提出的双精度 Bluestein 快速傅里叶变换（$(K, L) = (\infty, 1)$）与其他双精度快速傅里叶变换实现的相对误差比较。

且 n 增加。相比之下， $\boldsymbol{\omega}^*$ 的分割数随 n 增加，但与 ϕ无关，因为 $\boldsymbol{\omega}^*$ 不依赖于输入。32 位数论变换和逆数论变换的总数也随 ϕ和 n增加；当参数为（ϕ， n）时达到最大值，分别为（4。0，2_{17}）和（4。0，2_{18}），总数为 268。

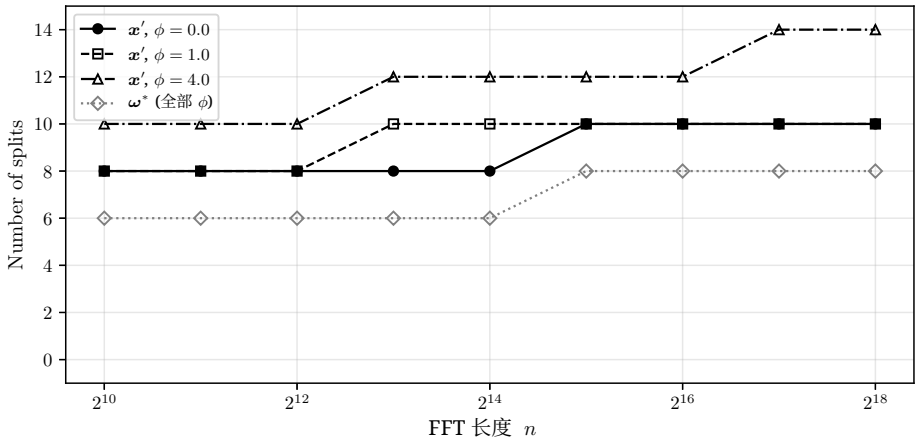

图 4. 所提出的双精度布鲁斯坦快速傅里叶变换中 x' 和 ω^* 的分割数（$(K, L) = (\infty, 1)$）。

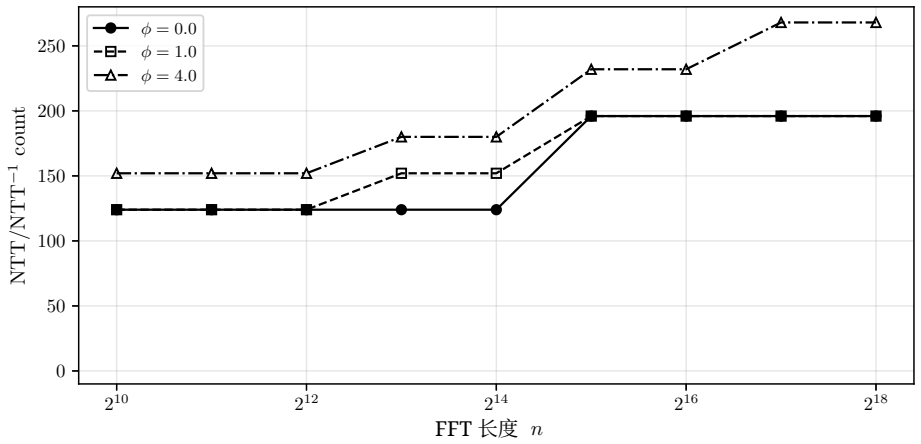

图 5. 所提出的双精度布鲁斯坦快速傅里叶变换中的 32 位数论变换和逆数论变换总数（$(K, L) = (\infty, 1)$）。

5.3 上界对分割数的影响

接下来，我们通过改变分割数的上界，评估了最大相对误差和相对误差K，具体取值为 ∞、1、2、3。结果分别如图 6 和 7所示。作为比较，我们还列出了基于 TS 的 Stockham 快速傅里叶变换的结果。随着 K 增大，两种误差度量均会减小。对于 $K = 3$，我们获得了与上界为$K = \infty$ 的所提方法以及基于 TS 的 Stockham 快速傅里叶变换相当的精度。

图 8 展示了范围 $[10^{-16}, 10^{-15}]$ 在图 7中的放大视图。如图所示，对于 n 和 ϕ的大多数组合， $K = 3$ 的相对误差与使用 $K = \infty$时的相当，且小于基于 TS 的 Stockham 快速傅里叶变换的相对误差。对于 $K = 3$，32 位数论变换和逆数论变换的总数为 96，约为

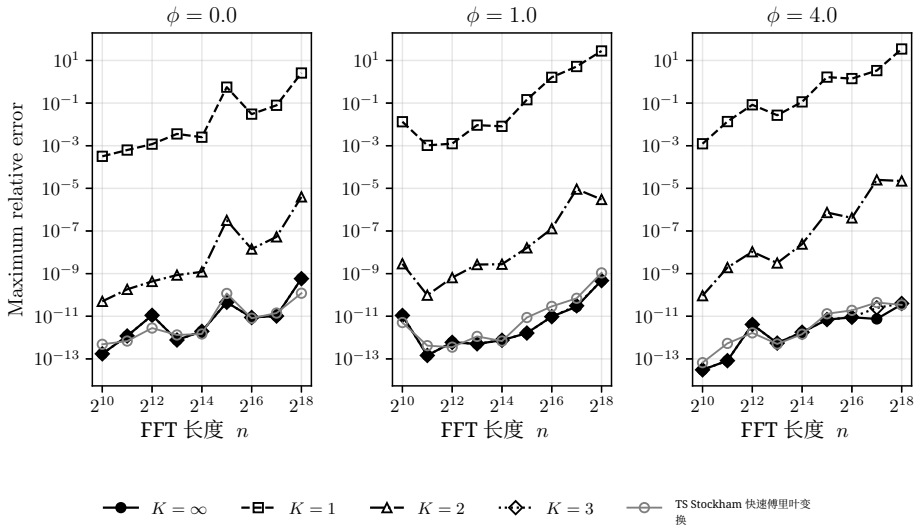

图 6. 所提出的双精度布鲁斯坦快速傅里叶变换（$L = 1$）在分割数上界 K 取不同值时的最大相对误差。

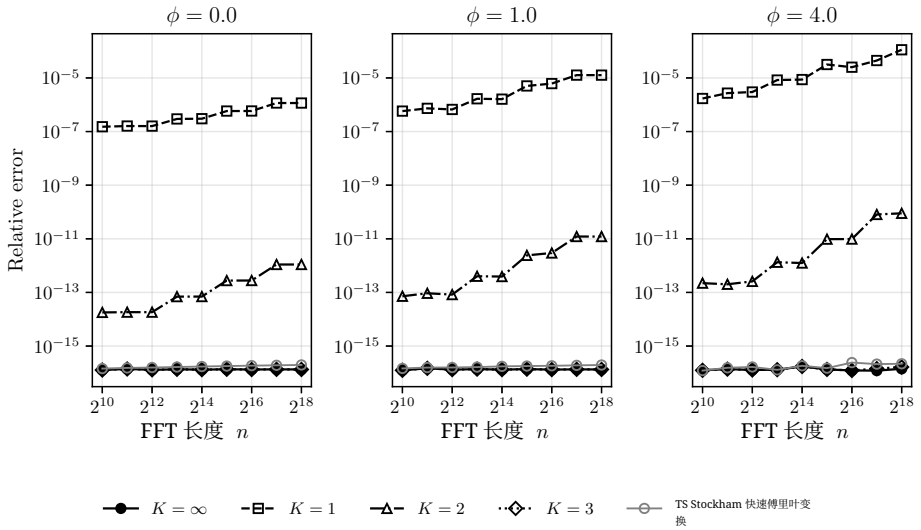

图 7. 所提出的双精度布鲁斯坦快速傅里叶变换（$L = 1$）在分割数上界 K取不同值时的相对误差。

$K = \infty$。这表明，将分割数的上界 K 设得较小，能够在保持精度的同时有效降低计算成本。

5.4 逆 NTT 前进行数论变换域累加的影响

我们评估了在逆数论变换之前进行数论变换域累加，如何减少 32 位逆数论变换的次数。图9 比较了以下设置的 32 位数论变换和逆数论变换总次数：

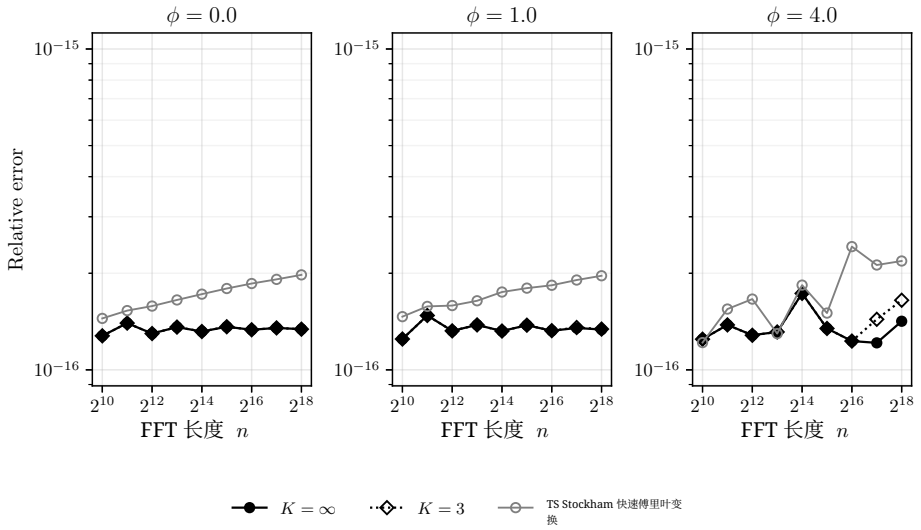

图 8. 图 7 （$L = 1$）在范围 $[10^{-16}, 10^{-15}]$。

$(K, L) = (3, 3)$，以及 $(K, L) = (\infty, 1)$ 和 $(3, 1)$。对于 $\phi = 0$ 和 1.0，设置 $(K, L) = (3, 3)$ 在所有快速傅里叶变换规模下的 32 位数论变换和逆数论变换总次数均为 64。这与将 $K = 3$ 代入 $24K - 8$后得到的最小次数 64 一致。相比之下，$(K, L) = (\infty, 1)$ 在 $\phi = 0$ 和 1.0 时的最大次数为 196，因此总次数降至约 33%。对于 $\phi = 4.0$，$(K, L) = (3, 3)$ 的最大次数为 80；虽然未达到最小次数 64，但与 $(K, L) = (\infty, 1)$ 的最大次数 268 相比，仍降至约 30%。

图 10 和图 11 分别比较了 $(K, L) = (3, 1)$ 和 $(3, 3)$ 的最大相对误差和相对误差。对于 $\phi = 0$，未观察到显著的误差差异；而在 $(\phi, n) = (1.0, 218)$ 和 $(4.0, 218)$ 处，$(K, L) = (3, 3)$ 的相对误差大于 $(3, 1)$。对于 $n = 2^{17}$，$(K, L) = (3, 1)$ 和 $(3, 3)$ 均产生 $\alpha = 21$ 比特；而对于 $n = 2^{18}$，$(K, L) = (3, 1)$ 仍产生$\alpha = 21$ 比特，但 $(3, 3)$ 产生 $\alpha = 20$ 比特，导致总精度损失为 $1 \times 3 = 3$ 比特。

图 11 揭示了实现精确计算所需的精度。对于 $\phi = 1.0$ 和 4。0，在 $\alpha \geq 21$ 位且 $L = 3$ $(n \leq 2^{17})$ 的范围内，$L = 3$ 和 $L = 1$ 的相对误差几乎相同；然而，一旦 α 降至 20 位（$n = 2^{18}$），就会出现明显差异。这一观察表明，所需的总位宽至少为 $21 \times 3 = 63$ 位，适用于 $\phi = 1.0$ 和 4。0。

这些结果表明，对于指数范围较窄的输入（例如 $\phi = 0$），数论变换域累加是有效的：每次逆数论变换调用之前，都会稳定地累加最多数量的项，且精度损失很小。然而，对于指数范围较宽的输入，数论变换域累加导致的 α 下降可能会对精度产生不利影响。此外，由于这种情况下很少达到最大累加次数，逆数论变换次数的减少也有限。

5.5 执行时间

我们测量了所提出的双精度 Bluestein 快速傅里叶变换的执行时间，其中 $(K, L) = (3, 3)$ 以及由 (17) 生成的输入数据，使用 $\phi = 0$。FFT 长度范围为 $n = 2^{10}$ 至 2^{18}。评估平台见表 1。包括 32 位数论变换在内的整个实现

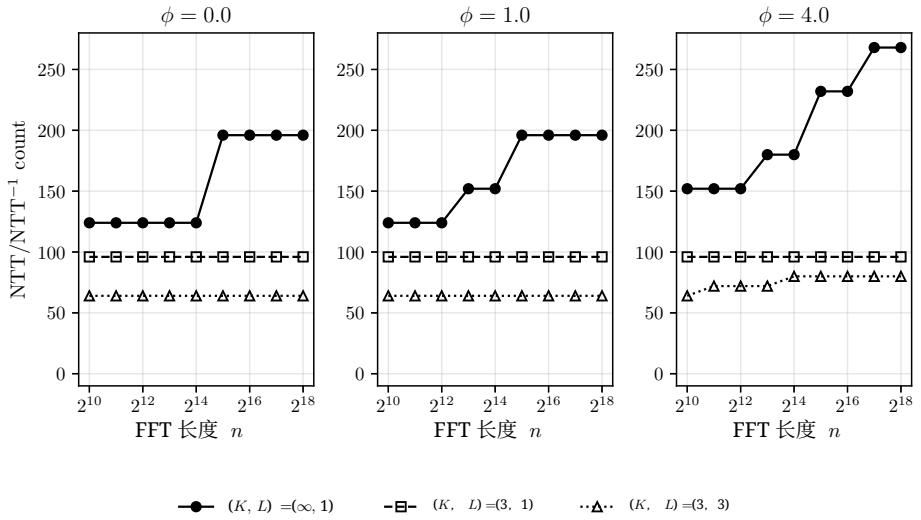

图 9. 在所提出的双精度 Bluestein 快速傅里叶变换中，逆数论变换之前进行数论变换域累加对 32 位数论变换和逆数论变换总次数的影响。

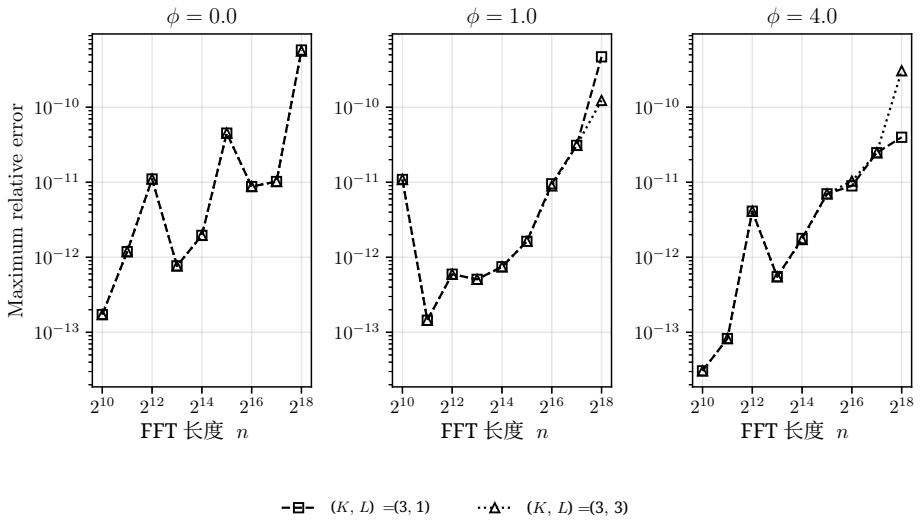

图 10. 所提出的双精度 Bluestein 快速傅里叶变换 ($K = 3$) 在逆数论变换之前采用不同数论变换域累加界 L 时的最大相对误差。

以及 TS 算术，均从头开始使用 C 编写++。32 位数论变换基于斯托克汉姆算法。模乘法采用 Montgomery乘法 [Montgomery1985] 和 Shoup乘法 [Färnqvist 2005]。我们使用 OpenMP 和 48 个线程并行化彼此独立的循环迭代。我们没有使用手写的 SIMD内在函数或显式的缓存优化；向量化完全依赖编译器自动向量化。在测量单次运行的执行时间之前，我们进行了 10 次预热运行。

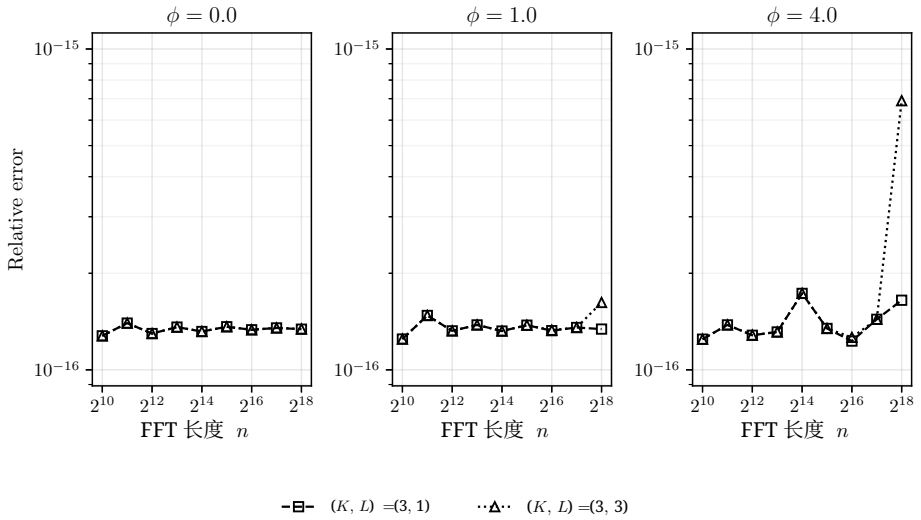

图 11. 所提出的双精度 Bluestein 快速傅里叶变换（$K = 3$）在逆数论变换之前，针对不同数论变换域累加界 L 的相对误差。

表 1. 执行时间对比的实验环境。

CPU	Intel Xeon Platinum 8468
核心/线程	48 核心 / 48 线程
内存	128 GB DDR5-4800
编译器	英特尔(R) oneAPI C++ 编译器 2025.0.4
编译器标志	-O3 -xHOST -qopenmp -fma
FFTW 版本	3.3.10

图 12 展示了执行时间。对于所有 n，32 位数论变换/数论变换$^{-1}$的调用次数均为 52 次（不包括为分裂旋转因子向量所需的正向数论变换 $\boldsymbol{\omega}^*$，该向量可以预先离线计算）。图中还列出了单次 32 位数论变换/数论变换$^{-1}$的执行时间，该时间由数论变换总时间除以 52 得出；此外，还列出了 FFTW 计算的双精度和单精度快速傅里叶变换的执行时间以供参考（PATIENT 模式，48 个线程，编译时启用了 AVX-512 支持）。FFTW 的执行时间是预热运行 10 次后，对 52 次调用取平均值所得。图 13 展示了所提方法在各个 FFT 长度下的执行时间分解。

所提方法的执行时间约为 FFTW 双精度快速傅里叶变换的 107 倍（$n = 2^{15}$）至 1315 倍（$n = 2^{10}$）；在 $n = 2^{15}$ 附近，该比值最小，并且随着 n变小或变大而增加。

图 13 显示，32 位数论变换/数论变换$^{-1}$ 约占所有 FFT 长度下总执行时间的 80%，而其余操作，即中国剩余定理重构、分量拆分卷积、逐元素乘积、分裂和数据类型转换，各自所占的比例都相对较小。

此外，32 位数论变换/NTT^{-1} 每次调用的执行时间约为 FFTW 单精度快速傅里叶变换的 31 倍（$n = 2^{10}$）至 2.5 倍（$n = 2^{15}$）。虽然不能

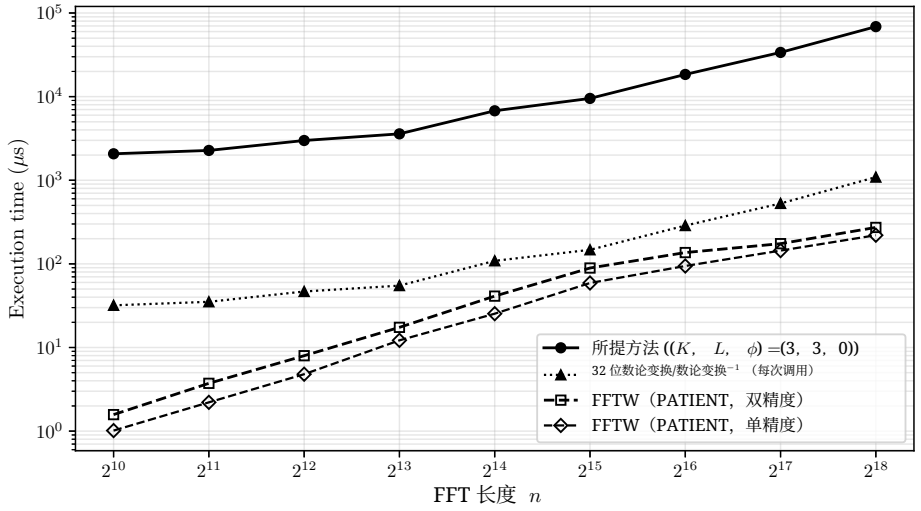

图 12. 所提出的双精度布鲁斯坦快速傅里叶变换（(K, L, ϕ) = (3, 3, 0)）与 FFTW 的双精度和单精度快速傅里叶变换（PATIENT 模式，48 个线程）的执行时间比较，以及单次 32 位数论变换/数论变换$^{-1}$的每次调用执行时间。

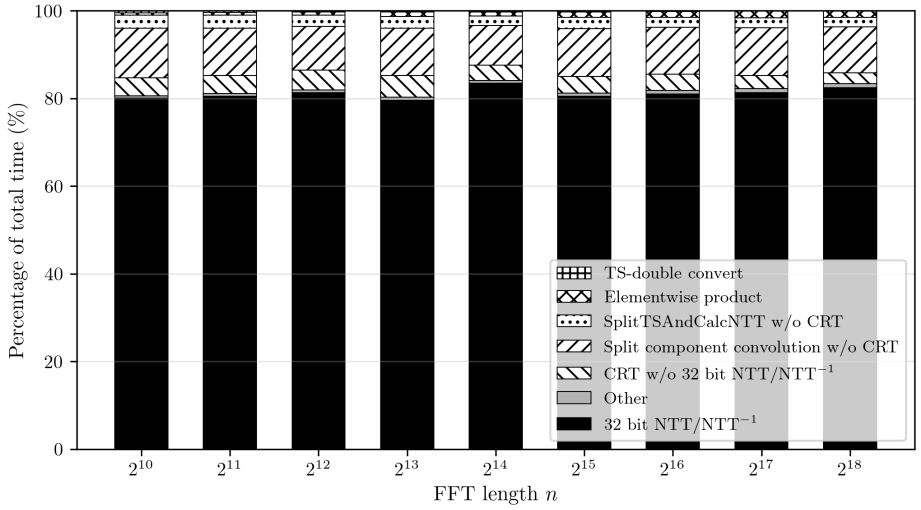

图 13. 所提出的双精度布鲁斯坦快速傅里叶变换（(K, L, ϕ) = (3, 3, 0)）的执行时间分解。

保证 32 位数论变换能够优化到 FFTW 单精度快速傅里叶变换的水平，但如果假设确实可以达到这一水平，那么数论变换在总执行时间中的占比将从约 80% 降至 11%（$n = 2^{10}$）至 62% （$n = 2^{15}$）。尤其是对于较小的 n，数论变换将不再主导总执行时间；即使对于较大的 n，非数论变换部分也不容忽视。这表明，要缩短总体执行时间，还需要优化非数论变换部分。由于数论变换计算的时间复杂度

为 $O(Kn\log n)$，而非数论变换部分为 $O(Kn)$，两者之比仅为$O(\log n)$，这证实了非数论变换部分的开销不可忽略。

6 讨论

现在，我们分析所提方法在什么条件下能够实现与基线双精度快速傅里叶变换相当的执行时间。令 T_{prop} 表示所提方法的总执行时间， T_{NTT} 表示单次 32 位数论变换或数论变换$^{-1}$调用的执行时间，C_{NTT} 表示运行时数论变换/数论变换$^{-1}$ 的调用总次数， T_{base} 表示基线双精度快速傅里叶变换的执行时间。令 $P_{\text{NTT}} = C_{\text{NTT}} \cdot T_{\text{NTT}}/T_{\text{prop}}$ 表示 T_{prop}中用于数论变换计算的比例。根据此定义， $T_{\text{prop}} = C_{\text{NTT}} \cdot T_{\text{NTT}}/P_{\text{NTT}}$ ，且条件$T_{\text{prop}} < T_{\text{base}}$ 等价于

$$\frac{T_{\text{base}}}{T_{\text{NTT}}} > \frac{C_{\text{NTT}}}{P_{\text{NTT}}}. \tag{18}$$

在使用 $(K, L) = (3, 3)$ 进行的实验中，我们有 $C_{\text{NTT}} = 52$ 和 $P_{\text{NTT}} \approx 0.8$，因此 (18) 要求$T_{\text{base}}/T_{\text{NTT}} > 65$。如果将 NTT 优化到与 FFTW 的单精度快速傅里叶变换一样快， P_{NTT}将降至约 0.62（$n = 2^{15}$）至 0.11（$n = 2^{10}$），从而将阈值提高到 $C_{\text{NTT}}/P_{\text{NTT}}$ ≈84–470。

在近期的消费级GPU上，双精度浮点吞吐量通常为 1/32或 1/64 的单精度和 32 位整数吞吐量，乍看之下似乎接近所需的 65 倍比率。然而，由于快速傅里叶变换和数论变换本质上都是内存受限算法，算术吞吐量的差距并不会直接转化为快速傅里叶变换和数论变换执行时间的同比例差距，因此即使在此类硬件上，(18) 也难以满足。

尽管本文重点讨论二次幂长度快速傅里叶变换，但 Bluestein 快速傅里叶变换适用于任意长度，因此所提方法可自然推广到任意长度快速傅里叶变换。对于任意长度的 Bluestein 快速傅里叶变换，即使不采用所提方法，也需要调用三次二次幂长度快速傅里叶变换。这实际上使式（1/3）右侧的值乘以18。代入我们实验中的数值后，所需比率 $T_{\text{base}}/T_{\text{NTT}}$ 约为 21。因此，对于任意长度快速傅里叶变换，所提方法更有可能实现具有竞争力的执行时间。

最近，Ozaki 等人 [2025] 提出了 Ozaki 方案的一种变体，该变体在拆分阶段使用中国剩余定理，将拆分分量乘积的数量从 $O(k^2)$ 降至 $O(k)$，其中 k 为分割数。如果这项技术能够扩展到循环卷积，就可以进一步减少我们方法中的数论变换调用次数。

Ozaki 方案在矩阵-矩阵乘积方面的一项关键优势是可移植性：它依赖基础线性代数子程序（BLAS），这是一种标准化且广泛可用的库，几乎在每个平台上都提供针对不同精度优化的厂商实现。相比之下，所提方法需要将 NTT 作为核心构建模块。与矩阵乘法和快速傅里叶变换不同，后两者拥有广泛可用的标准化库（如 BLAS 和 FFTW），而目前还没有可比的 NTT 库能针对不同位宽和架构提供优化实现。不过，NTT 已广泛用作基于格的密码学和全同态加密中的核心运算，高效的 NTT 实现也正在积极开发中。具体而言，已有多种基于 GPU 的实现 [Ozcan等人，2025年；Ji等人，2025年；Yin等人，2025年；Sugizaki 和 Takahashi 2026]。由于所提方法以模块化方式使用 NTT，因此，如果某个平台在不同算术精度下的性能差异显著，且能以高吞吐精度高效实现 NTT，那么只需调整分割数，该方法便可适用于各种精度。

利用此类性能差异的一种自然方法，是直接使用较低精度的 NTT，例如 16 位或 8 位整数，就像 Ozaki 方案在矩阵乘法中所做的那样。然而，NTT 的最大长度由模数大小决定，而较小的模数会严重限制可实现的 NTT 长度，从而限制该方法在实际中的适用性。近期，Yin等人 [2025] 以及 Sugizaki 和 Takahashi [2026]提出了其他方法，旨在利用较低精度的算术单元进行 NTT 计算，同时缓解变换长度的限制。由于我们的方法将目标精度快速傅里叶变换计算归约为 NTT 调用，因此将其与此类技术结合，有望完全使用低精度算术单元计算目标精度快速傅里叶变换。

7 结论

本文提出了一种利用低精度快速傅里叶变换以目标精度计算快速傅里叶变换的方法。该方法基于布鲁斯坦快速傅里叶变换，该变换通过循环卷积计算离散傅里叶变换，并将 Ozaki 方案应用于此循环卷积。若使用浮点快速傅里叶变换，要保证循环卷积的精确计算，就需要在分量拆分卷积中进行大量快速傅里叶变换调用；因此，我们采用了数论变换（一种有限域上的快速傅里叶变换），并结合中国剩余定理。我们还引入了一个分割数上界 K，以控制数论变换调用次数，并提出了一种数论变换域累加策略，进一步减少所需的逆数论变换次数。

我们使用 32 位数论变换实现了双精度布鲁斯坦快速傅里叶变换，并评估了其精度和计算成本。与现有的双精度快速傅里叶变换实现（包括 FFTW 和基于 TS 的快速傅里叶变换）相比，所提方法有效降低了快速傅里叶变换的相对误差。尤其是，相对误差并未随 FFT 长度增加，这表明 Ozaki 方案能够抑制舍入误差的累积。根据输入数据的分布，对于从 2^{10} 到 2^{18}的 FFT 长度，所提双精度快速傅里叶变换最多需要调用 268 次 32 位数论变换和逆数论变换。将分割数上界设为 $K = 3$后，在保持精度的同时，32 位数论变换和逆数论变换的调用次数最多降至 96 次。此外，在计算逆数论变换前允许进行最多 $L = 3$ 次数论变换域累加，可进一步将调用次数降至 64 次，但这可能会因输入分布而降低精度。

在 Intel Xeon Platinum 8468 上，测得的执行时间约为 FFTW 双精度快速傅里叶变换的 107 倍（$n = 2^{15}$）至 1315 倍（$n = 2^{10}$）。32 位数论变换/数论变换$^{-1}$计算约占总执行时间的 80%。如果将 32 位数论变换/数论变换$^{-1}$优化到 FFTW 单精度快速傅里叶变换的水平，其在总执行时间中的占比将降至 11%（$n = 2^{10}$）至 62%（$n = 2^{15}$）。此外，由于数论变换部分与非数论变换部分的时间比相对于 FFT 长度仅按 $O(\log n)$ 增长，因此也有必要优化非数论变换部分。

要使所提方法优于基线双精度快速傅里叶变换，单次数论变换调用必须显著快于基线快速傅里叶变换；但在当前硬件上，这仍然很难实现。不过，对于任意长度快速傅里叶变换，获得加速的条件较为宽松，因此更有可能实现具有竞争力的执行时间。尽管目前没有可与基础线性代数子程序或 FFTW 相媲美的标准化数论变换库，但数论变换已广泛应用于基于格的密码学和全同态加密领域，高效的数论变换实现也正在积极开发中。尤其是，已有多种基于 GPU 的实现方案。由于所提方法以模块化方式使用数论变换，在算术精度之间性能差距显著的平台上，若能以高吞吐精度实现高效的数论变换，就可以仅通过调整分割数，使该方法适用于各种精度。

参考文献

Leo Bluestein。1970年。一种计算离散傅里叶变换的线性滤波方法。IEEE 音频与电声学汇刊。18卷，4期（1970年12月），451–455页。doi:10.1109/TAU.1970.1162132 William T. Cochran、James W. Cooley、David L. Favin、Howard D. Helms、Reginald A. Kaenel、William W. Lang、George C. Maling、David E. Nelson、Charles M. Rader 和 Peter D. Welch。1967年 55卷，10期 1967年 1664 1674 10 1109 1967 5957 。什么是快速傅里叶变换？《IEEE 会刊》（10月），–。doi: . /PROC. . James W. Cooley 和 John W. Tukey。1965年。一种用于计算复傅里叶级数的机器算法。《数学与计算》19卷，90期（1965年4月），297–301页。doi:10.2307/2003354 T. J. Dekker。1971年。一种扩展可用精度的浮点技术。《数值数学》18卷，3期（1971年6月），224–242页。doi:10.1007/BF01397083 Nicolas Fabiano、Jean-Michel Muller 和 Joris Picot。2019年 68卷，11期 2019年 1573 1583 10 1109 2019 2918451 。三字算术算法。《IEEE 计算机汇刊》（5月），–。doi: . /tc. . Tommy Färnqvist。2005年。数论与缓存局部性相结合：为 GNU 多重精度算术库高效实现小素数快速傅里叶变换。硕士论文，斯德哥尔摩大学（2005年1月）。Laurent Fousse、Guillaume Hanrot、Vincent Lefèvre、Patrick Pélissier 和 Paul Zimmermann。2007年。MPFR：一种支持正确舍入的多精度二进制浮点库。《ACM 数学软件汇刊》33卷，2期，文章13（2007年6月），15页。doi:10.1145/1236463.1236468 Michael Frewer。2021年。mpFFT。https://github.com/mfrewer/mpFFT Matteo Frigo 和 Steven G. Johnson。2005年。FFTW3 的设计与实现。《IEEE 会刊》93卷，2期（2005年2月），216–231页。doi:10.1109/JPROC.2004.840301 Harvey L. Garner。1959年。剩余数系统。《IRE 电子计算机汇刊》EC-2卷，2期（1959年6月），140–147页。doi:10.1109/ TEC.1959.5219515 GNU。2023年。GNU 多重精度库。https://gmplib.org Zibo Gong、Nathan Zhu、Matthew Ngaw、Joao Rivera、Larry Tang、Eric Tang、Het Mankad 和 Franz Franchetti。2022年。基于区间算术的快速傅里叶变换，用于大整数乘法。载于：第26届高性能极限计算会议论文集（HPEC 2022）。Yozo Hida、Xiaoye S. Li 和 David H. Bailey。2001年。四倍双精度浮点运算算法。载于：第15届 IEEE 计算机算术研讨会论文集。IEEE，Piscataway，NJ，155页。doi:10.1109/ARITH.2001.930115 IEEE。2008年。IEEE 浮点算术标准。IEEE 754-2008 标准（2008年），1–70页。doi:10.1109/IEEESTD.2008.4610935 Zhuoran Ji、Jianyu Zhao、Peimin Gao、Xiangkai Yin 和 Lei Ju。2025 年。利用多 GPU 系统加速数论变换，以实现高效的零知识证明。载于：第30届 ACM 编程语言与操作系统架构支持国际会议论文集（ASPLOS 2025），第1卷。ACM，纽约，NY，1–14页。doi:10.1145/3669940.3707241 Shota Kawakami 和 Daisuke Takahashi。2024年。在 GPU 上实现并评估八倍精度快速傅里叶变换。载于：第22届 IEEE 并行与分布式处理及应用国际研讨会论文集（ISPA 2024）。IEEE，Piscataway，NJ，287–294页。doi:10.1109/ISPA63168.2024.00044 Peter L. Montgomery。1985年。无需试除法的模乘法。《数学与计算》44卷，170期（1985年5月），519–521页。doi:10.2307/2007970 Daichi Mukunoki、Katsuhisa Ozaki、Takeshi Ogita 和 Toshiyuki Imamura。2020年。使用张量核心的 DGEMM 及其准确且可复现的版本。载于：2020年高性能计算国际会议论文集（ISC 高性能计算 2020），第12151卷。Springer，Cham，230–248页。doi:10.1007/978-3-030-50743-5_12 Daichi Mukunoki、Katsuhisa Ozaki、Takeshi Ogita 和 Toshiyuki Imamura。2021年。通过 Ozaki 方案加速的 binary128 格式准确矩阵乘法。载于：第50届并行处理国际会议论文集（ICPP 2021）。ACM，纽约，NY，1–11页。doi:10.1145/3472456.3472493 NVIDIA。2023年。NVIDIA H100 张量核心 GPU 数据手册。https://resources.nvidia.com/en-us-hopper-architecture/nvidia- h100-tensor-c NVIDIA。2024年。NVIDIA Blackwell 架构技术概览。https://resources.nvidia.com/en-us-blackwell-architecture/blackwell-architecture-technical-brief NVIDIA。2025a。NVIDIA Rubin 平台揭秘：六款新芯片，一台人工智能超级计算机。https://developer.nvidia.com/blog/inside-the-nvidia-rubin-platform-six-new-chips-one-ai-supercomputer NVIDIA。2025b。NVIDIA RTX Blackwell GPU 架构。https://images.nvidia.com/aem-dam/Solutions/geforce/blackwell/nvidia-rtx-blackwell-gpu-architecture.pdf Hiroyuki Ootomo、Katsuhisa Ozaki 和 Rio Yokota。2024年。在整数矩阵乘法单元上执行 DGEMM。《国际高性能计算应用期刊》38卷，4期（2024年3月），297–313页。doi:10.1177/10943420241239588 Katsuhisa Ozaki、Takeshi Ogita、Shin’ichi Oishi 和 Siegfried M. Rump。2012年。利用快速矩阵乘法例程对矩阵乘法进行无误差变换及其应用。《数值算法》59卷，1期（2012年6月），95–118页。doi:10.1007/s11075-011-9478-1

Katsuhisa Ozaki、Yuki Uchino 和 Toshiyuki Imamura。2025年。Ozaki 方案 II：一种采用整数模技术对浮点矩阵乘法进行面向 GEMM 的仿真方法。*arXiv* 预印本 *arXiv:2504.08009*(2025年4月)。Alisah Ozcan、Arsalan Javeed 和 Erkay Savas。2025年。通过基数 2 蝶形变换和四步算法在 GPU 上实现高性能数论变换。*IEEE Access* 13（2025年5月），87862–87883。doi:10.1109/ACCESS.2025.3570024 Colin Percival。2003年。高度合成数之和与差的快速模乘法。数学计算72，241（2003年1月），387–395。doi:10.1090/S0025-5718-02-01419-9 John M. Pollard。1971年。有限域上的快速傅里叶变换。数学计算25，114（1971年5月），365–374。Anumeena Sorna、Xiaohe Cheng、Eduardo D'Azevedo、Kwai Won 和 Stanimire Tomov。2018年。利用张量核心硬件上的混合精度优化快速傅里叶变换。载于第 *25* 届 *IEEE* 高性能计算国际会议研讨会论文集（*HiPCW 2018*）。IEEE，Piscataway，NJ，3–7。doi:10.1109/HiPCW.2018.8634417 Yukimasa Sugizaki 和 Daisuke Takahashi。2026年。使用张量核心改进 NVIDIA GPU 上数论变换的实现。载于亚洲超级计算与亚太地区高性能计算国际会议论文集（*SCA/HPCAsia 2026*）。ACM，纽约，NY，142–152。doi:10.1145/3773656.3773673 Dmitrii Tolmachev。2023年。VkFFT——一款高性能、跨平台的开源 GPU 快速傅里叶变换库。*IEEE Access* 11（2023年2月），12039–12058。doi:10.1109/ACCESS.2023.3242240 Taiga Utsugiri 和 Tomonori Kouya。2022年。基于三双精度（TD）和三单精度（TS）算术加速矩阵乘法。载于计算科学与高性能计算国际研讨会论文集（*CSHPC 2022*）。Springer，Cham，406–423。doi:10.1007/978-3-031-10562-3_29 Joris Van Der Hoeven 和 Grégoire Lecerf。2015年。中等精度下更快的快速傅里叶变换。载于第 *22* 届 *IEEE* 计算机算术研讨会论文集（*ARITH 2015*）。IEEE，Piscataway，NJ，75–82。doi:10.1109/ARITH.2015.10 Xiangkai Yin、Shuoyu Wang、Zhaorui Zhang、Zimeng Zhou、Lei Ju 和 Zhuoran Ji。2025年。TensorNTT：面向张量核心单元上数论变换的架构感知优化。载于*2025*年 *IEEE* 大数据国际会议论文集（*BigData 2025*）。IEEE，Piscataway，NJ，4835–4844。doi:10.1109/BigData66926.2025.11402438

收稿日期：xxxx年xx月xx日；修订日期：xxxx年xx月xx日；录用日期：xxxx年xx月xx日