PaReNTT：用于同态加密的低延迟并行剩余数系统及基于NTT的长多项式模乘法

Weihang Tan*，*IEEE* 学生会员；Sin-Wei Chiu*，*IEEE* 学生会员；Antian Wang，*IEEE* 学生会员；Yingjie Lao，*IEEE* 高级会员；以及 Keshab K. Parhi，*IEEE* 会士

arXiv:2303.02237v2 [cs.AR] 6 Jul 2023

摘要—**高速长多项式乘法对于同态加密（HE）和基于格的密码系统中的应用至关重要。本文研究了利用数论变换（NTT）和逆数论变换（iNTT）实现长多项式模乘法的低延迟硬件架构。提出了并行 NTT 和 iNTT 架构，以减少处理多项式所需的时钟周期数。采用中国剩余定理（CRT）将模数分解为多个较小的模数。我们所提出设计，即 PaReNTT，作出了三项创新贡献。首先，提出了级联并行 NTT 和 iNTT 架构，从而无需在将 NTT 的乘积输入 iNTT 之前对其进行置换所需的任何缓冲。该目标通过为 NTT 和 iNTT 使用不同的折叠集合来实现。其次，提出了一种扩展可行特殊模数集合的新方法，其中模数可表示为少数几个带符号的二次幂项。第三，提出了用于预处理的创新架构，以利用 CRT 计算余式多项式，以及用于后处理的创新架构，以合并余式多项式。这些架构显著降低了预处理和后处理步骤的面积消耗。所提出的长模多项式乘法非常适合需要低延迟和高采样率的应用，例如云端应用，因为这些前馈架构可以进行任意级别的流水线化。本文还研究了流水线化与延迟之间的权衡。与先前的设计相比，所提出架构将延迟降低了 49.2 倍，并分别将查找表和数字信号处理器的面积时间积（ATP），即 ATP（查找表）和 ATP（数字信号处理器），降低了 89.2% 和 92.5%。具体而言，我们表明，对于 n=4096以及 180 位系数，所提出的 2-并行架构在采用 Xilinx Virtex Ultrascale+FPGA、使用 6 个模数且每个模数长度为 30 位的情况下，以 240 MHz 运行时功耗为 6.3 瓦。**

索引词—**多项式模乘、并行数论变换/逆数论变换、剩余数系统、模数选择、格密码学、同态加密**

I. 引言

隐私保护协议和信息安全对于云计算至关重要。为此，云平台通常使用某些传统的对称密钥密码系统或非对称密钥密码系统对数据进行加密，以保护用户隐私。然而，由于数据必须在计算前解密，这些方法无法防止在云端计算过程中发生信息泄露，因此数据必须在计算前解密。为进一步增强隐私保护，同态加密（HE）已成为一种前景广阔的工具，可确保信息在不可信云端的机密性。同态加密也应用于隐私保护联邦学习 [1]和神经网络推理 [2]。

同态乘法和同态加法是同态加密方案的两种基本运算。现有的大多数同态加密方案都基于环上带误差学习问题（R-LWE）构造 [3] ，通过向密文中加入一定噪声来确保后量子安全。然而，同态乘法导致的噪声呈二次增长，因此需要使用非常大的密文模数，进而造成算术运算效率低下。一种可能的解决方案是对模数进行分解并并行处理。这种方法已用于剩余数系统（RNS）表示。文献中已有若干基于RNS的软件 [4], [5]和硬件实现 [6]–[8]。然而，RNS依赖中国剩余定理（CRT），因此需要额外的预处理和后处理操作。这些步骤中的硬件构件需要经过优化；否则，RNS系统的复杂度将抵消其并行度带来的优势。同时，模多项式乘法是基于环上带误差学习问题的密码系统，尤其是同态加密方案的一项关键算术运算。与基于常规乘法的模多项式乘法相比，基于数论变换（NTT）的模多项式乘法可大幅降低复杂度。

针对不同的应用，可以采用不同的模长多项式乘法器架构。例如，低面积的时分复用架构非常适合边缘设备。然而，云端需要速度极高的架构，每个时钟周期都需要处理多项式的多个系数。这本质上要求采用并行架构，其并行度取决于每个时钟周期处理的系数数量。尽管针对顺序架构和时分复用架构的设计与实现已有大量研究，但针对基于数论变换的并行架构的研究却少得多。如果不考虑调度方面的问题，计算两个多项式的数论变换乘积的逆数论变换（iNTT）可能会导致延迟较长，并需要额外的缓冲区，因为在计算

Weihang Tan、Sin-Wei Chiu 和 Keshab K. Parhi 任职于美国明尼苏达州明尼阿波利斯市明尼苏达大学电气与计算机工程系，邮编 55455。电子邮箱：{wtan、chiu0091、parhi}@umn.edu。Antian Wang 和 Yingjie Lao 任职于美国南卡罗来纳州克莱姆森市克莱姆森大学 Holcombe电气与计算机工程系，邮编 29634。电子邮箱：{antianw、ylao}@clemson.edu。*（贡献相同）

逆数论变换之前，必须对乘积进行重排。

尽管基于数论变换的并行架构能够实现低延迟和高速度，但由于系数的字长可能很大，其算术运算需要占用较大的硅面积。为减小面积，采用剩余类算术将系数转换为多个较小的系数，从而可以使用更短的字长实现。本文提出了一种并行剩余类算术和基于数论变换的模长多项式乘法方案，称为 PaReNTT。通过对数论变换和逆数论变换操作采用不同的调度方式（折叠），无需额外的缓冲区。因此，完整操作的延迟得以降低。采用并行 NTT架构减少了处理长多项式模乘法所需的时钟周期数。所提出设计的并行数论变换和逆数论变换架构完全采用前馈结构，并实现了百分之百的硬件利用率。这些架构可以在任意级别进行流水线化。据我们所知，所提出架构是首个面向通用、前馈且并行的基于数论变换的实现方案，能够消除中间数据重排或缓冲区需求。

本文的贡献主要有三点，概述如下。

- 我们提出的级联数论变换-逆数论变换架构无需执行中间数据重排操作。数论变换和逆数论变换采用不同的折叠方式，使得两个数论变换的乘积能够立即在逆数论变换中处理。这显著降低了延迟，并完全消除了对任何中间缓冲区的需求。

- 我们针对中国剩余定理考虑了一种特殊格式的素数，以降低实现成本。具体而言，所有素数不仅与数论变换兼容，而且适用于中国剩余定理，并且汉明重量较低（即仅包含少量带符号的二次幂项）。传统的模数选择方法若要满足这些约束，可选模数的数量会受到限制。本文提出了一种新方法，以扩展满足这些约束的模数集合。这使得能够构建适用于长字长系数的同态加密架构。

- 本文提出了用于剩余类算术预处理和后处理的新型优化架构；这些架构能够降低面积和功耗。最后，将用于剩余类算术的低成本预处理和后处理块集成到基于并行数论变换的模多项式乘法器中，从而实现高速、低延迟、低面积设计。

本文其余部分安排如下：第二节回顾同态加密、剩余数系统和基于数论变换的多项式模乘的数学背景，以及先前工作中相应的硬件架构。第三节提出一种用于基于数论变换的多项式乘法的并行架构，无需中间存储。随后，第四节介绍我们优化后的基于剩余数系统和中国剩余定理的多项式乘法器。第五节给出并分析所提出设计的性能。最后，第六节对本文进行总结。

II. 背景

A. 符号说明

对于多项式环 $R_{n,\ q} = \mathbb{Z}_q[x]/(x^n+1)$，其系数必须对 q 取模（即位于范围 $[0, q-1]$ 内），且多项式的次数小于 n （n 是 2 的幂整数）。为确保所有中间结果均属于该多项式环，需要进行模约简操作，表示为“mod $(x^n+1,\ q)$”或 $[\cdot]_q$。环 $R_{n,\ q}$ 的多项式记为 $a(x) = \sum_{j=0}^{n-1} a_j x^j$，其中多项式 $a(x)$ 的第 j 个系数表示为 a_j。

两个多项式模 (x^n+1, q) 的加法和乘法（即模多项式加法和乘法）分别记为 $a(x) + b(x)$ 和 $a(x) \cdot b(x)$。我们还用 $\odot$ 表示两个多项式在 (x^n+1, q) 上的逐点乘法。参数 $m = \log_2 n$和 $s \in [0, m-1]$ 分别表示数论变换（逆数论变换）中的总级数和当前级数。

B. 同态加密

同态加密允许直接对密文进行计算（例如乘法、加法），而无需解密，因此用户可以将数据上传到任何云服务器（即使是不可信的服务器），同时保护隐私。同态加密方案大致可分为全同态加密（FHE）和部分同态加密（SHE）。全同态加密方案允许进行任意数量的同态求值，但计算复杂度较高 [9]。部分同态加密是比全同态加密效率更高的一种替代方案，只允许在不解密的情况下执行有限次数的运算 [3], [10], [11]。

同态加密方案的高级步骤可概括为四个阶段：密钥生成、加密、求值和解密。具体而言，密钥生成步骤根据安全参数 λ，生成三个密钥：私钥、公钥和重线性化密钥。随后，加密算法使用公钥将消息加密为密文 ct。在求值阶段，安全的求值函数会对所有输入密文执行同态计算，并使用重线性化密钥输出新的密文 ct' 。最后，在解密阶段，可使用私钥和 ct' 获得结果。

密钥生成、加密和解密步骤通常由客户端执行。与此同时，求值步骤则分配给云服务器，以进行同态计算。不同的同态求值函数具有不同的计算成本。同态加法相对简单，因为它通过模多项式加法实现。然而，同态乘法需要耗费大量计算资源的模多项式乘法。因此，针对模多项式乘法器的硬件或软件加速，尤其是在多项式次数较大且系数字长较长的同态加密参数下，要求很高。

C. 剩余数系统

在各种应用中实现同态加密时，同态乘法的深度是需要研究的主要因素之一，并且会随着系数字长的增加而成比例增长

。例如，在先前工作中 [7]，要在 80 位安全级别下执行深度为四的同态乘法，需要 180 位的密文模数和长度为 4096 的多项式。然而，涉及长字长系数的计算并不简单，若不采用高层次变换，效率也不高。由于大多数广泛使用的部分同态加密方案（如 BGV [3]、BFV [10] 和 CKKS [11]）的模数不受限于素数，因此可以通过使用中国剩余定理，将每个模数选为若干互不相同的素数之积，其中每个素数都是字长较小的数论变换兼容素数。

中国剩余定理算法将 q 分解为 q_1，q_2，……，q_t（即，$q = \prod_{i=1}^{t} q_i$、q_i互素），并建立环同构 $R_q \equiv R_{q_1} \times R_{q_2}$，……，$\times R_{q_t}$。分解后，每个 R_{q_i} 中的环运算都可单独执行，因此可以并行处理。从实现角度来看，参数 t越大，每个 q_i越小，且在 R_{q_i}上进行的算术运算越简单。

D. 基于 *NTT* 的多项式乘法

除了系数字长较长之外，为了使同态加密方案维持较高的安全级别，多项式次数 n 也可能达到数千，这成为软件和硬件实现 [12], [13]的瓶颈。因此，采用了一种时间复杂度为 $\mathcal{O}(n\log\ n)$ 的高效基于 NTT 的多项式乘法方法。

为了计算 $p(x) = a(x) \cdot b(x) \bmod (x^n + 1,\ q)$，首先将多项式 $a(x)$ 和 $b(x)$映射为其 NTT 域多项式 $A(x)$ 和 $B(x)$。例如，多项式 $a(x)$ 的数论变换计算表示为：$A_k = \sum_{j=0}^{n-1} a_j \omega_n^{kj} \bmod\ q$，其中 $k \in [0, n-1]$。ω 是模 q 的本原 n 次单位根（即旋转因子），满足 $\omega^n \equiv 1 \bmod q$。随后，对 $A(x)$ 和 $B(x)$ 进行高效的逐点乘法，得到 $P(x) = A(x) \odot B(x)$。最终结果通过逆数论变换计算得到：$n^{-1} \sum_{j=0}^{n-1} P_j \omega_n^{-kj}$ q $k \in [0, n-1]$ ( ) mod ，其中 。

与朴素多项式乘法及模多项式约简的 $\mathcal{O}(n^2)$ 复杂度方法相比，该方法显著降低了时间复杂度。然而，这种原始方法需要进行长度为 n 的零填充，使数论变换/iNTT计算中的多项式长度加倍。已有研究表明，使用负循环卷积（*NWC*）

零填充可以被完全消除 [14]。然而，这要求对输入和输出进行加权。这些额外的加权操作可通过重构算法予以消除。这被称为低复杂度负循环卷积 [15], [16]。 [17] 中回顾了原始和低复杂度数论变换方法。图1展示了基于低复杂度负向环绕卷积的数论变换和逆数论变换的流程图，其中 $n = 8$。具体而言，ψ_{2n} 是模 q 意义下的原$2n$次单位根，且 q 必须是满足（$q - 1$）可被 $2n$ 整除的 NTT 兼容素数。该低复杂度算法减少了模整数乘法的数量。同时，乘以 2^{-1} 的模乘

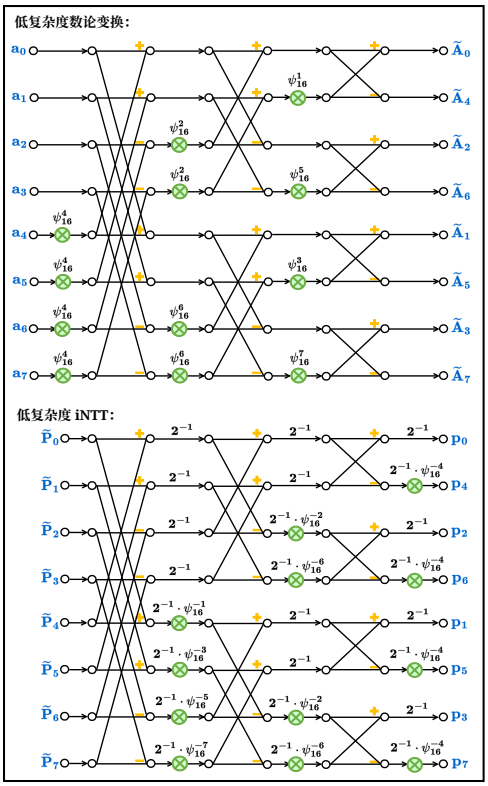


图 1. 当 $n = 8$ 时，用于多项式乘法的低复杂度数论变换和 iNTT 数据流图。

可在硬件中采用低成本模加法器和多路复用器（MUX）高效实现（见补充信息）。

E. 先前的硬件实现

文献中已提出多种基于中国剩余定理优化的硬件架构 [6], [7], [18]–[19]，这些架构采用反馈式架构，并通过循环以时分复用的方式执行多项操作 [6], [7], [18]。

[6], [7] 中的工作为 BFV 方案引入了一种近似中国剩余定理方法，该方法涉及提升和缩放操作，以在较小的密文模数 q与较大的密文模数 Q之间切换。随后， [18]提出了一种利用剩余数系统和数论变换算法的多级并行加速器。然而，这些工作主要侧重于优化数论变换块，而非中国剩余定理系统的预处理和后处理功能块。

具体而言，先前工作针对 NTT 和 iNTT 采用统一架构，通常基于存储器或折叠架构框架构建 [15], [18], [20]–[23]。这种设计策略可以减少所需处理单元（PE）的数量。与先前工作不同，我们所提出的架构采用前馈并行架构。我们的架构没有反馈环路/数据通路。因此，中间结果可以直接由下一级处理单元执行和处理。值得注意的是，并行流水线化及基于存储器的 NTT/iNTT 架构设计，与先前快速傅里叶变换/逆快速傅里叶变换设计类似 [24], [25]。

III. 无打乱操作的基于并行 NTT 的多项式乘法器

本节提出一种新颖的实时、前馈、高吞吐量并行 NTT 多项式乘法架构设计，如图2所示，该设计无需中间数据重排。对于传统实现，通常需要额外的重排电路，在计算 iNTT 之前重新排列输出数据。然而，这种重排电路需要大量时钟周期和寄存器。在这一新颖架构中，双并行乘积被送入双并行逆数论变换架构，因此无需中间缓冲区。这样，乘积输出即可由 iNTT 立即处理。这之所以可行，是因为我们可以为 NTT 和 iNTT 选择不同的折叠集合。此外，所提出设计适用于任意 n值，支持参数化，并可实现任意级别的流水线化，从而实现高速运行。

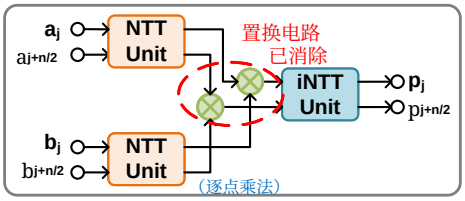


图2. 采用双并行 NTT 和 iNTT 的模多项式乘法器架构。

具体而言，图2中的 NTT/iNTT 单元基于双并行架构；这些单元通过适当的折叠集合和折叠变换 [26], [27] 构建而成。图3和图4分别展示了对 $a(x)$ 进行16点正向数论变换和对 $P(x)$ 进行 iNTT 的数据流图，其中每个圆圈表示一次蝶形运算。

应用折叠变换后，同一颜色（即同一级）的运算由同一个 PE 处理，然后以时分复用方式执行。在同一个 PE 中执行蝶形运算的顺序称为折叠顺序。此外，图3和图4中还以蓝色突出显示了每个蝶形运算对应的时钟周期。在这个16点示例中，正向 NTT 的折叠集（即每个 PE 中执行的有序运算集合）表示为：

$$\begin{aligned}\mathcal{A} &= \{\mathcal{A}_0,\mathcal{A}_1,\mathcal{A}_2,\mathcal{A}_3,\mathcal{A}_4,\mathcal{A}_5,\mathcal{A}_6,\mathcal{A}_7\}\\ \mathcal{B} &= \{\mathcal{B}_4,\ \mathcal{B}_5,\mathcal{B}_6,\ \mathcal{B}_7,\mathcal{B}_0,\ \mathcal{B}_1,\mathcal{B}_2,\ \mathcal{B}_3\}\\ \mathcal{C} &= \{\mathcal{C}_2,\ \ \mathcal{C}_3,\mathcal{C}_4,\ \ \mathcal{C}_5,\mathcal{C}_6,\ \ \mathcal{C}_7,\mathcal{C}_0,\ \ \mathcal{C}_1\}\\ \mathcal{D} &= \{\mathcal{D}_1,\mathcal{D}_2,\mathcal{D}_3,\mathcal{D}_4,\mathcal{D}_5,\mathcal{D}_6,\mathcal{D}_7,\mathcal{D}_0\}.\end{aligned} \tag{1}$$

为了避免在数论变换与逆数论变换之间使用中间缓冲区或进行数据格式转换，数论变换单元中最后一个处理单元的输出样本必须在同一时钟周期内馈入逆数论变换单元中的第一个处理单元。这通过以下方式实现

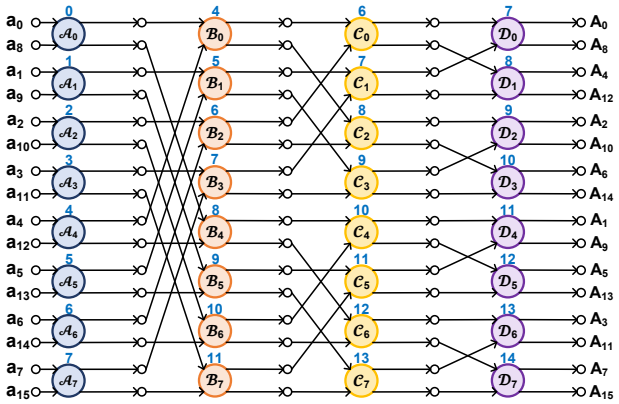


图3. 16点正向数论变换的数据流图。

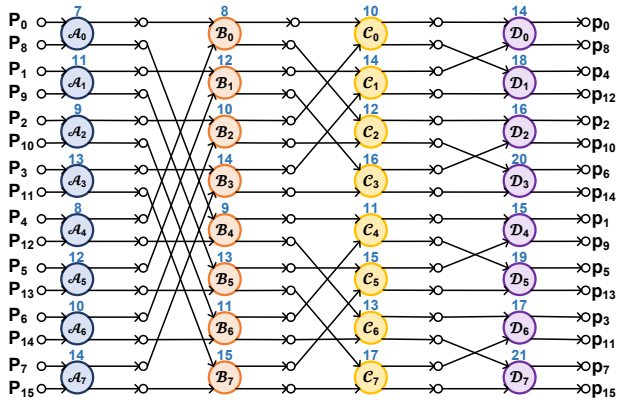


图4. 16点逆数论变换的数据流图。

逆数论变换的折叠集：

$$\begin{aligned}\mathcal{A} &= \{\mathcal{A}_4,\mathcal{A}_2,\mathcal{A}_6,\mathcal{A}_1,\mathcal{A}_5,\mathcal{A}_3,\mathcal{A}_7,\ \mathcal{A}_0\}\\ \mathcal{B} &= \{\mathcal{B}_0,\ \mathcal{B}_4,\mathcal{B}_2,\ \mathcal{B}_6,\mathcal{B}_1,\ \mathcal{B}_5,\mathcal{B}_3,\ \ \mathcal{B}_7\}\\ \mathcal{C} &= \{\mathcal{C}_3,\ \ \mathcal{C}_7,\mathcal{C}_0,\ \ \mathcal{C}_4,\mathcal{C}_2,\ \ \mathcal{C}_6,\mathcal{C}_1,\ \ \ \mathcal{C}_5\}\\ \mathcal{D} &= \{\mathcal{D}_2,\mathcal{D}_6,\mathcal{D}_1,\mathcal{D}_5,\mathcal{D}_3,\mathcal{D}_7,\mathcal{D}_0,\mathcal{D}_4\}.\end{aligned} \tag{2}$$

NTT 和 iNTT 的设计受到基于折叠集的并行快速傅里叶变换架构设计的启发 [24]。我们在早期工作中提出了基于折叠集的并行 NTT 架构 [28]。图5中的 NTT架构是根据公式（1）所示的折叠集推导得出的。具体而言，该架构包含四个处理单元和三个延迟-切换-延迟单元（DSD），处理单元和 DSD 的结构分别如图8和图6所示。此外，DSD 模块采用两个多路复用器和两个寄存器组，因此能够存储数据通路中的特定数据，然后切换数据或将其传递给处理单元。请注意，每个寄存器组中的寄存器数量会因阶段而异。在第 s 级，NTT架构的 DSD 模块中每个寄存器组都有 2^{m-s-2} 个寄存器。此外，iNTT 架构如图7所示，其组件如图9和图6所示。图9展示了一种用于 iNTT 的硬件友好型处理单元设计，该设计只需执行一次右移运算、一次与常数 $\frac{q+1}{2}$ 相加的模加，以及使用一个多路复用器进行一次模二除法 [29]。NTT 和 iNTT 架构的主要区别之一在于其内部寄存器的数量

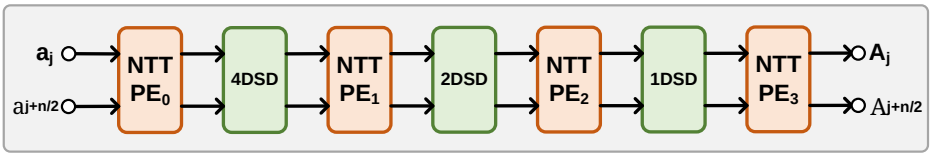


图5. 16点正向数论变换单元的架构。

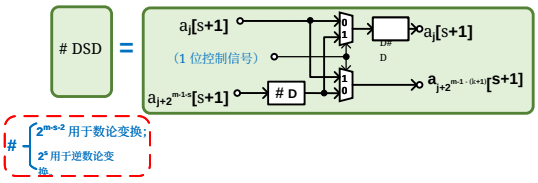


图6. 延迟-切换-延迟（DSD）单元的架构。

这是因为每个 DSD 模块中的寄存器数量由公式（2）中的折叠集决定。具体而言，iNTT 架构在第 s 级的每个寄存器组需要 2^s 个寄存器。尽管 NTT 和 iNTT 的运算非常相似，我们仍采用两种独立架构，而不是采用统一且可重配置的架构。理由如下：同态乘法中大量使用模乘，因此为 NTT 和 iNTT 采用两种不同的架构，可以使输入多项式连续流动，从而显著加速同态加密乘法。

图5和图7中的16点架构也可以通过配置 m个处理单元和 $m-1$个延迟-切换-延迟（DSD）块，轻松推广到任意二次幂长度 n 。此外，一般情况下的数论变换和逆数论变换折叠集合定义如下。我们将 s级中的处理单元记为 PE_s，该处理单元内部执行的蝶形运算对应的数论变换折叠集合如表 I所示。表中的条目描述了数据流图中该级节点的节点索引。折叠顺序描述了节点执行时所在的时钟分区。例如，折叠顺序 s 意味着该节点在时钟周期（$n/2$）$l+s$执行，其中 l为整数。由于数论变换的一个级中有 $n/2$ 个运算（节点），因此折叠集合的基数为 $n/2$ 。因此，调度周期为 $n/2$。

iNTT 的折叠集也可以像表 II 中那样进行推广，其中符号“$\langle\cdot\rangle$”表示针对 $(m-1)$ 位整数的折叠集的位反转表示（例如， $m=4$ 时的 $\langle 1\rangle = \langle 001_b\rangle = 100_b = 4$ ）。具体而言，如果 NTT 中的节点 i 的折叠顺序为 i，则 iNTT 中对应节点的折叠顺序为模 $(n/2)$ 的 $\langle i\rangle - 1$ 。虽然众所周知，位反转调度能够在数据流图层面消除延迟和缓冲区需求，但同一特性也适用于并行 NTT-iNTT 级联这一发现并不直观，而且是新的。

注意，如果使用公式（1）中的相同折叠集来设计逆数论变换，则需要将乘积输入到一个大小为 4（通常为 $n/4$）的 DSD 电路中。这将额外引入 4（通常为 $n/4$ ）个时钟周期的延迟。对数论变换和逆数论变换使用不同的折叠集合，可消除任何额外的 DSD 电路及其相关延迟。

IV. 模数选择与 PARENTT 架构

A. 所提出的 *PaReNTT* 架构概述

图 10 展示了所提出设计的 PaReNTT 架构概览，该架构可分为三个组成步骤。第一步称为剩余多项式计算（预处理操作），它将两个输入多项式拆分为若干个系数较小的多项式。不同于采用单个模多项式乘法器，在剩余域中并行执行多个模多项式乘法。随后，后处理操作使用中国剩余定理将乘积多项式通过逆映射转换为一个多项式。其结果与直接对两个输入多项式执行模多项式乘法相同。

B. 特殊的数论变换兼容且适用于中国剩余定理的素数选择

不同于先前工作随机选择互素数，本文研究并利用特殊互素数的性质，以降低计算成本和硅面积。这种优化的核心思想是牺牲互素数选择的灵活性，换取架构在时序/面积方面的性能。

在所提出的架构中，每个 q_i 不仅需要是数论变换兼容素数，而且还需要具有较短的字长，其定义如下：

$$q_i = 2^v - \beta_i, \quad \beta_i = 2^{v_{1i}} \pm 2^{v_{2i}} \pm \ldots \pm 2^{v_{n_q i}} - 1, \qquad (3)$$

其中， v是 q_i, $v_{1i} > v_{2i} \ldots > v_{n_q i}$的字长。 q_i 中的带符号二次幂项数为 $(n_q + 2)$。

通过穷举搜索 t 互素因子，可以找到特殊的数论变换兼容且适用于中国剩余定理的素数，并将其组合成 vt 位密文模数 q。需要满足以下两个约束：(1) $(q_i - 1)$ 是 $2n$ 的倍数；(2) $\lceil \frac{\mu-1}{n_\beta} \rceil > v_{1i} > v_{2i}$。第二个约束将在 C 小节中推导，参见公式 (6)。此处的 μ 是 Barrett 约减单元输入的字长（见下文 C 小节中的说明）。在典型的 Barrett 约减实现中， $\mu = 2v$。在所提出的方法中，对于给定的 v 和 t，增大 μ 和 n_q，以扩大可行模数的数量。

适用于中国剩余定理的模数能够为预处理和后处理步骤带来优化后的硬件架构，从而改善整体时序和面积表现。我们的穷举搜索方法生成了与 Solinas 素数相似的 q_i ，其中包含少量带符号的二次幂项 [30], [31]。

在硬件实现中，整数乘法器比整数加法器消耗更多面积且延迟更长。此外，面积和延迟与

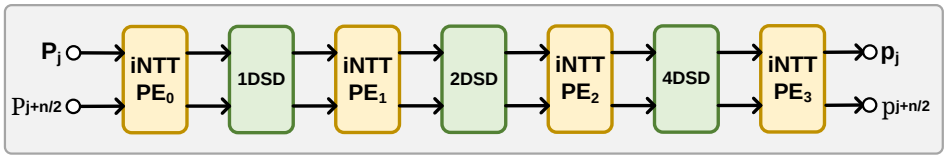


图7. 16点逆数论变换单元的架构。

表 I 数论变换的通用折叠顺序

折叠顺序	0	1		l		$\frac{n}{2}-1$
PE_0	0	1	...	l	...	$\frac{n}{2}-1$
PE_1	2^{m-2}	$2^{m-2}+1$	...	$2^{m-2}+l$ 模 $\frac{n}{2}$	...	$2^{m-2}-1$
				...		
PE_s	2^{m-s-1}	$2^{m-s-1}+1$	...	$2^{m-s-1}+l$ 模 $\frac{n}{2}$	...	$2^{m-s-1}-1$ 模 $\frac{n}{2}$
				...		
PE_{m-1}	1	2	...	$l+1$	...	0

表 II INTT 的广义折叠顺序

折叠顺序	0	1		l		$\frac{n}{2}-1$
PE_0	$\langle 1\rangle$	$\langle 2\rangle$	...	$\langle l+1\rangle$	...	$\langle 0\rangle$
PE_1	$\langle 0\rangle$	$\langle 1\rangle$	...	$\langle l\rangle$	...	$\langle 2^{m-1}-1\rangle$
				...		
PE_s	$\langle 2-2^s$ 模 $\frac{n}{2}\rangle$	$\langle 2-2^s+1$ 模 $\frac{n}{2}\rangle$	...	$\langle 2-2^s+l$ 模 $\frac{n}{2}\rangle$	...	$\langle 2-2^s-1$ 模 $\frac{n}{2}\rangle$
				...		
PE_{m-1}	$\langle 2\rangle$	$\langle 3\rangle$	...	$\langle l+2$ 模 $\frac{n}{2}\rangle$	...	$\langle 1\rangle$

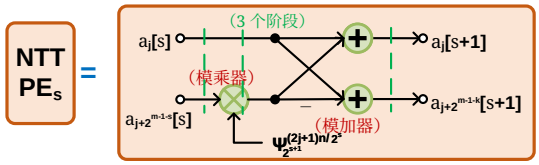


图8. 将数论变换中的加权运算合并后的基于DIT的蝶形结构架构。绿色线条标出了流水线切割集。

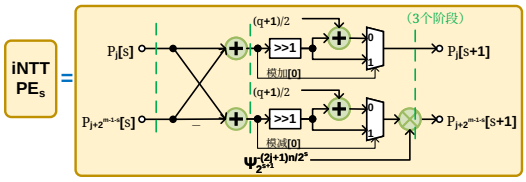


图9. 将逆数论变换中的加权运算合并后的基于DIF的蝶形运算架构。绿色线条标出了流水线切割集。

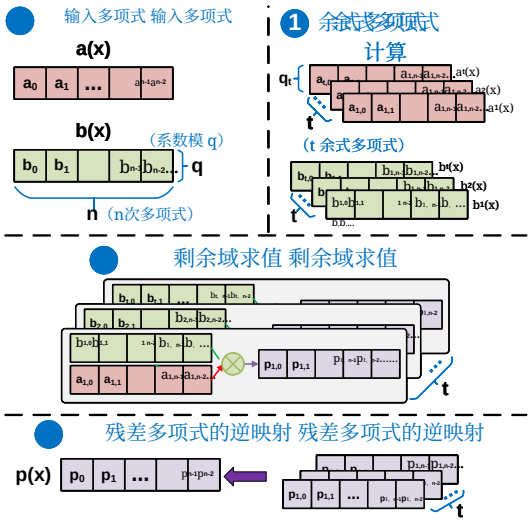


图 10. 所提出设计的 PaReNTT 架构概览。

字长成正比。因此，优化模乘器、预处理级和后处理级架构的一个可行方向是减少整数乘法器的数量，尤其是长整数乘法器的数量。在所提出设计的方法中，乘以 q_i时可消除所有整数乘法器，从而显著降低计算成本。

C. 剩余多项式计算单元

预处理阶段通过应用中国剩余定理算法，将输入多项式映射为其余式多项式，如图10的步骤1所示。对于多项式 $a(x)$，其余式多项式为

$$a_i(x) = [a(x)]_{q_i} = \sum_{j=0}^{n-1} (a_{i,j} \bmod q_i) x^j, \quad i \in [1, t]. \tag{4}$$

预处理阶段中的一项关键操作是执行模约简。为避免在计算模运算时进行除法操作，一种方法是采用 Barrett 约减 [32]。其描述如下：

$$a \mod q = a - (\frac{am}{2^\mu}) \cdot q = a - ((a\epsilon) \gg \mu) \cdot q$$

其中，$\epsilon = \lfloor \frac{2^\mu}{q} \rfloor$ 可以预先计算，μ 是 a的字长。

先前工作采用分治范式来计算余式多项式，以提升并行度并降低复杂度，例如 [6]。图 11(a) 展示了这种方法的一个示例，即针对 $t = 4$ 的全并行实现。尽管这种方法具有优势，

该方法仍需在每个分段内进行模乘，因此存在进一步优化的空间。具体而言，我们利用模数汉明重量较低这一特性，以移位加法单元（SAU）取代模乘器。

算法1介绍了我们所提出的、用于实现公式（4）的一种新颖且适合硬件的优化方法。先前工作和所提出算法的架构分别如图 11(a)和图 11(b)所示。与先前工作类似，[6], 算法1的第1行首先将一个大整数 a_j 拆分为多个分段，其中每个分段包含 v位（v 是 q_i的字长）。为简便起见，我们定义基数 $B = 2^v$。因此，$a_{i,\ j}$ 中的每个分段可表示为 $z_k \cdot B^k$、 $k \in [0, t-1]$。下一步是对每个分段进行模约简，这是我们硬件优化的重点。

算法1 高效的剩余系数计算

输入： $a_j \in [0, q-1]$ 和 q_i
输出： $a_{i,\ j} = a_j$ 模 q_i， $a_{i,\ j} \in R_{q_i}$
1: $a_j = z_0 + z_1 \cdot B + z_2 \cdot B^2 +$**, ...,** $+ z_{t-1} \cdot B^{t-1}$ //$B = 2^v$
2: **对于** $k = 1$ **至** $t-1$**，执行**
3: $r_k = z_k \times \beta_i^k$ // $\beta_i = B$ 模 q_i
4: $a_{i,\ j} = z_0 + r_1 + ... + r_{t-1}$ 模 q_i

算法1中的第3行不再需要使用 β_i^k 进行 $v \times v$ 位整数乘法来获得每个 r_k。相反，所提出设计利用移位加法操作消除开销高昂的模乘。

此外，与图 11(a) 中需要将每个 r_k模约简为 q_i的基准设计不同，我们的设计在理想情况下将（$t-1$）个约减单元减少至仅一个，以满足算法1第 4 行的要求。其原理如下。先前工作中的乘积r_k 是一个 $2v$ 位整数，因为 β_i^k 和 z_k 均为v位。

由于 q_i 仅包含少量带符号的二次幂项，因此算法1第 3 行中的长整数乘法可由一个 SAU 替代。例如，对于特殊素数 $q_i = 2^v - 2^{v_{1i}} - 2^{v_{2i}} + 1$，算法1第 3 行中的 β_i 可表示为

$$\beta_i = [2^v]_{q_i} \equiv 2^{v_{1i}} + 2^{v_{2i}} - 1. \tag{5}$$

使用 SAU 乘以 β_i 的过程如图12所示。此处， $z_1 \times \beta_i$ 的字长为（$v + v_{1i} + 1$）。经过 n_β 个 SAU 单元后，字长增至（$v + n_\beta$（$v_{1i} + 1$））。算法1第 4 行的 $a_{i,\ j}$ μ的字长大于或等于（$v + n_\beta$（$v_{1i} + 1$）$+1$），其中 μ 是 Barrett约减单元输入的字长。因此得到以下约束：

$$\lceil \frac{\mu - 1}{n_\beta} \rceil > v_{1i} > v_{2i}. \tag{6}$$

一般情况下的参数 $n_\beta = t - 1$。

算法1的框图如图 13 所示，适用于 $t = 4$。可以看出，图 11(a) 中 $z_k \times \beta_i^k$ 的模乘法可以替换为移位和加法运算，从而降低硬件成本。由于相对于字长而言，乘法器的成本通常按平方增长，采用这种移位和加法运算比使用乘法器得到结果更节省面积 r_k。

D. 按需增加素数的数量

互素因子 t 数量的增加最终可能加深 SAU 的深度，导致中间结果的字长变长，从而降低计算效率。为克服这一瓶颈，我们采用了两种替代方案。

方法1. 第一种方案是在数据通路中策略性地放置一个额外的 Barrett 约减单元，以降低 SAU 的最大深度。在 SAU 单元之间插入额外的 Barrett 约减单元，可以将 SAU 的深度降至零，从而将中间结果的字长缩短至 v位。例如，如图 14 和图 13 中橙色部分所示，为 r_3 增加一个 Barrett 约减单元，可将 SAU 的深度降至 1，确保所有 Barrett 约减单元的输入字长都较短。因此，由于运算过程中的中间结果 r_k 采用较短的字长表示，合并所有 r_k 和 z_0 以计算$a_{i,\ j}$ 只需使用加法器和一个 Barrett 约减单元。

尽管存在这一额外开销，但如图 11(a) 所示，由于 Barrett 约减单元数量减少且无需进行整数乘法，其硬件资源需求仍低于先前的设计。当模数数量较少时（例如少于 5 个），这种方法较为适用。

方法2。 当模数数量 t 较大时，由于 SAU 单元的数量会随着 t 呈平方级增长，上述方法效率不高。针对这种情况，我们提出了一种新方法，如算法 2 所述。首先，将 t 分解为 $t = d \cdot t'$，其中 t' 个模数通过与方法1类似的 SAU 单元组合，构成一个块。随后使用 d 个这样的块，每个块处理 t' 个模数。每个块中 SAU 单元的最大深度为 ($t' - 1$)。

请注意，这种方法中使用的互素因子需要对公式（5）进行调整 $n_\beta = t' - 1$，以满足条件 $\lceil \frac{\mu-1}{n_\beta} \rceil > v_{1i} > v_{2i}$。

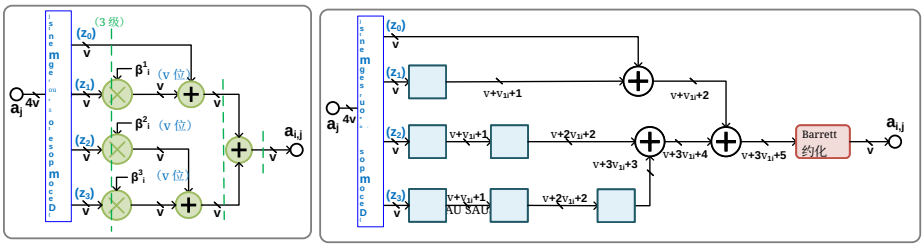
图 15 展示了一个包含六个互素因子（t =6）的示例。该电路主要由两块（$d = 2$）SAU 单元组成（以蓝色标示），每块具有三个输入（$t' = 3$），并配有额外的 Barrett约减单元和一个乘法器。在此示例中，每个分段 z_k 通过乘以 β_i^k 进行模约简，其中 $k \in [0, 5]$。第一块使用低成本的 SAU 单元计算从 β_i^0 到 β_i^2 的乘法：

$$sum_{i,0} = z_0 \cdot \beta_i^0 + z_1 \cdot \beta_i^1 + z_2 \cdot \beta_i^2. \tag{7}$$

由于 $\beta_i^0 = 1$，无需对 z_0进行模运算。

分段 z_3 至 z_5 用于执行模约简，随后通过应用乘法的分配律对其进行优化：

$$\begin{aligned} sum_{i,1} &= [z_3 \cdot \beta_i^3 + z_4 \cdot \beta_i^4 + z_5 \cdot \beta_i^5]_{q_i}, \\ &= [z_3 \cdot \beta_i^0 + z_4 \cdot \beta_i^1 + z_5 \cdot \beta_i^2]_{q_i} \cdot [\beta_i^3]_{q_i}, \end{aligned} \tag{8}$$



（a）优化前　　（b）优化后

图11. 当 $t = 4$ 时，剩余系数计算单元的顶层架构。

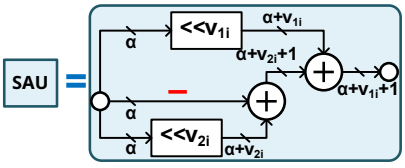


图12. 输入字长为 α 的剩余系数计算单元的 SAU 单元。

其中，$[z_3 \cdot \beta_i^0 + z_4 \cdot \beta_i^1 + z_5 \cdot \beta_i^2]_{q_i}$ 可通过对第一块中的相同组件实现，该组件由移位加法单元（SAU）构成，后接一个 Barrett约减单元；乘法 $[\beta_i^3]_{q_i}$（一个 v 位预计算常数）则通过一个（$v \times v$）位乘法器实例化。这一针对公式（8）的新型优化确保 $sum_{i,1}$ 的中间结果固定为 $2v$ 位。最终，$sum_{i,0}$ 和 $sum_{i,1}$ 通过 Barrett约减单元累加并约简为 v 位结果。

算法 2 通过以下方式高效计算剩余系数因式分解

输入： $a_j \in [0, q-1]$，q_i，以及 $t = t' \cdot d$

输出： $a_{i,\ j} = a_j$ 模 q_i, $a_{i,\ j} \in R_{q_i}$

1: $a_j = z_0 + z_1 \cdot B + z_2 \cdot B^2 +, ..., + z_{t-1} \cdot B^{t-1}$ //$B = 2^v$
2: **对于** $\rho = 0$ **到** $d-1$ **执行**
3: 　**对于** $k = 1$ **到** $t'-1$，**执行**
4: 　　$r_k = z_k \times \beta_i^k$ 　// $\beta_i = B$ 模 q_i
5: 　**如果** $\rho == 0$，**则**
6: 　　$sum_{i,0} = z_0 + r_1 + ... + r_{t'-1}$
7: 　**else**
8: 　　$sum_{i,\rho} = [z_0 + r_1 + ... + r_{t'-1}]_{q_i} \cdot [\beta_i^{t'\rho}]_{q_i}$
9: $a_{i,j} = sum_{i,0} + sum_{i,1} + \cdots + sum_{i,d-1}$ 模 q_i

在计算复杂度分析方面，这种所提出的方法降低了硬件资源消耗。与图 11(a) 中的设计相比，该方法将整数乘法器和模约简单元的数量分别从 t 和 t 减少到 $(d-1)$ 和 d。然而，它会额外使用 $^{t}{}_{(t'-1)\ 2}$ 个 SAU 单元。例如，当 $t = 2 \cdot 3$时，所提出的方法可将六个整数乘法器和六个模约简单元分别减少至一个整数乘法器和两个模约简单元。需要注意的是，在穷举搜索过程中，采用此方法并不要求在生成互素 (q_i) 时对约束参数 $t = 6$ 施加限制。相反，该方法利用约束参数 $t' = 3$ 获得六个满足条件的互素因子，因为 SAU 单元的最大深度为二（即 $n_\beta = 2$ 而非 $n_\beta = 5$），这显著拓宽了互素因子搜索空间的灵活性。

E. 剩余域求值

使用中国剩余定理表示后，$f(a_i(x)\ b_i(x))$ 在 $R_{n,\ q_i}$ 上的函数可以独立计算，如图10中的步骤 2 所示。因此，整体的 t 运算可以并行执行。在我们的案例中，该函数利用基于 NTT 的多项式乘法，在 $R_{n,\ q_i}$ 上为 $i \in [1, t]$ 计算剩余乘积 $p_i(x)$。计算 $p_i(x) = a_i(x) \cdot b_i(x)$ 模 $(x^n + 1, q_i)$ 的架构基于我们在图2中提出的新型基于 NTT 的多项式乘法器。因此，我们所提出的架构通过提高中国剩余定理表示的并行度，实现了高吞吐量和低延迟。

F. 多项式剩余系数的逆映射

在图10的步骤 3 中，需要将通过剩余域求值获得的结果转换回环 $R_{n,\ q}$ 上，这与 $f(a(x)\ b(x))$ 在 $R_{n,\ q}$ 上的结果相同（即未使用中国剩余定理表示计算出的结果）。

此后处理阶段基于逆中国剩余定理算法：

$$\begin{aligned} p(x) &= \sum_{i=1}^{t} p_i(x) \cdot e_i \bmod q \\ &= \sum_{i=1}^{t} \sum_{j=0}^{n-1} p_{i,j} \cdot e_i \cdot x^j \bmod q, \end{aligned} \tag{9}$$

其中每个 $e_i = q_i^* \cdot \tilde{q}_i$ 均为常数，$q_i^* = (\frac{q}{q_i}) \in \mathbb{Z}$，且 $\tilde{q}_i = [(\frac{q}{q_i})^{-1}]_{q_i} \in \mathbb{Z}_{q_i}$。

然而，直接乘以常数 e_i 涉及较长的整数乘法，并且需要对 q 进行代价高昂的模约简，这会导致实现效率低下且关键路径较长。同时，特殊互素数的性质可以降低后处理阶段对 q_i 进行模运算的成本。因此，我们利用 [33] 中的技术，将公式（9）进一步表示为：

$$\begin{aligned} p(x) &= \sum_{i=1}^{t} \left[p_i(x) \cdot \tilde{q}_i\right]_{q_i} \cdot q_i^* \bmod q \\ &= \sum_{i=1}^{t} \sum_{j=0}^{n-1} \left[p_{i,j} \cdot \tilde{q}_i\right]_{q_i} \cdot q_i^* \cdot x^j \bmod q. \end{aligned} \tag{10}$$

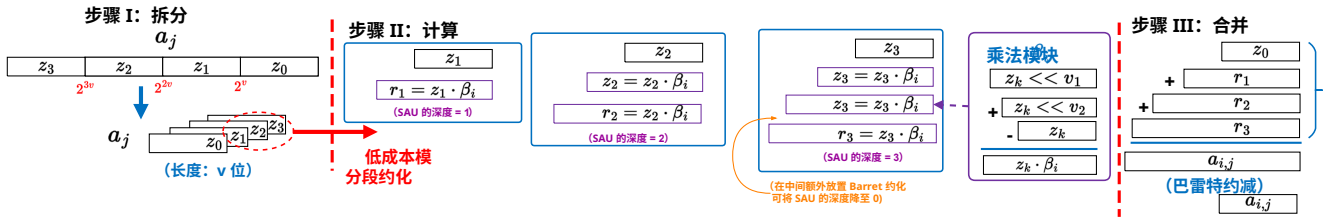


图 13. 当 $t = 4$ 时，剩余系数计算单元的流程图。

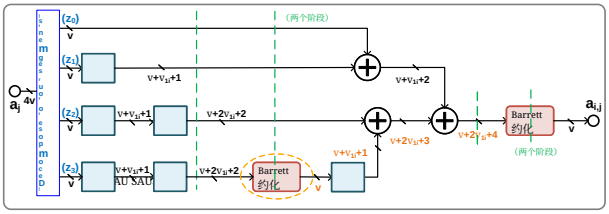


图 14. 当 $t = 4$ 时，带有额外 Barrett 约减单元的剩余系数计算单元。图中圈出了用于降低 SAU 深度的额外 Barrett 约减单元，而放置该额外 Barrett 约减单元后的字长以橙色突出显示。

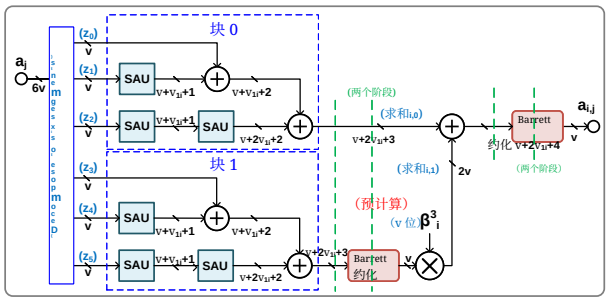


图 15. 通过因式分解单元进行剩余系数计算，适用于 $t = 6$。

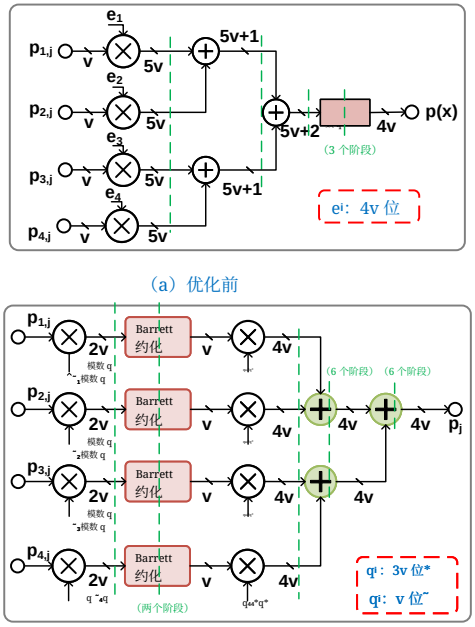


图 16. 当 $t = 4$时的逆映射架构。该电路展示了逆中国剩余定理的后处理步骤。每个整数加法器/乘法器之后均设置流水线切割集。

该方法的核心思想是将一个较长的 $v \times vt$ 位乘法器拆分为一个 $v \times v$ 位乘法器和一个 $v \times (t-1)v$ 位乘法器。因此，相对于 q 的模约简被替换为针对不同 q_i 的四次独立模约简。这种优化带来的资源节省可解释如下：

由于对 q_i 进行模约简的成本低于对 q进行模约简的成本，因此可以高效地执行 $0 \le \left[p_{i,j} \cdot \tilde{q}_i\right]_{q_i} < q_i$ 中的计算。由于 q_i^* 是一个 $(t-1)v$ 位的预计算常数，后处理级无需进行除法。此外，$\left[p_i(x) \cdot \tilde{q}_i\right]_{q_i} \cdot q_i^*$中的系数范围处于 $[0, q-1]$ 内，因此计算乘积时无需进行模乘。

多项式剩余系数的逆映射的优化后架构如图 16(b)所示（以 $t = 4$ 为例）。在该架构中，每个用于计算 e_i 的长字长（$4v \times v$ 位）乘法器被拆分为一个带常数 $\tilde{q}_i$ 的 $v \times v$ 位乘法器和一个带常数 q_i^* 的 $v \times 2v$ 位乘法器。无需像图 16(a) 中那样对大模数 q 块执行开销较大的模约简，只需进行三个模加法器操作和四次对 q_i 的模约减，即可得到最终结果 $p(x)$。

具体而言，基于这一特殊的互素性质，对 q_i 进行模约简也很高效。

总体而言，所提出的新型架构能够显著降低面积和功耗。

五、实验结果

本节分别评估互素因子选择、未采用打乱操作的基于并行 NTT 的多项式乘法器（如第三节所述），以及中国剩余定理算法的预处理/后处理单元（详见第四节）的性能。随后，本文对所提出的 PaReNTT 多项式乘法器进行全面的性能讨论与比较分析。

在我们的评估中，所提出设计使用 SystemVerilog 实现，并映射到 Xilinx Virtex Ultrascale+FPGA 上。我们还专门考虑了一个固定的 180 位 q，其包含四个或六个互素因子，多项式次数为 $n = 4096$，以研究设计在不同中国剩余定理并行度下的表现。因此，180 位模数 q 由 45 位或 30 位的互素数构成，且这些互素数符合特殊的数论变换兼容和适用于中国剩余定理的格式。

请注意，我们的设计可以轻松扩展到更长的字长模数，方法是增加互素因子的数量，或增大每个互素因子的字长。此外，对于长度为 4096 的 $t = 4$基于数论变换的多项式乘法器，在 $m = \log_2(4096) = 12$的情况下，采用 48 个处理单元和 44 个 DSD 单元。当 t =6 时，则采用 72 个处理单元和 66 个 DSD 单元。也可以集成次数更高的多项式，此时只需增加处理单元和 DSD 单元的数量。

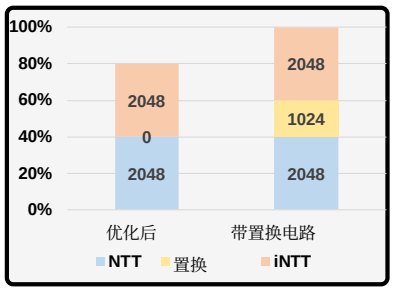


图 17. 当 $n = 4096$ 时，采用和不采用打乱操作的双并行 NTT 多项式乘法的延迟比较。

A. 互素因子的扩展

表 III 展示了不同设置下适用于数论变换且适用于中国剩余定理的特殊素数总数。选择了 μ 的两个值: $\mu = (2v + 15)$ 和 $(2v + 30)$。带符号的二次幂项数量为 4 或 5。当μ、带符号的二次幂项数量和 n 分别设为 ($2v + 30$) 位、5 项和长度 4096 时，可行的互素因子数量分别为 169（对应 $v = 30$）和 480（对应 $v = 45$）。因此，互素因子的数量足以满足较长系数字长的需求。在我们的硬件实现中，针对 30 位（$v = 30$）和 45 位（$v = 45$）互素数（对应 $\mu =$ （$2v$+15）），分别考虑了 75 位和 105 位 μ。此外，这些互素数均由四个带符号的二次幂项构成。

表 III 不同设置下特殊数论变换兼容素数和中国剩余定理友好素数的数量，其中 $t = 4$ 且 $t = 6$ （$v = 45$和 30）

t	v	μ	# 二次幂	n	$\lceil$对数$_2\,\epsilon\rceil$	# 素数
4	45	$(2v + 15)$	4	4096	61	12
4	45	$(2v + 30)$	4	4096	76	33
4	45	$(2v + 15)$	5	4096	61	126
4	45	$(2v + 30)$	5	4096	76	480
6	30	$(2v + 15)$	4	4096	46	8
6	30	$(2v + 30)$	4	4096	61	26
6	30	$(2v + 15)$	5	4096	46	23
6	30	$(2v + 30)$	5	4096	61	169

μ: Barrett约减单元的输入字长；# 二次幂：每个互素因子中带符号的二次幂项数量。

B. 基于并行 NTT 的多项式乘法器的评估指标与性能

为分析各实现的时序性能，我们定义了两个时序性能指标：块处理周期(BPP) 和延迟。BPP 定义为处理 n 个系数输入所需的时间，或生成 n 个系数输出所需的时间。对于长度为 n 的基于 NTT 的双并行多项式乘法器，BPP 的表达式为

$$T_{BPP} = n/2, \tag{11}$$

其中，吞吐量为每个时钟周期处理两个样本。此外，一次模多项式乘法的延迟为

$$T_{Lat} = (n - 2) + T_{pipe}, \tag{12}$$

其中，T_{pipe} 表示为缩短关键路径而添加到数据通路中的额外流水线级数。此外，执行 L 次模多项式乘法所消耗的总时钟周期数为

$$T_{total} = T_{Lat} + T_{BPP} \cdot L. \tag{13}$$

对于 $n = 4096$，BPP 为 2048 个时钟周期，延迟为 4096 个时钟周期（不包括流水线化所需的额外时钟周期）。与先前工作中采用重排电路的基于 NTT 的多项式乘法器相比，延迟显著降低。图 17 展示了我们优化后的方法与传统方法（不考虑流水线化）的比较。具体而言，采用重排电路的传统方法需要额外 1024 个（一般情况下为 $n/4$ 个）时钟周期进行重排序，因此双并行设计的延迟会增加约 20.0%，并且 $n = 4096$。

当 $t = 4$ AND $t = 6$ ($\lceil \log_2 q_i \rceil = 45$ AND 30) 时剩余系数计算单元的面积消耗和频率

设计	t	频率	LUTs	DSPs	FFs	N_{pip}
先前工作	4	76	6350	0	0	0
所提出设计	4	**62**	**4034**	**0**	**0**	**0**
先前工作	4	200	5836	0	1288	5
所提出设计	4	**271**	**3937**	**0**	**1164**	**6**
先前工作	6	105	2032	0	0	0
所提出设计	6	**55**	**1148**	**0**	**0**	**0**
先前工作	6	300	2660	0	1244	6
所提出设计	6	**309**	**1537**	**0**	**682**	**6**

表 V AREA 消耗量与频率：逆映射架构 $t = 4$ 与 $\lceil \log_2 q_i \rceil = 45$

设计	频率	LUTs	DSPs	FFs	N_{pip}
传统	45	17729	63	0	0
所提出设计	**50**	**15894**	**60**	**0**	**0**
传统	111	15066	63	2544	6
所提出设计	**244**	**12302**	**60**	**6686**	**16**

C. 剩余系数计算单元与逆映射架构的比较

图 14 和图 15 展示并比较了残差系数的设计，图16 展示了逆映射

表 VI $n = 4096$模多项式乘法器的面积消耗和频率

设计	$\lceil \log_2 q \rceil$	t	频率[MHz]	查找表[a]	数字信号处理器[a]	FFs[a]	功耗 [W]
所提出设计	180	4	244	322K (27.2%)	1.6K (22.8%)	92K (3.9%)	6.6
	180	6	240	341K (28.9%)	1.1K（16.5%）	103K（4.3%）	6.3
Roy [7]	180	6	225	64K	0.3K	25K	（未报告）

[a]：FPGA板上已使用资源数量（利用率）。

表 VII $n = 4096$的模多项式乘法器时序性能

$\lceil$对数$_2\, q\rceil$	t	CRT	BPP[b]		延迟[c]		ABP[d] (查找表)	ABP[d] (数字信号处理器)	ATP[e] (查找表)	ATP[e] (数字信号处理器)
			# 周期数	周期 [μ秒]	# 周期数	周期 [μ秒]				
180	4	Yes	2048	8.5	4246	17.4	2.7M	13.1K	5.6M	27.8K
180	6	Yes	2048	8.4	4254	17.7	2.9M	9.6K	6.0M	19.5K
180	6	Yes	N/A	N/A	196003	871.1	N/A	N/A	55.8M	261.3K

[b]：块处理周期（BPP）是处理 n 个系数输入，或在第一个样本输出后生成 n 个样本输出所需的周期（μs）。[c]：延迟是第一个样本输入到第一个样本输出所需的周期（μs）。[d]：面积-块处理周期积（ABP）由查找表/数字信号处理器的数量乘以块处理周期（μs）计算得到。[e]：面积-延迟积（ATP）由查找表/数字信号处理器的数量乘以延迟（μs）计算得到。

计算架构。这些基础组件在采用和未采用流水线化两种情况下的实验结果和比较结果，均列于表 IV 和表 V 中。图 14、图 15 和图 16 中以绿色标出了构建块中的流水线切割集。

在评估所提出设计的剩余系数计算单元的结果时，我们考虑了图 14（针对 $v = 45$、$t = 4$）和图 15（针对 $v = 30$、 $t = 6$）中所示两种不同方法的实验结果。此外，我们参考了图 11(a) 中描述的先前设计，并根据我们的参数设置，采用 Barrett 约减单元以全并行方式实现了该设计。该设计被用作表 IV 中所示剩余系数计算单元比较的基准设计。同时考虑了流水线化和非流水线化设计。对于非流水线化设计（$N_{pip} = 0$），我们观察到，所提出设计用于预处理的面积需求小于先前设计。对先前设计（图 11(a)）与图 14 所示所提出设计的比较表明，在流水线化之前，前者具有更短的关键路径。这些设计均为前馈结构，可以在适当的层级进行流水线化。为进行公平比较，我们对两种设计均进行了适当的流水线化，以实现高速运行。结果表明，我们的设计在查找表消耗方面显著减少了 32.5%。这一节省主要源于用两个 Barrett 约减单元及额外的低成本移位加法单元（SAU）替代四个整数乘法器和四个 Barrett约减单元。同时，图 11(a) 与我们在图 15 中所提出设计的比较显示，经过流水线化后，查找表节省量提高至 67.7%。

此外，采用 $v = 45$ 和 $t = 4$ 的参数设置，对比了逆映射架构中的传统设计图 16(a)与所提出设计图 16(b)。面积消耗结果表明，所提出设计分别将查找表和数字信号处理器的使用量降低了 18.3%和 4.8%。这些节省主要源于：使用四个采用特殊素数 q_i的 Barrett 单元，替代了与 q 相关的高开销 Barrett 约减单元。具体而言，在使用 q进行 Barrett 约减时，我们的方法不再与一个 180 位整数q 相乘，而是采用四次短字长移位加法运算，计算与 45 位专用 q_i的乘法。尽管与使用 e_i的乘法器相比，$\tilde{q_i}$ 和 q_i^* 的乘法器总字长保持不变，但所提出硬件架构在算法层面对长字长乘法进行分解，从而无需进一步转换即可实现直接的流水线化优化。

D. PaReNTT 多项式乘法器的求值

本小节深入探讨所提出的 PaReNTT 多项式乘法器（基于双并行剩余类算术的 NTT 架构）在 n =4096 和 $\lceil \log_2 q \rceil = 180$ 下的实现与比较。

参数设置 $t = 4$、 $v = 45$ 和 $t = 6$、 $v = 30$ 下的性能与实验结果见表 VI 和表 VII。这两种实现采用相同的架构设计，用于在剩余域中进行求值（即第三节所述、适用于不同 q_i 的基于并行 NTT 的多项式乘法器）以及对多项式的剩余系数进行逆映射。然而， $t = 4$ 和 $t = 6$ 所采用的剩余多项式计算单元分别基于图 14 和图 15。表 IV 详细列出了这两个块的结果。

在时序性能方面，两种设计经过流水线化后，均可在 240MHz 的高时钟频率下运行。还可以观察到，尽管字长 v 有所变化，但由于多项式次数固定，BPP 和以时钟周期数衡量的延迟仍基本保持不变。此外，还对 PaReNTT 架构在 $t = 4$和 $t = 6$ 下的面积性能进行了考察。如表 VI 所示，与针对 $t = 4$实现的设计相比，针对 $t = 6$ 实现的设计所用查找表增加了 5.6%，同时数字信号处理器的使用量减少了 31.25%。

为全面比较我们所提出设计的时序和面积性能，我们评估了面积-BPP 乘积（ABP）。与 $t = 4$ 设计相比，$t = 6$ 设计分别使 ABP（查找表）和 ABP（数字信号处理器）降低了 6.90% 和 26.72%。

我们的 PaReNTT 架构中，功耗的主要来源是 DSD 单元中部署的移位寄存器，以及查找表和数字信号处理器中执行的逻辑运算。由于 $t = 6$ 实现使用的资源较少，因此功耗也有所降低。具体而言，与$t = 4$ 实现相比，功耗约降低了 4.5%。

尽管 $n = 4096$ 和 $\lceil \log_2 q \rceil$ =180 的参数设置表明，$t = 6$ 实现在 ABP（查找表）和 ABP（数字信号处理器）方面表现更优，但为 n、v和 t 选择不同参数，也可能影响互素因子选择的灵活性和 ABP 性能。这表明，图 14 或图 15 所示设计之间的选择以及参数的选取，应根据不同同态加密应用的需求进行仔细调整。

由于系统采用不同的数据通路，并针对不同技术使用不同的 FPGA 器件，因此难以与先前工作直接比较。尽管如此，我们现在将所提出设计与一项先前设计进行比较，该设计采用相同的参数设置 $n = 4096$、$\log_2(q) = 180$，基于 [7]，并使用相同的 FPGA 器件。先前设计的时序和面积性能列于表VII和表VI的最后一行。此外，为减小参数设置的差异，所提出的 PaReNTT 架构采用了 $v = 30$ 和 $t = 6$ 的参数设置，这与 [7]相同。

尽管之前设计的面积性能优于 PaReNTT 架构，但我们的设计具有更好的时序性能，因为降低延迟和提高吞吐量是本工作的主要目标。具体而言，之前的设计针对 BFV 方案进行了定制优化，需要执行提升和缩放操作。因此，与不包含这些操作的设计相比，同态乘法中模乘所需的时钟周期数大约增加了一倍。为进行公平比较，我们将其设计中基于中国剩余定理、NTT 和逆数论变换操作的时钟周期数及延迟消耗均减半。等效的时钟周期数为 196，003 =(87，582 × 2 + 102，043 + 15，662 +99，137)/2，延迟为 871.1 μs。请注意，近似时序结果取自 [7]中的表 II，计算方式为将两个 NTT（分别针对多项式 $a(x)$ 和 $b(x)$）、逐系数乘法、逆数论变换、提升相加后再除以二。由于其缩放步骤因方案要求而需要比余式多项式的一般逆映射更复杂的操作，因此该步骤的时钟周期数未计入其近似时序结果。

比较和评估结果表明，我们的设计将延迟降低了 49.2 倍。此外，我们还比较了这两种设计的面积-时序乘积（ATP）。结果表明，与 [7]中的设计相比，我们的设计分别将 ATP（查找表）和 ATP（数字信号处理器）降低了 89.2% 和 92.5%。

VI. 结论

本文提出了 PaReNTT，一种高效的基于中国剩余定理和数论变换的长多项式乘法器。该设计利用经过特殊选择的素数的特性，对中国剩余定理算法的预处理单元和后处理单元进行了优化。此外，基于位反转调度设计了一种新型逆数论变换单元，从而消除了高成本的重排电路，并显著降低了延迟。未来的工作将基于硬件-软件协同设计，使用所提出的高效长多项式乘法器对 BFV、BGV 和 CKKS 等不同的同态加密算法进行评估。

参考文献

[1] F. Wibawa、F. O. Catak、M. Kuzlu、S. Sarp 和 U. Cali，“基于同态加密和联邦学习的隐私保护卷积神经网络训练：COVID-19 检测用例”，载于*2022*年欧洲跨学科网络安全会议论文集，2022年，第85–90页。[2] H. Chen、W. Dai、M. Kim 和 Y. Song，“利用打包密文实现高效多密钥同态加密及其在不经意神经网络推理中的应用”，载于*2019*年*ACM SIGSAC*计算机与通信安全会议论文集，2019年，第395–412页。[3] V. Lyubashevsky、C. Peikert 和 O. Regev，“关于理想格及环上的带误差学习”，载于密码技术理论与应用国际年会。Springer，2010年，第1–23页。[4] S. Halevi 和 V. Shoup，“HElib中的算法”，载于密码学年会。Springer，2014年，第554–571页。[5] H. Chen、K. Laine 和 R. Player，“简易加密算术库——SEAL v2.1”，载于金融密码学与数据安全国际会议。Springer，2017年，第3–18页。[6] S. S. Roy、K. Jarvinen、J. Vliegen、F. Vercauteren 和 I. Verbauwhede，“HEPCloud：用于FV有限同态函数求值的基于FPGA的多核处理器”，*IEEE*计算机汇刊，2018年。[7] S. S. Roy、F. Turan、K. Jarvinen、F. Vercauteren 和 I. Verbauwhede，“用于加密数据同态计算的基于FPGA的高性能并行架构”，载于*2019*年*IEEE*高性能计算机体系结构国际研讨会（*HPCA*）。IEEE，2019年，第387–398页。[8] M. S. Riazi、K. Laine、B. Pelton 和 W. Dai，“HEAX：一种用于加密数据计算的架构”，载于第二十五届编程语言与操作系统体系结构支持国际会议论文集，2020年，第1295–1309页。[9] C. Gentry，一种全同态加密方案。斯坦福大学，2009年。[10] J. Fan 和 F. Vercauteren，“实用性较强的全同态加密”。*IACR*密码学电子印本档案，第2012卷，第144页，2012年。[11] J. H. Cheon、A. Kim、M. Kim 和 Y. Song，“用于近似数算术的同态加密”，载于密码学与信息安全理论及应用国际会议。Springer，2017年，第409–437页。[12] A. Aysu、C. Patterson 和 P. Schaumont，“基于格的密码学的低成本、高面积效率FPGA实现”，载于*2013*年*IEEE*面向硬件的安全与可信国际研讨会（*HOST*）。IEEE，2013年，第81–86页。[13] W. Dai 和 B. Sunar，“cuHE：同态加密加速器库”，载于巴尔干地区密码学与信息安全国际会议。Springer，2015年，第169–186页。[14] V. Lyubashevsky、D. Micciancio、C. Peikert 和 A. Rosen，“SWIFFT：一种简易的FFT哈希方案”，载于快速软件加密国际研讨会。Springer，2008年，第54–72页。[15] N. Zhang、B. Yang、C. Chen、S. Yin、S. Wei 和 L. Liu，“采用低复杂度数论变换/逆数论变换的NewHope-NIST在FPGA上的高效架构”，*IACR*密码硬件与嵌入式系统汇刊，第49–72页，2020年。[16] P. Longa 和 M. Naehrig，“加速数论变换以实现更快的基于理想格的密码学”，载于密码学与网络安全国际会议。Springer，2016年，第124–139页。[17] S.-W. Chiu 和 K. K. Parhi，“基于NTT的同态加密多项式模乘：教程”，*arXiv:230612519*，2023年。

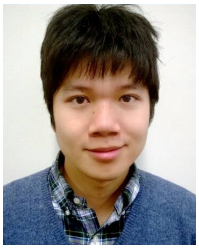
[18] G. Xin、Y. Zhao 和 J. Han，“一种用于机器学习中同态计算的多层并行硬件架构”，载于*2021*年*IEEE*国际电路与系统研讨会（*ISCAS*）。IEEE，2021年，第1–5页。[19] F. Turan、S. S. Roy 和 I. Verbauwhede，“HEAWS：一种用于Amazon AWS FPGA的同态加密加速器”，*IEEE*计算机汇刊，第69卷，第8期，第1185–1196页，2020年。[20] Y. Xing 和 S. Li，“在FPGA上实现紧凑型CCA安全密钥交换机制CRYSTALS-KYBER的硬件方案”，*IACR*密码硬件与嵌入式系统汇刊，第328–356页，2021年。[21] A. C. Mert、S. Kwon、Y. Shin、D. Yoo、Y. Lee、S. S. Roy 等，“Medha：用于加密数据计算的微码硬件加速器”，密码学电子印本档案，2022年。[22] A. C. Mert、E. ¨Ozt¨urk 和 E. Savas.，“运行时可配置的基于NTT的多项式乘法硬件的FPGA实现”，微处理器与微系统，第78卷，第103219页，2020年。[23] R. Paludo 和 L. Sousa，“面向Linux的RISC-V全同态加密加速器的NTT架构”，*IEEE*电路与系统汇刊*I*辑：常规论文，2022年。[24] M. Ayinala、M. Brown 和 K. K. Parhi，“通过折叠变换实现流水线并行FFT架构”，*IEEE*超大规模集成系统汇刊，第20卷，第6期，第1068–1081页，2011年。[25] M. Ayinala、Y. Lao 和 K. K. Parhi，“一种实值信号原位FFT架构”，*IEEE*电路与系统汇刊*II*辑：简报，第60卷，第10期，第652–656页，2013年。[26] K. K. Parhi，超大规模集成电路数字信号处理系统：设计与实现。John Wiley & Sons，1999年。[27] K. K. Parhi、C.-Y. Wang 和 A. P. Brown，“折叠流水线DSP架构中控制电路的综合”，*IEEE*固态电路杂志，第27卷，第1期，第29–43页，1992年。[28] W. Tan、A. Wang、Y. Lao、X. Zhang 和 K. K. Parhi，“面向基于格的密码学的流水线高吞吐量NTT架构”，载于*2021*年亚洲面向硬件的安全与信任研讨会（*AsianHOST*）。IEEE，2021年，第1–4页。[29] Y. Zhang、C. Wang、D. E. S. Kundi、A. Khalid、M. O’Neill 和 W. Liu，“一种高效并行的R-LWE密码处理器”，*IEEE*电路与系统汇刊*II*辑：简报，第67卷，890，2020年5，886no. pp. – 。[30] W. Tan、B. M. Case、A. Wang、S. Gao 和 Y. Lao，“面向基于格的密码系统的高速模乘器”，*IEEE*电路与系统汇刊*II*辑：简报，第68卷，第8期，第2927–2931页，2021年。[31] M. Hamburg，“Ed448-goldilocks，一种新的椭圆曲线。”*IACR*密码学电子印本档案，第2015卷，第625页，2015年。[32] P. Barrett，“在标准数字信号处理器上实现Rivest、Shamir和Adleman公钥加密算法”，载于密码技术理论与应用会议。Springer，1986年，第311–323页。[33] S. Halevi、Y. Polyakov 和 V. Shoup，“BFV同态加密方案的改进型RNS变体”，载于*RSA*会议密码学家专题。Springer，2019年，第83–105页。



Sin-Wei Chiu （IEEE 研究生会员）于 2020 年获得台湾国立中央大学电气工程学士学位。目前，他正在明尼苏达大学双城分校攻读电气工程博士学位。他目前的研究兴趣包括超大规模集成电路架构设计、后量子密码学和数字信号处理系统。



Antian Wang （IEEE研究生会员）于2017年获得上海海事大学通信工程学士学位。目前，他正在克莱姆森大学攻读博士学位。他的研究兴趣包括硬件安全、超大规模集成电路架构设计以及设计自动化。



Yingjie Lao （IEEE高级会员）现任克莱姆森大学电气与计算机工程系助理教授。他于2009年获得中国浙江大学学士学位，并于2015年获得明尼苏达大学双城分校电气与计算机工程系博士学位。他曾获得美国国家科学基金会职业奖、国际低功耗电子与设计研讨会（ISLPED）最佳论文奖，以及IEEE电路与系统学会超大规模超大规模集成系统最佳论文奖。



Weihang Tan （IEEE研究生会员）是明尼苏达大学双城分校电气与计算机工程系的博士后研究助理。他于2018年、2020年和2022年分别获得美国克莱姆森大学电气工程学士、硕士和博士学位。他的研究兴趣包括面向全同态加密、后量子密码学和数字信号处理系统的硬件安全与超大规模集成电路架构设计。他是亚洲硬件安全与可信研讨会（AsianHOST）的最佳博士生论坛报告奖。



Keshab K. Parhi （IEEE会士）是Erwin A. Kelen电气工程讲席教授，同时也是明尼苏达大学电气与计算机工程系的杰出麦克奈特大学教授。他于1988年在加利福尼亚大学伯克利分校获得电气工程与计算机科学博士学位。他已发表700多篇论文，拥有34项专利，并著有教材《超大规模集成电路数字信号处理系统》（威立出版社，1999年）。他目前的研究涉及面向机器学习的超大规模集成电路架构、硬件安性、数据驱动神经科学和DNA计算。Parhi博士曾荣获众多奖项，包括2003年IEEE Kiyo Tomiyasu技术领域奖、2017年Mac Van Valkenburg奖，以及IEEE电路与系统学会颁发的2012年Charles A. Desoer技术成就奖。他于2004年和2005年担任*IEEE*电路与系统汇刊第一部分：常规论文的主编。他是美国计算机协会、美国医学与生物工程研究所、美国科学促进会和美国国家发明家学会会士。

补充信息：PaReNTT：低延迟并行剩余数系统与基于NTT的长多项式模乘法，用于同态加密

本补充信息简要介绍通过 NTT/iNTT 在频域进行多项式模乘的算法。文中回顾了两种方法：负循环卷积（NWC）和低复杂度 NWC。 [17] 对这一主题进行了详细讲解。

本文采用一种高效的基于数论变换（NTT）的多项式乘法方法，其时间复杂度为$\mathcal{O}(n\log\ n)$。与朴素多项式乘法及多项式模约简的 $\mathcal{O}(n^2)$复杂度方法相比，该方法显著降低了时间复杂度。

[14] 中的先前工作提出了一种高效的基于 NTT 的多项式乘法算法，用于计算 $p(x) = a(x)\cdot b(x) \bmod (x^n+1, q)$，即负循环卷积，如算法1所示。需要注意的是，在负循环卷积中，NTT 之前和逆数论变换之后都需要进行加权运算，以避免代价高昂的零填充 [14]。

该算法的核心步骤是数论变换，它将多项式 $a(x)$ 和 $b(x)$ 转换为其数论变换域中的 $A(x)$ ~和 $B(x)$，如步骤 2 所示。多项式 $a(x)$ 的数论变换在数学上表示为

$$\widetilde{A}_k = \sum_{j=0}^{n-1} a_j \psi_{2n}^j \omega_n^{kj} \mod q, \quad k \in [0, n-1]. \tag{14}$$

~多项式 $b(x)$ 同样被变换为 $B(x)$。具体而言，ω 是模 q 意义下的 n次本原单位根（即旋转因子），满足 $\omega^n \equiv 1 \bmod q$。 ψ_{2n} 是模 q意义下的$2n$次本原单位根，因此 $\omega = \psi_{2n}^2 \bmod q$。使用数论变换算法后，对 $A(x)$和 $B(x)$进行高效的逐点乘法~ ~，随后执行逆数论变换（iNTT）。逆数论变换将乘积 P，转换回原始代数域中的多项式 $p(x)$，其定义为

$$p_k = n^{-1}\psi_{2n}^{-k} \sum_{j=0}^{n-1} \widetilde{P}_j \omega_n^{-kj} \mod q, \quad k \in [0, n-1], \tag{15}$$

其中， n^{-1} 是模 q意义下 n 的模乘法逆元。

在数论变换和逆数论变换过程中，加权运算要求将多项式乘以相应权重：数论变换使用 $\psi_{2n}^j \bmod q$ ，逆数论变换使用 $\psi_{2n}^{-j} \bmod q$ 。此外，还会使用数论变换兼容素数，即 q 必须满足$(q-1)$能被 $2n$整除。

由于数论变换/逆数论变换中的加权运算需要进行大量开销高昂的模乘， [15], [16] 中的近期研究提出了一种将加权运算合并到蝶形运算中的新方法。具体而言，公式（14）中的新~ ~数论变换可重新表示为 A_k 和 $A_{k+n/2}$，其推导如下：

算法1 负循环卷积 [14]

输入： $a(x)\ b(x)\ \in R_{n\ q}$
输出： $p(x)\ = a(x)\ \cdot b(x)$ 模 (x^n+1, q)

1：加权运算：
$\widetilde{a}(x)\ = \sum_{j=0}^{n-1} a_j \psi_{2n}^j x^j$ 模 q
$\widetilde{b}(x)\ = \sum_{j=0}^{n-1} b_j \psi_{2n}^j x^j$ 模 q

2：数论变换计算：
$A(x)$: $A_k = \sum_{j=0}^{n-1} \widetilde{a}_j \omega_n^{kj}$ 模 q, $k \in [0, n-1]$
$\widetilde{B}(x)$: $B_k = \sum_{j=0}^{n-1} \widetilde{b}_j \omega_n^{kj}$ 模 q, $k \in [0, n-1]$

3：逐点乘法：
$P(x)\ = \widetilde{A}(x)\ \odot \widetilde{B}(x)\ = \sum_{k=0}^{n-1} \widetilde{A}_k \widetilde{B}_k x^k$

4：iNTT计算：
$\tilde{p}(x)\ = n^{-1} \sum_{j=0}^{n-1} \tilde{P}_j \omega_n^{-kj}$ 模 q, $k \in [0, n-1]$

5：加权运算：
$p(x)\ = \sum_{j=0}^{n-1} \widetilde{p}_j \psi_{2n}^{-j} x^j$

使用时间抽取（DIT）方法：

$$\widetilde{A}_k = a_k^{(0)} + \psi_{2n}\omega_n^k a_k^{(1)} \mod q, \tag{16}$$

$$\widetilde{A}_{k+n/2} = a_k^{(0)} - \psi_{2n}\omega_n^k a_k^{(1)} \mod q, \tag{17}$$

其中 $k \in [0, \frac{n}{2}-1]$ 和

$$a_k^{(0)} = \sum_{j=0}^{n/2-1} a_{2j}\psi_n^j \omega_{n/2}^{kj} \mod q, \tag{18}$$

$$a_k^{(1)} = \sum_{j=0}^{n/2-1} a_{2j+1}\psi_n^j \omega_{n/2}^{kj} \mod q. \tag{19}$$

由于 $\omega = \psi_{2n}^2$ 模 q，整数 ψ_{2n}^j 和 $\omega_{n/2}^{kj}$ 可合并为一个整数 $\psi_{2n}\omega_n^k = \psi_{2n}^{(2k+1)}$。因此，蝶形运算中仅需一次模乘。

如 [15] 所示，改进的逆数论变换算法不仅将加权运算，还将与常数 n^{-1} 的乘法合并到蝶形运算中。基于公式（15）和频率抽取（DIF）方法，新的逆数论变换算法表示为

$$p_{2k} = (\frac{n}{2})^{-1}\psi_n^{-k} \sum_{j=0}^{n/2-1} \widetilde{P}_j^{(0)} \omega_{n/2}^{-kj} \mod q, \tag{20}$$

$$p_{2k+1} = (\frac{n}{2})^{-1}\psi_n^{-k} \sum_{j=0}^{n/2-1} \widetilde{P}_j^{(1)} \omega_{n/2}^{-kj} \mod q, \tag{21}$$

其中 $k \in [0, \frac{n}{2}-1]$，且

$$\widetilde{P}_j^{(0)} = \frac{\widetilde{P}_j + \widetilde{P}_{j+n/2}}{2} \mod q \tag{22}$$

$$\widetilde{P}_j^{(1)} = \frac{\widetilde{P}_j - \widetilde{P}_{j+n/2}}{2}\omega_n^{-j}\psi_{2n}^{-1} \mod q. \tag{23}$$

同样，整数 ω_n^{-j}和 ψ_{2n}^{-1}组合成整数 $\psi_{2n}^{-1}\omega_n^{-j} = \psi_{2n}^{-(2j+1)}$。与 NTT 蝶形架构不同，iNTT 蝶形中的模加和模减中间结果需要除以二。事实上，乘以 2^{-1} 的模乘可以

无需模乘器 [15], 即可实现，因为

$$\frac{x}{2} = \begin{cases} \frac{x}{2}, & \text{if } x \text{ is even} \\ \lfloor \frac{x}{2} \rfloor + \frac{q+1}{2} \mod q, & \text{if } x \text{ is odd.} \end{cases} \tag{24}$$

具体而言，当x 为偶数时， $x \times 2^{-1}$ 可实现为（$x \gg 1$）。如果 x 为奇数，则 $x \times 2^{-1}$ 可表示为:

$$\begin{aligned} \frac{x}{2} &\equiv (2\lfloor \frac{x}{2} \rfloor + 1)\frac{q+1}{2} \\ &\equiv \lfloor \frac{x}{2} \rfloor (q+1) + \frac{q+1}{2} \\ &\equiv \lfloor \frac{x}{2} \rfloor + \frac{q+1}{2} \mod q \end{aligned} \tag{25}$$

其中， $\lfloor \frac{x}{2} \rfloor$ 也可实现为（$x \gg 1$），而（$q+1$）$/2$ 是一个预计算常数。因此，无需进行模乘，但需要一个模加器和一个多路复用器（MUX）。