计算机研究与发展
Journal of Computer Research and Development
https://crad.ict.ac.cn
62(6): 1547−1561, 2025

基于数据流架构的 NTT 蝶式计算加速

石泓博[1,2] 范志华[1] 李文明[1,2] 张志远[1,2] 穆宇栋[1,2] 叶笑春[1,2] 安学军[1,2]
[1](处理器芯片全国重点实验室(中国科学院计算技术研究所) 北京 100190)
[2](中国科学院大学计算机科学与技术学院 北京 100049)
(Shihongbo0322@outlook.com)

NTT Butterfly Arithmetic Acceleration Based on Dataflow Architecture

Shi Hongbo[1,2], Fan Zhihua[1], Li Wenming[1,2], Zhang Zhiyuan[1,2], Mu Yudong[1,2], Ye Xiaochun[1,2], and An Xuejun[1,2]
[1](*State Key Lab of Processors* (*Institute of Computing*, *Chinese Academy of Sciences*), *Beijing* 100190)
[2](*School of Computer Science and Technology*, *University of Chinese Academy of Sciences*, *Beijing* 100049)

Abstract Fully homomorphic encryption (FHE), which enables computation on encrypted data without decryption throughout the entire processing flow, offers a promising solution for privacy preservation in cloud computing and other distributed environments. However, the practical deployment of FHE remains significantly constrained by its high computational complexity, poor data locality, and limited parallelism. Among the core operations in FHE, the number theoretic transform (NTT) plays a pivotal role in determining overall system performance. We target the butterfly computation pattern, which is central to the NTT algorithm, and propose a high-efficiency NTT accelerator architecture based on a dataflow computing model. First, we design an RVFHE extension instruction set tailored for NTT butterfly operations, incorporating custom modular multiplication and modular addition/subtraction units to enhance the efficiency of modular arithmetic. Second, we introduce a novel NTT data reordering scheme, combined with a structured butterfly address generation strategy, to reduce the control complexity and access conflicts associated with cross-row and cross-column data exchanges. Finally, we develop a dataflow-driven NTT accelerator architecture that leverages data dependency-triggered execution to enable efficient on-chip scheduling and data reuse, thereby exploiting instruction-level parallelism to the fullest extent. Experimental results demonstrate that, compared with NVIDIA GPU, the proposed architecture achieves up to 8.96 times speedup and 8.53 times improvement in energy efficiency. Furthermore, compared with state-of-the-art dedicated NTT accelerators, our design delivers a 1.37 times performance gain.

Key words dataflow; full homomorphic encryption (FHE); NTT algorithm; butterfly computation; RISC-V instruction set

摘 要 全同态加密(fully homomorphic encryption, FHE)因其在计算全过程中保持数据加密的能力,为云计算等分布式环境中的隐私保护提供了重要支撑,具有广泛的应用前景.然而,FHE 在计算过程中普遍存在运算复杂度高、数据局部性差以及并行度受限等问题,导致其在实际应用中的性能严重受限.其中,快速数论变换(number theoretic transform, NTT)作为 FHE 中关键的基础算子,其性能对整个系统的效率具

收稿日期: 2025-03-01; **修回日期**: 2025-04-08
基金项目: 国家重点科技发展计划(2023YFB4503500); 北京市新星计划(20220484054, 20230484420); 北京市自然科学基金项目(L234078); 中国科学院青年创新促进会资助项目
This work was supported by the National Key Research and Development Program of China (2023YFB4503500), the Beijing Nova Program (20220484054, 20230484420), the Beijing Natural Science Foundation (L234078), and the CAS Project for Youth Innovation Promotion Association.
通信作者: 范志华(fanzhihua@ict.ac.cn)

有决定性影响. 针对 NTT 中的核心计算模式——蝶式（butterfly）计算, 提出一种基于数据流计算模型的 NTT 加速架构. 首先, 设计面向 NTT 蝶式计算的 RVFHE 扩展指令集, 定制高效的模乘与模加/模减运算单元, 以提升模运算处理效率. 其次, 提出一种 NTT 数据重排方法, 并结合结构化的蝶式地址生成策略, 以降低跨行列数据交换的控制复杂度与访问冲突. 最后, 设计融合数据流驱动机制的 NTT 加速架构, 通过数据依赖触发方式实现高效的片上调度与数据复用, 从而充分挖掘操作级并行性. 实验结果表明, 与 NVIDIA GPU 相比, 提出的架构获得了 8.96 倍的性能提升和 8.53 倍的能效提升; 与现有的 NTT 加速器相比, 所提架构获得了 1.37 倍的性能提升.

关键词 数据流; 全同态加密; NTT 算法; 蝶式计算; RISC-V 指令集

中图法分类号 TP183

DOI: 10.7544/issn1000-1239.202550160 **CSTR:** 32373.14.issn1000-1239.202550160

全同态加密（fully homomorphic encryption, FHE）[1]是一项关键的数据保护技术, 能够在无需解密的情况下对加密数据直接进行计算, 且计算结果在解密后与对明文执行相同操作所得结果完全一致. 这一特性使 FHE 在实现数据隐私保护方面具有独特优势, 特别适用于在不可信环境中处理敏感信息的应用场景. 在云计算等对数据安全性要求极高的分布式系统中, FHE 技术展现出广阔的应用前景, 成为保障用户数据隐私和系统安全的重要手段. 然而, 尽管 FHE 在理论上具备良好的安全性, 其计算过程复杂度极高, 计算开销远超传统加密算法. 其核心操作涉及大量多项式模加和模乘, 导致数据局部性差、并行性有限, 严重制约了其在实际系统中的高效实现. 据研究表明, 引入 FHE 后, 系统计算性能可能下降 4~6 个数量级[2], 这成为 FHE 技术大规模落地应用的主要瓶颈之一.

因此, 加速 FHE 的计算过程, 已成为推动其在具有较高实时性要求的应用场景中部署的关键研究方向. 为提升 FHE 计算效率, 研究者从软件与硬件 2 个层面提出了多种优化方案[3-13]. 在软件层面, 已有研究聚焦于算法结构优化与噪声控制机制的改进. 例如, Gentry[3] 首次提出了 Bootstrapping 技术, 通过递归解密有效抑制噪声增长; Smart 等人[4] 通过优化密钥生成流程与密文表达方式, 显著降低了加解密计算开销; Brakerski 等人[5] 则引入模切换（modulus switching）技术, 控制噪声积累速率, 减少 Bootstrapping 操作频率, 从而降低了整体的计算与存储负担. 在硬件层面, 针对 FHE 中高开销算子的加速需求, 研究者提出了多种异构硬件加速方案, 包括基于 GPU, FPGA 以及 ASIC 的专用加速架构[14-25]. 例如, Akleylek 等人[15] 提出了一种高效的快速数论变换（number theoretic transform, NTT）实现架构, 通过优化内存访问模式与并行化策略, 提升了多项式乘法的执行效率; Riazi 等人[18] 设计了 HEAX FHE 加速器, 专注于 FHE 基本计算模块的硬件优化; Feldmann 等人[21] 提出 F1 加速方案, 结合模块化算法、NTT 与结构化排列机制, 实现了对 FHE 流程的全面加速.

尽管上述研究在提升 FHE 整体性能方面取得了显著进展, 但 FHE 中的蝶式（butterfly）计算仍面临并行性差、数据依赖强、通信开销大等挑战, 是导致整体计算性能受限的关键瓶颈之一. 因此, 针对蝶式计算模式进行高效并行化与硬件加速设计, 具有重要的研究价值与现实意义.

在 FHE 计算过程中, NTT 中的蝶式计算作为核心计算模块, 发挥着至关重要的作用. FHE 的本质在于对密文所表示的多项式进行加法、乘法及其组合操作, 而 NTT 能够将这些多项式从系数域映射至频域, 转换为点值表示形式, 从而显著降低乘法运算的复杂度, 加速整体计算过程. 在 NTT 的实现过程中, 蝶式计算构成了各层计算的基本单元. 每一层的蝶式操作通过递归地执行加法、乘法与旋转因子的乘积计算, 完成对数据的逐级变换, 实现高效的频域变换过程. 该结构不仅提高了多项式加法与乘法的并行处理能力, 还有效保持了密文结构的同态性, 支持加密数据在不解密的前提下直接参与计算.

针对蝶式计算中存在的计算模式复杂和数据依赖性强等问题, 本文引入数据流计算模型（dataflow computing model）[26], 通过挖掘程序的多层次并行性, 并将指令级数据依赖转化为数据流图, 实现对蝶式计算挑战的高效应对. 具体而言, 本文从指令集设计、数据重排策略及体系结构优化 3 个层面展开研究. 首先, 设计了面向 NTT 蝶式计算的 RVFHE 扩展指令集, 针对核心模运算（如模乘、模加等）, 定制了高效的专用计算部件及其数据通路, 提升了指令执行效率. 其次, 提出了蝶式数据重洗（butterfly data re-shuffling）方

法，并结合结构化蝶式地址生成机制，有效简化了蝶式计算中复杂的数据预处理与跨行列数据交换操作，通过重洗机制实现数据重排与取数操作的协同优化. 最后，构建了基于数据流计算模型的 NTT 加速微架构. 该架构采用包含 16 个 NTT 计算核的二维阵列结构，并在每个计算核内部集成 32 套并行模运算部件，支持高吞吐的蝶式并行计算. 架构通过数据依赖驱动的触发机制，实现高效的片上调度和数据复用，有效缓解了传统架构在数据依赖处理上的瓶颈. 本文的主要贡献有 3 个方面：

1）指令集层面. 设计了面向 NTT 蝶式计算的 RVFHE 扩展指令集，包含模乘、模加等关键操作，并配套设计了专用的计算部件与数据通路，提升模运算执行效率.

2）数据布局层面. 提出了 NTT 数据重洗与蝶式地址生成策略，充分考虑蝶式运算过程中的分治结构和数据蝶变特性，有效降低数据访问冲突与控制复杂度.

3）体系结构层面. 提出了融合数据流模型的 NTT 加速微架构，通过将数据依赖关系映射为计算触发条件，显著提升了操作级并行度和片上资源利用效率.

实验结果表明，与 NVIDIA GPU 平台相比，本文所提出的加速架构在执行 NTT 蝶式计算任务时，实现了 8.96 倍的性能提升与 8.53 倍的能效提升；与当前主流 NTT 专用加速器相比，取得了 1.37 倍的性能提升，验证了所提方案在性能和能效方面的优势.

1 研究背景

FHE 是一种支持在加密数据上直接执行运算的加密技术，其显著特性在于：对密文执行的计算操作，解密后所得结果与在对应明文上执行相同操作所得结果完全一致. 这一特性赋予了 FHE 在隐私保护计算中的关键价值，尤其适用于需要在不泄露原始数据的前提下进行复杂处理的场景. 典型应用包括云计算[27−29]、医疗数据分析[30]、金融计算[31] 等领域. 如图 1 所示，FHE 支持用户在本地对原始数据进行加密，并将密文上传至云端进行计算处理，计算过程全程在加密域中完成，最终用户只需对密文结果解密，即可获得正确的明文计算结果，从而确保数据隐私的同时实现功能性计算.

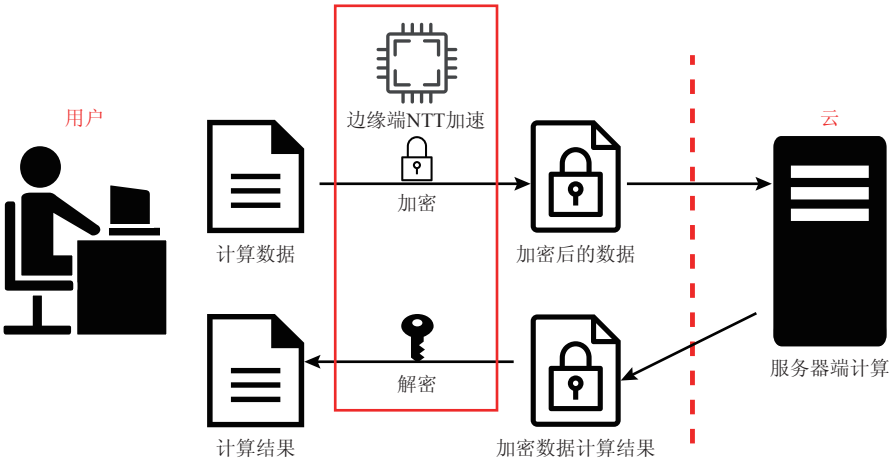


Fig. 1 Application scenario of FHE

图 1 FHE 应用场景

NTT 算法[32] 可以降低 FHE 计算的时间复杂度. FHE 计算中涉及大量的多项式乘法运算，其时间复杂度为 $O(n^2)$，是 FHE 中最耗时的操作. 为提升多项式乘法的计算速度，常用的做法是采用 NTT 算法，将多项式乘法的时间复杂度从 $O(n^2)$ 降为 $O(n\log n)$.

$$C(x)=A(x)\cdot B(x)=\sum_{k=0}^{2n-2}c_k x^k. \tag{1}$$

式(1)展示了多项式乘法的计算过程，其中 $A(x)$ 和 $B(x)$ 为输入的多项式，c_k 为多项式计算结果 $C(x)$ 的系数，满足：$c_k=\sum_{i=0}^{k}a_i b_{k-i}$. 直接计算 c_k 的值需要进行两重求和，时间复杂度为 $O(n^2)$. NTT 算法通过利用频域特性，将多项式乘法的问题转化为点值形式，进行点值计算，从而实现加速. 首先，利用 NTT 将输入多项式 $A(x)$ 和 $B(x)$ 转化为它们的点值形式. 在有限域 F_q 上，NTT 使用 ω 作为原根，对多项式 $A(x)$ 和 $B(x)$ 进行变换，得到多项式 $A(\omega^k)$ 和 $B(\omega^k)$ 的值. 将转换后的

点值进行逐点相乘, 即计算$C(\omega^k)=A(\omega^k)\cdot B(\omega^k)$对所有$k$的乘积. 最后, 使用逆 NTT 将点值形式的结果转换回系数表示, 从而得到多项式乘法计算结果$C(x)$. NTT 变换的时间复杂度为$O(n\log n)$, 因此通过加速 NTT 运算可以显著提高 FHE 的计算效率.

在 NTT 计算中, 蝶式计算是不可或缺的部分, 其数学表达为:

$$a'=\left(a+\omega^k\cdot b\right)\bmod q, \tag{2}$$

$$b'=\left(a-\omega^k\cdot b\right)\bmod q, \tag{3}$$

其中, a和b是输入数据点, ω是模q下的N次原根($\omega^N\equiv 1\bmod q$), k为当前阶段的旋转因子指数, 由分解阶段决定, q为质数模数, 满足$N|q-1$.

图 2 展示了$N=8$时的蝶式计算过程, 分为$\mathrm{lb}N$个阶段, 每个阶段的蝶式跨度逐级倍增. 图 2 中, 每个阶段由多组蝶式单元(虚线方框)构成, 黑色线条表示数据流向, 蝶式单元旁标注了当前阶段所乘旋转因子ω^k的值. 每个阶段需要进行$N/2$次蝶式运算, 总复杂度为$O(N\log N)$, 每次蝶式计算需要进行 2 次模乘和 2 次模加减.

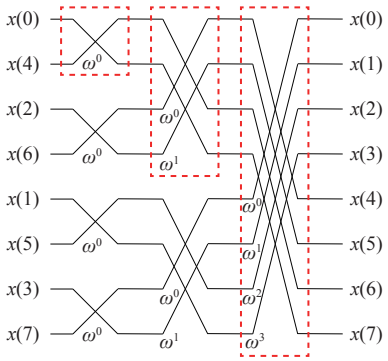


Fig. 2 Butterfly computation process

图 2 蝶式计算过程

蝶式计算中的旋转因子ω^k是 NTT 计算中不可缺少的部分, 计算ω^k涉及对有限域上的原根的幂次运算, 这意味着每一步都需要通过模运算来确保计算结果处于有限域内. 因此, 模运算的计算开销是制约蝶式计算的原因之一. 同时, 蝶式计算本质上是串行的, 每一轮的计算都依赖于上一阶段的计算结果, 因此存在显著的数据依赖关系, 导致了数据重洗需求. 由于数据的依赖性, 传统的并行方法无法充分利用多核处理器的计算能力. 并且, 在进行蝶式计算时, 大量的数据需要在不同的计算阶段之间传递, 这导致处理器产生大量的内存访问和计算延迟. 综上, 蝶式计算由于其复杂的计算过程和数据依赖, 制约了 NTT 算法对 FHE 的加速效果.

2 相关工作

目前, 主流的 FHE 加速方法主要集中在软件层面和硬件层面.

软件层面上, 多种对 FHE 计算的优化方案被提出[3-13]. Gentry[3] 的方案基于理想格, 引入了 Bootstrapping 技术, 通过递归解密来减少噪声增长, 从而支持无限次同态操作. Smart 等人[4] 在 Gentry[3] 方案的基础上, 通过优化多项式环上的计算, 减少了密钥和密文的存储和传输开销, 通过引入了分圆域的概念, 进一步优化了计算效率. Brakerski 等人[5] 引入了层次化 FHE 的概念, 通过使用模切换技术, 降低模数来控制噪声增长, 减少了 Bootstrapping 的频率, 降低了计算和存储的开销. Cheon 等人[13] 引入了重缩放技术, 通过近似计算和噪声控制, 进一步优化了计算效率.

硬件层面上, 基于多种加速架构(如基于 GPU, FPGA, ASIC)的硬件加速架构被提出[14-25]. 表 1 展示了一些具有代表性的硬件加速架构.

Table 1 Features of Hardware Accelerated Architecture

表 1 硬件加速架构的特点

硬件加速架构	计算平台	特点
TensorFHE[14]	GPU	最大化数据复用并减少片外数据移动
cuHE[17]	GPU	多 GPU 配置
HEAX[18]	FPGA	层次化内存设计
F1[21]	ASIC	可编程 FHE、无界计算
CraterLake[22]	ASIC	硬件架构、功能单元、算法和编译器技术
本文架构		针对 NTT 蝶式计算进行优化

在基于 GPU 的加速方案方面, Fan 等人[14] 提出了 TensorFHE, 一种面向 FHE 的高效加速方案. 该方案设计了 3 种数据流策略(Max-parallel, Digit-Centric, Output-Centric), 旨在最大化数据复用并降低片外内存访问, 还充分利用 GPU 的并行计算能力以加速 NTT 运算. Akleylek 等人[15] 提出了高效的 NTT 实现方案, 主要通过优化内存访问模式和并行策略来提升多项式乘法性能, 同时优化了 CUDA 的线程分配机制、内存层次结构和指令调度, 实现了更高效的 GPU 运算. Dai 等人[17] 利用多 GPU 协同计算提升计算性能, 并结合内存最小化技术和流调度策略, 显著减少了数据在设备间移动所带来的开销.

在基于 FPGA 的加速方案中, Riazi 等人[18] 提出了模块化的硬件加速架构 HEAX, 面向 FHE 中的 NTT

和模运算性能瓶颈. HEAX 通过层次化的内存系统设计(包括高速缓冲和分布式存储)减少了数据移动开销, 并集成了专用计算单元以加速核心运算流程. Roy 等人[19] 针对异构 ARM+FPGA 平台, 设计了一种特定领域架构, 结合算法数据依赖性分析与电路级流水线技术, 实现了计算吞吐量的提升. Pöppelmann 等人[20] 基于带误差的学习(RLWE)问题提出了一种加速器架构. 为解决大规模密文处理中的存储瓶颈, 该方案引入了高效的双缓冲存储访问机制, 以优化外部内存的带宽利用效率, 加速了整体访存过程.

在专用加速器方面, Feldmann 等人[21] 提出了一款可编程的全同态加密加速器 F1. 该架构支持无界计算, 通过引入新的硬件结构与功能单元, 能够处理任意规模的 FHE 程序, 同时结合编译器优化, 有效减少数据移动, 提高整体计算效率. Samardzic 等人[22] 提出了 CraterLake, 该加速器设计重点在于实现全同态乘法的深度无界性. 该方案通过联合设计新的硬件架构、功能单元、算法和编译技术, 有效扩展了密文规模, 降低了计算过程中数据移动带来的开销. Karabulut 等人[23] 提出了一种基于 RISC-V 的架构, 用于加速 NTT 计算, 并针对特定运算模式进行了定制优化. 该方案引入存储器相关性预测机制, 有效降低了访存延迟和访问冲突. Lu 等人[25] 面向大规模数据场景, 设计了专用的内存访问单元, 并提出了一种无冲突的内存映射算法, 从而显著提升了 NTT 的并行处理能力与吞吐率. 本文的设计聚焦于 FHE 计算过程中的加速, 分别从核内计算单元优化、数据流读写模式重构以及数据流微架构设计 3 个层面展开研究, 旨在提升 NTT 蝶式计算的效率, 缓解其计算与数据依赖带来的性能瓶颈.

现有的软件优化方案与硬件加速架构在一定程度上提升了 FHE 的计算性能, 但针对 NTT 蝶式计算所引发的复杂计算过程与数据依赖问题仍缺乏有效的解决手段. 蝶式计算作为 NTT 的核心计算单元, 存在以下关键挑战: 首先, NTT 蝶式计算始终在模 q 的有限域中进行, 为避免数值溢出并保证计算正确性, 计算过程中涉及大量高频率的模乘和模加操作, 显著增加了运算负载, 制约了系统的加速效果. 其次, 蝶式计算的非连续访存模式导致数据局部性较差, 为实现有效计算, 需进行复杂的数据预处理与重排操作, 进一步增加了访存开销. 此外, NTT 蝶式计算具有明显的递归结构, 其非规则的内存访问行为导致了跨层级的数据依赖, 严重影响了并行计算的效率与片上资源的调度能力. 上述问题不仅成为 FHE 性能进一步提升的主要瓶颈, 也限制了其在高实时性、高吞吐需求场景中的应用部署. 因此, 围绕蝶式计算展开架构层面的创新设计, 以缓解其计算复杂性与数据依赖问题, 已成为当前 FHE 加速领域的重要研究方向.

3 优化方法

基于对 FHE 计算与数据流计算模型的深入分析, 本文提出了一种基于数据流架构的 FHE 加速方法. 该方法充分利用数据流架构的优势, 从以下 3 个方面对 FHE 进行加速:

1)核心模运算加速. 在蝶式计算中的核心模运算环节, 针对密集的模运算本文设计了专用的模运算器, 并通过扩展 RVFHE 指令集, 实现计算核内部模运算的加速, 从而显著提高计算效率.

2)数据预处理优化. 在蝶式计算中的数据预处理阶段, 为了减少由复杂数据预处理带来的延迟, 本文提出了一种数据重洗方法. 该方法将数据预处理与访存操作紧密融合, 提高了计算核的计算资源利用率, 并有效减少了计算延迟.

3)数据依赖优化. 在蝶式计算中的数据依赖问题上, 本文设计了优化的数据流架构, 重新优化了数据流的传输路径. 通过将数据依赖转化为计算的触发条件, 充分挖掘数据的并行性, 实现高性能计算.

3.1 RVFHE 指令集与部件设计

图 3 展示了在不同数据规模下, 模运算在 NTT 蝶式计算中的时间占比. 模乘运算的平均用时占比为 47.63%, 模加运算的平均用时占比为 24.71%, 模减运算的平均用时占比为 24.45%. 模运算在不同的数据规模下, 均为计算开销最大的部分, 是计算过程中的主要瓶颈.

针对模乘、模加、模减运算, 本文设计了 RVFHE 扩展指令集. 表 2 展示了基于 RISC-V 扩展的指令: 数

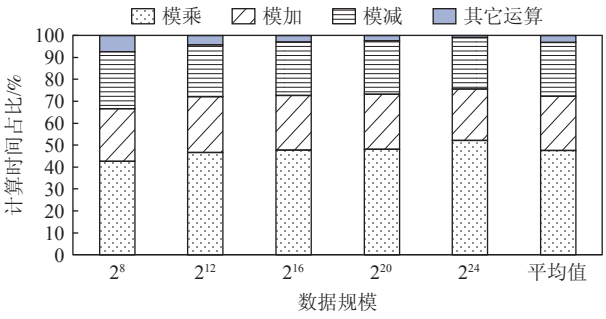


Fig. 3 Time proportion analysis diagram for modular operation

图 3 模运算时间占比分析图

据预取指令、模运算指令(模加、模减、模乘)、数据传输指令. 数据预取指令将数据与目的寄存器对应. 模运算指令分别完成模加、模减、模乘操作. 数据传输指令用于将计算后的数据传输至对应的计算核.

模加、模减、模乘指令长度均为 32 b, 0~6 b 为 Opcode(即指令字段), 用于识别指令的类型, 此处由于均为模指令, 因此 Opcode 均采用了 0x2b, 指令执行的区分由 Funct7 进行区分: 7~11 b 为操作数 2, 用于确定目的寄存器下标; 12~14 b 为 Funct3, 用于选择不同的模数; 15~19 b 为操作数 0, 用于确定被加数、被减数或被乘数的寄存器下标; 20~24 b 为操作数 1, 用于确定加数、减数或乘数的寄存器下标; 25~31 b 为 Funct7, 用于区分该模运算进行的是模加(Funct7 取 0)、模减(Funct7 取 1)还是模乘(Funct7 取 2).

图 4 展示了 NTT 计算核中扩展的计算数据通路. NTT 计算核中扩展了如图 4 上面部分所示组合模运算器. 其中, 被加/减数 x 作为模加/减器的第 1 个输入值, 模乘的输出 y' 作为模加/减器的第 2 个输入值. 由于分别执行了 1 次模加和 1 次模减, 因此可以得到 2 个输出值, 即模加输出 r_1 和模减输出 r_2. r_1 和 r_2 将作为

Table 2 RVFHE Extended Instruction Set

表 2 RVFHE 扩展指令集

指令	Funct7[31:25]	Rs2[24:20]	Rs1[19:15]	Funct3[14:12]	Rd[11:7]	Opcode[6:0]
数据预取(LDP)	0		数据地址	输入数据大小	目的寄存器	0x2a
模加(MADD)	0	操作数 1	操作数 0	0	操作数 2	0x2b
模减(MSUB)	1	操作数 1	操作数 0	0	操作数 2	0x2b
模乘(MMUL)	2	操作数 1	操作数 0	0	操作数 2	0x2b
数据传输(COPY)	0	源计算核	待传输数据	输入数据大小	目的计算核	0x2c

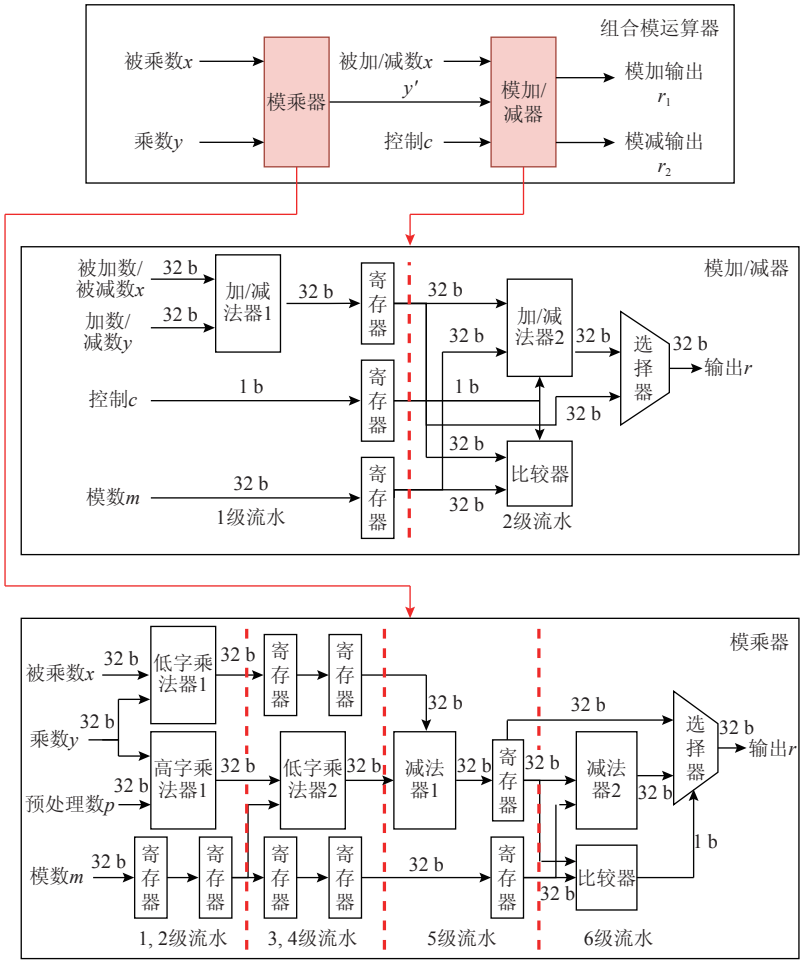


Fig. 4 Schematic diagram of the data path of the combined modulo operator

图 4 组合模运算器数据通路示意图

本次计算的结果，在本次计算完成后将会对各个计算数据重新分组，作为下一次计算中被乘数x和乘数y这 2 个输入的数据. 当所有层的所有数据都计算完成后，即可存回片上的存储中.

图 4 中间部分展示了模加/减器的硬件结构图. 模加/减单元的输入端为被加数/被减数x、加数/减数y、控制c和模数m. 设机器字长为M，则模数m的取值范围为$0 \leqslant m \leqslant 2^{M-1}$. 被加数/被减数$x$的取值范围为$0 \leqslant x < m$，加数/减数$y$的取值范围为$0 \leqslant y < m$. 控制$c$用来控制如何使用加/减法器，决定了该次的运算是做模加还是模减运算. 模加/减器的输出端为模加/减结果r，即$r = x + y \bmod m$或$r = x - y \bmod m$.

模加和模减运算中，执行加减的顺序和位置相反. 对于模加运算，先执行一次加法，在加法结果超过模数m的时候，需要再执行一次减法，减去模数m. 由于被加数x和加数y都是小于模数m的，因此此时的结果一定是满足取模条件的. 同理，对于模减运算，先执行一次减法，在减法结果小于 0 的时候，需要再执行一次加法，加上模数m. 由于被加数x和加数y都是小于模数m的，因此加上模数m后，结果一定是大于 0 的. 通过控制c，在模加时，控制加/减法器 1 为加法，加/减法器 2 为减法，模减时相反，加/减法器 1 为减法，加/减法器 2 为加法；比较器在模加时，用于比较输入是否大于模数m，在模减时，用于比较输入是否小于 0.

图 4 的下面部分展示了模乘单元结构. 模乘单元的输入端为模数m、被乘数x、乘数y和预处理数p. 设机器字长为M，则模数m的取值范围为$0 \leqslant m \leqslant 2^{M-1}$，被乘数$x$的取值范围为$0 \leqslant x < m$，乘数$y$的取值范围为$0 \leqslant y < m$，预处理数$p = \left\lfloor \frac{x \cdot 2^M}{m} \right\rfloor$. 预处理数$p$的值由被乘数$x$和模数$m$确定，保存了乘数中的高位部分. 模乘单元的输出端为模乘结果r，即$r = x \cdot y \bmod m$.

模运算器实现了模乘的运算过程. 在图 4 的下面部分中，虚线分割了 6 个流水线级别，左侧的 2 个区域中分别包含了 2 个流水级，右侧的 2 个区域中分别包含了 1 个流水级别. 第 1 级和第 2 级流水中的低字乘法器 1 完成了被乘数x和乘数y相乘的结果的低M位，高字乘法器 1 完成了取$\frac{x \cdot y}{m}$的高M位. 低字乘法器 1 内含 2 级寄存器，输入被乘数x和乘数y，输出乘法结果的低M位. 高字乘法器 1 同样内含 2 级寄存器，输入乘数y和预处理数p，输出乘法结果的高M位. 第 3 级和第 4 级流水完成了取$m \cdot \left\lfloor \frac{y \cdot p}{2^M} \right\rfloor$的运算结果. 低字乘法器 2 输入模数$m$和高字乘法器 1 的输出结果，输出乘法结果的低$M$位. 第 5 级流水完成了乘法结果和取整结果相减，减法器 1 实现了减去取整后的运算结果. 第 6 级流水中还需要再用比较器将计算结果与模数m进行比较，判断是否需要再做一次减法. 最终的输出结果即为取模后的值.

模乘器的实现需要减小变量范围. 当输出结果r的取值范围过大时，能够使用的模数范围会非常小. 在 FHE 中的一次特定的模乘运算中，其中一个输入可以预先计算出来，即对于该次模乘运算，有一个输入是定值. 因此，考虑采用固定模乘的一个输入，使用单变量的模乘，从而能够使用更大的模数，提升计算效率.

对于模乘运算$r = x \cdot y \bmod m$，令$x \cdot y = km + r$，通过求出k，再用$x \cdot y$减去km即可求出模运算结果. 因此，可以令$k = \left\lfloor \frac{x \cdot y}{m} \right\rfloor$得到$k$值. 但是，除法的时间开销过大，会显著延长计算的时间.

本文设计的模乘器首先将被乘数x和乘数y相乘的结果取低M位，赋给输出r. 然后，对乘数y进行单独处理，将乘数y与预处理数p相乘，并取高M位赋值给k. 其中，预处理数p由被乘数x和模数m确定，即$p = \left\lfloor \frac{x \cdot 2^M}{m} \right\rfloor$，因此$k$保存了$\frac{x \cdot y}{m}$的高$M$位. 然后，需要再将$k$和模数$m$进行一次取低$M$位的乘法. 此时，$k$中保存的结果为$m \cdot \left\lfloor \frac{y \cdot p}{2^M} \right\rfloor$，将$r$与$k$相减即可求出模乘结果. 由于在计算过程中，进行了 2 次向下取整，所以存在最后求出的结果r比模数m大的情况，因此需要再比较一次，如果结果r>m，就需要再完成一次减法操作.

3.2 数据重洗与地址生成优化方法

在 NTT 蝶式计算中，非连续访存模式导致数据局部性较差，需要在展开计算前进行复杂的数据预处理操作. 为降低 NTT 蝶式计算引起的数据预处理开销，本文设计了数据重洗与地址生成方法，将数据预处理与访存操纵融合.

图 5 展示了数据重洗的流程. 数据重洗需要循环 2 次，第 1 次循环先生成一个变换数组，用于标记蝶式计算中最终需要变换到的位置，第 2 次循环遍历整个待变换的数据. 首先，获取待计算的数据规模，并向上取整到最小的 2 的幂次，保证蝶变到最后一层时，每组都有 2 个数据进行计算. 其次，通过将下标i的二进制位反转来确定计算次序. 先计算i/2 的反转位，从 *temp* 数组中获取该反转位，将其右移 1 位得到 *length*−1 位的数，再对最高位补 0，最终得到i的反转位. 如果当前遍历的i为奇数，则将 *temp*[i] 的最

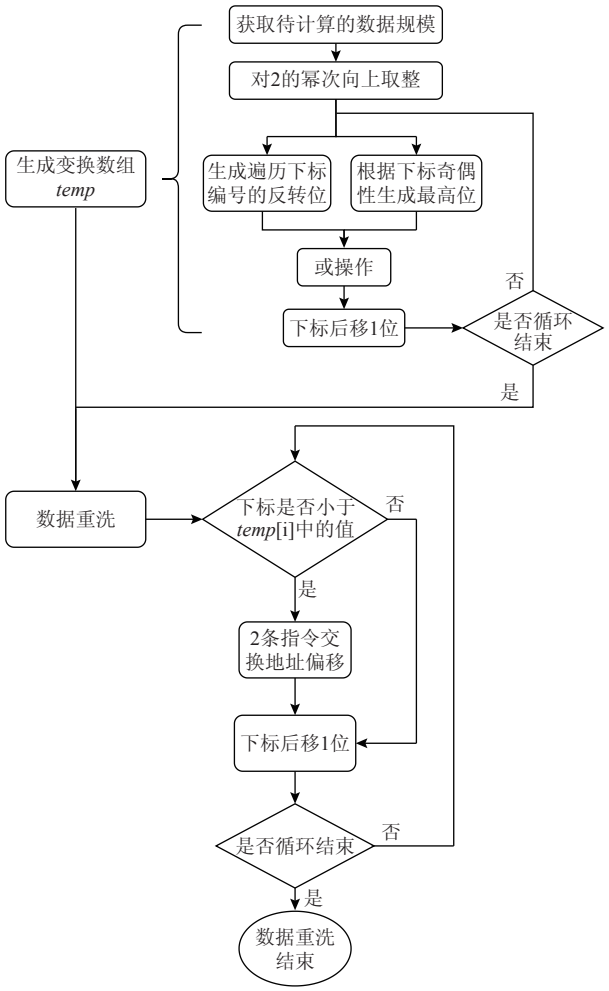


Fig. 5 Data reordering process

图 5 数据重洗流程

高位设置为 1, 否则最高位为 0(针对奇偶性进行判断). 再将上述变换后得到的 2 个二进制结果通过或操作组合起来, 得到 i 的反转. 对每一个 i 都做上述的操作, 就能得到蝶式计算最终层的计算数据排列顺序.

第 2 次循环遍历的过程中会生成地址, 将数据预取指令的数据下标 i 与变换数组 *temp* 进行对比, 若数据预取指令下标较小, 则需要将 *temp*[i] 对应的数据在片上的偏移写入本条正在比较的数据预取指令中. 通过这种方式, 可以确保在数据重排时, 每个数据项的存储位置已经按照预期的顺序进行了更新. 当所有的数据重排完成后, 按照顺序执行数据预取指令. 此时, 由于数据已经预先按正确的顺序进行了存储和重排, 可以直接展开计算, 避免了不必要的延迟, 并提高了计算效率. 这种方法通过优化数据预取和重排, 减少了计算中的数据访问延迟, 确保了计算过程的流畅进行.

为挖掘 NTT 蝶式计算的并行性并充分发挥数据重洗的特点, 本文采用分治计算策略, 在多个计算核上进行并行处理, 从而提高核间计算的并行性.

具体来说, 本文将较大数据规模的计算数据划分为 n 份, 分别分配给 n 个计算单元, 展开分治并行计算. 通过采用并行 NTT 算法, 可以将一次大数据规模的 NTT 计算划分为 4 个步骤:

1)对输入的矩阵数据的每一列进行 NTT 运算;

2)将列 NTT 结果矩阵的每一元素乘上对应的旋转因子;

3)对乘上旋转因子后的结果矩阵进行转置;

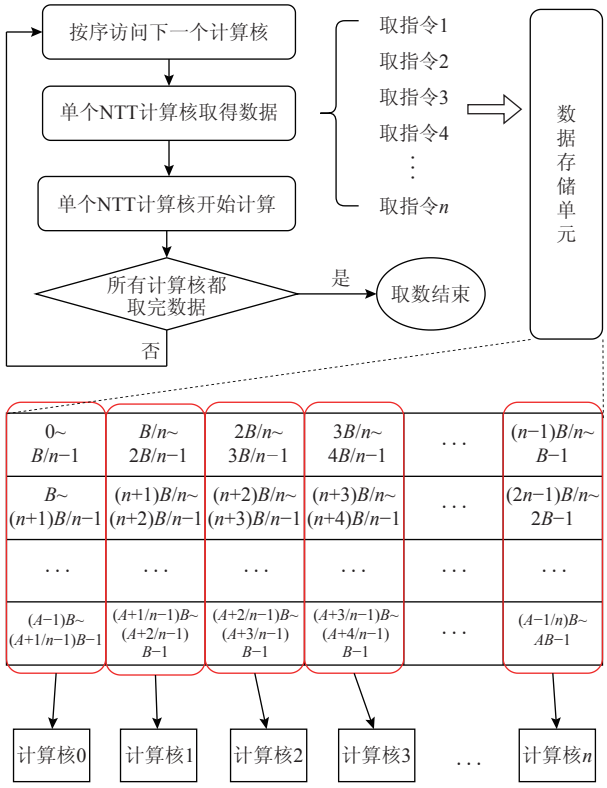
4)对转置后的矩阵再次进行列 NTT 运算.

通过这一分治策略, 能够有效地利用多个计算核进行并行计算, 从而提升 NTT 计算的性能. 经过一次转置后, 分治计算的结果与采用单计算核对整个矩阵进行的 NTT 蝶式计算结果相同. 因此, 通过这种方式, 各计算单元可以展开分治计算. 并且, 当$n = A \times B$中的A, B是合数的条件下, 仍能继续对A或B进行拆分. 由于合数分解的方式是任意的, 因此理论上任意规模的 NTT 都可以分解为规模更小的行列形式的 NTT 运算, 进而在有限运算资源的条件下完成任意规模的运算.

图 6 展示了计算单元的取数映射, 每个计算核对应取一列数据. 由于各核所取的数据规模相同, 因此可以只执行 1 次数据重洗的过程. 实际运行中, 通常会将行列的长度设定为 2 的幂次, 以确保蝶式变换时能够进行等长的拆分. 通过预取指令将数据加载到计算核上后, 可以立即展开计算, 无需对计算对象进行重排. 这样可以有效提高计算效率并减少不必要的计算开销.

3.3 数据流优化

NTT 蝶式计算的递归特性和非规则访存模式导致了计算过程中频繁的跨层数据依赖. 为了解决这一问题, 本文设计了一个包括控制核、指令存储单元、数据存储单元、片上计算阵列和片上网络的数据流架构. 该架构如图 7 所示, 其中控制核包含 I/O 接口和控制单元, 负责协调整体计算过程以及管理指令的发放和计算单元的调度. 指令存储单元则用于存储编译完成后的指令, 确保计算核按照预定流程进行计算. 同时, 数据存储单元用于存储大规模的计算数据, 以满足高带宽的数据访问需求. 片上计算阵列中包含 16 个 NTT 计算核, 采用分布式计算的方式执行 NTT 蝶式计算任务. 每个计算核执行特定的计算任务, 从而提升计算的并行性. 此外, 片上网络由 3 套网络组成: 控制网络、数据网络和存储网络. 控制网络负责计算节点之间的通信, 协调任务分配和资



注：A 是行的个数，B 是列的个数.

Fig. 6 Data access mapping relationship of the computing unit

图 6 计算单元的取数映射关系

源调度；数据网络则负责在计算核之间传输数据，确保各计算核能够同步展开计算；存储网络用于访问存储单元，确保数据能够高效地从存储单元加载到计算核. 这种设计有效应对了 NTT 计算中的数据依赖问题，通过多层次的片上网络协同工作，提高了计算效率和数据吞吐量，同时保障了计算任务的并行性和同步性.

在每一个 NTT 计算核中，设计了多个关键部件，包括路由、取指/译码单元、存储单元和执行运算部件. 路由部分包含控制缓存、地址匹配器和发射单元，负责数据的传输和匹配. 控制缓存用于存储数据转发计算核的地址. 在 NTT 蝶式计算中，由于计算节点之间存在数据依赖关系，各计算核需要频繁地通过片上数据网络交换数据. 地址匹配器则用于匹配本计算核需要的数据. 当 NTT 计算核接收到其他计算核传输的数据后，地址匹配器会根据需要将数据存储到相应的寄存器中. 发射单元在计算结果经过地址匹配器匹配后，保留本计算核所需的数据，并将其他计算核所需的数据发送到数据网络中，以供其他核使用. 取指/译码单元由指令存储器、译码器和地址生成器组成. 指令存储器用于存储 NTT 计算核中待执行的指令. 译码器负责将存储的指令进行解码，并将需要重新生成地址的指令传输给地址生成器. 地址生成器则根据预定的地址生成方法，在指令重洗过程中确定计算数据的寄存器下标，以确保计算核能够正确地访问和操作数据.

NTT 计算核从片上数据存储中得到的数据存储在寄存器组中. 在计算执行过程中，由于地址生成器重新生成了计算数据的下标地址，因此会乱序地从寄存器组中获取计算数据. 执行单元中包括整型运算部件和模运算部件，其中整型运算部件负责 NTT 蝶式计算中产生的辅助参数计算. 模运算部件由 32 套模运算组合构成，每一套组合能够同时对 32 组计算数据进行模运算，从而显著提高了计算并行性. 每套模运算组合中包括一个模乘器和一个模加/减器. 模乘器的输出作为中间过程的输入传递给模加/减器，模加/减器将执行一次模加运算和一次模减运算，所得的 2 个输出结果即为当前计算的最终结果. 这 2 个结果将分别存储回寄存器组中的原始地址，供后续计算或数据传输使用.

图 8 展示了地址匹配器的映射关系，左上角的数

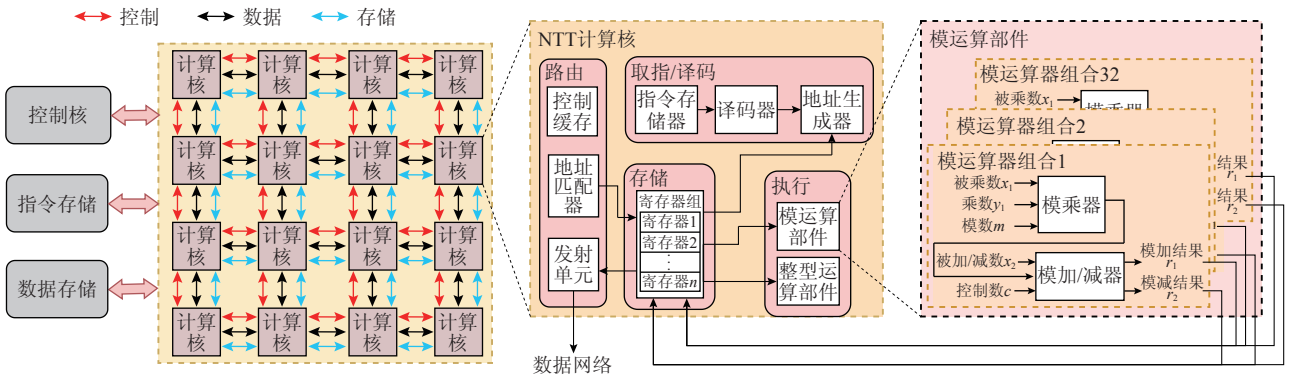


Fig. 7 Dataflow architecture of RVFHE

图 7 RVFHE 数据流架构

据矩阵表示每个 NTT 计算核中的数据排列顺序. 细箭头线表示了 NTT 计算阵列与数据矩阵的对应关系. 当计算核收到其他计算核传输的数据后，按照矩阵中相同位置的规则，将本计算核所需的数据存入相应的寄存器中. 每个计算核都需要从其他 15 个计算核接收数据，因此，只需按照计算核下标将存在数据依赖关系的数据存入寄存器即可. 在分治的 NTT 蝶式计算中，转置后的蝶变顺序与第 1 次相同. 因此，当单个计算核所需的数据收集完成后，就可以展开计算. 图 8 还展示了数据传递的方式. 依托于片上数据网络，计算核将计算后的数据传输给相邻的计算核. 每个计算核根据矩阵中的数据对应关系，将数据传递给其他计算核. 由于各计算核加载并展开计算的时间不同，因此计算核计算数据与数据在片上传输存在时间和空间上的重叠，从而能够掩盖部分传输过程中数据网络的拥塞和传输延迟.

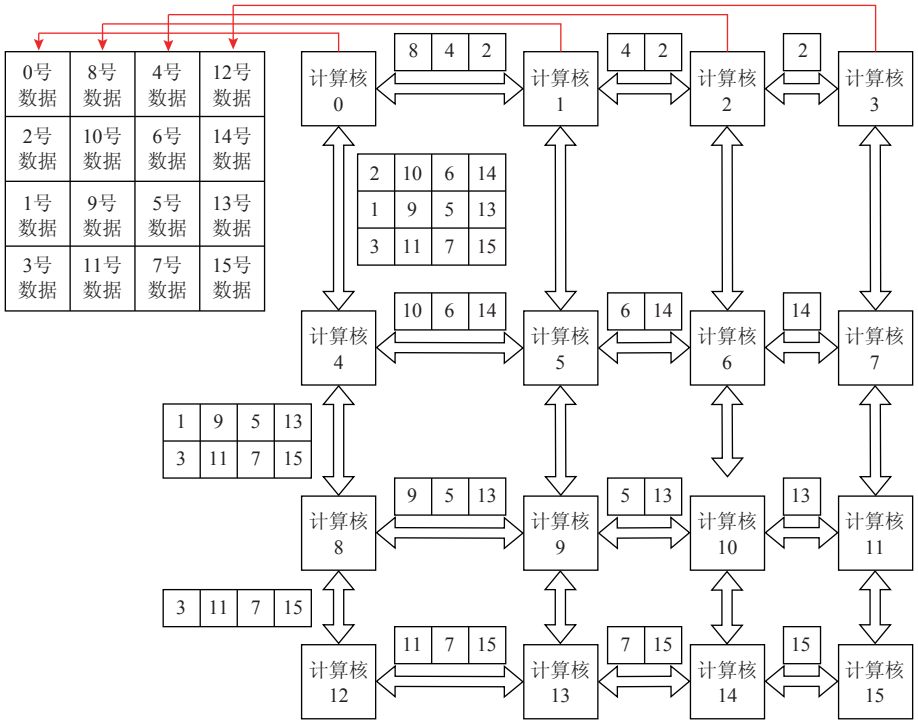


Fig. 8 Mapping relationship diagram of address matcher

图 8 地址匹配器映射关系图

4 实验设置

本文使用开源的 Rocket 架构[33] 作为 base 核，并在 base 核内做了优化拓展，增加了 RVFHE 的扩展指令集. 为挖掘 NTT 蝶式计算的并行性，本文将单核扩展为4×4规模的计算核阵列.

本文使用 Verilog 语言实现了数据流众核架构的 RTL 级仿真，并利用 Synopsys Design Compiler 工具进行综合，采用 12 nm 工艺获取功耗和面积数据. 此外，本文基于 SimICT 并行框架[34] 实现了数据流众核架构的模拟器. 该模拟器主要用于性能和计算资源利用率的评估，能够精确模拟指令的执行过程、内存访问和指令冲突等. 实验过程对模拟器和仿真平台进行了校准，模拟误差可控制在 7% 以内[35]. 数据流众核架构的配置参数如表 3 所示.

在实验评估方面，本文首先进行了消融实验，以 Rocket 原核为基础进行对比，得到了在不同数据规模下 NTT 计算的计算性能. 然后，选取 NVIDIA GPU A100、HEAX[18] 和文献 [24] 工作作为对比平台进行实验. 以计算时长作为加速性能衡量指标. 能效的定义为单位功率的计算性能，单位为 GOPS/W.

Table 3 Parameter Configuration of Dataflow Acceleration Architecture

表 3 数据流加速架构参数设置

模块	配置信息
计算核	16 KB 指令缓存、144 KB 数据缓存、1 GHz、SIMD32、1 TOPS（INT32）
片上网络	2D Mesh、1 套核间通信网络、1 套控制网络、1 套访存网络
片上存储	SPM、Ping-Pong、3 MB
访存带宽	32.00 GB/s

5 结果分析

5.1 消融实验

图 9 展示了本文扩展后的数据流众核架构与 base 核在不同数据规模下运行 NTT 计算所需时钟周期数(cycles)的对比，并针对本文提出的 3 项优化方法进行了消融实验，以验证各优化策略的加速效果. 在各数据规模下，本文设计的数据流众核架构相较于 base 核展现了显著的性能提升：通过扩展指令集与部件设计，平均加速比为 29.44；优化读写方式(RVFHE)带来了 1.3 的加速比；优化数据流策略(RVFHE+Reshuffle)实现了 3.37 的加速比；本文方案使得整体性能提升了平均 140.75 倍. 随着数据规模的增大，优化效果进一步显著提升. 当数据规模为 2^{14} 时，整体加速效果达到了 213.29 的加速比. 上述结果表明，本文方法在加速 NTT 计算过程中具有明显的效果，且随着数据规模的增加，性能收益更加明显.

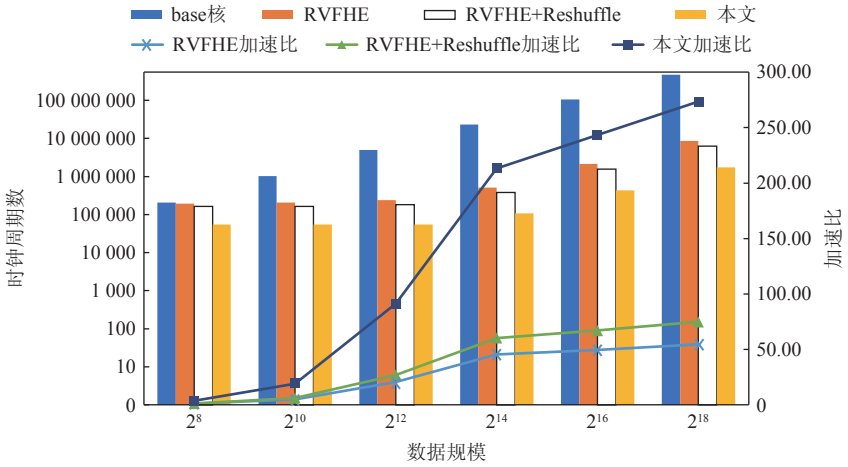


Fig. 9 Aablation experiments

图 9 消融实验

5.2 性能对比分析

图 10 展示了本文设计的数据流众核架构与 GPU A100、HEAX[18] 以及文献 [24] 在不同数据规模下的计算时间对比. 本文测试了数据规模在 2^{10}~2^{14} 时各平台的计算时间，并在数据规模为 2^{12}~2^{14} 时与 HEAX 进行比较，在数据规模为 2^{10}~2^{12} 时与文献 [24] 进行比较. 由于数据规模的变化，不同数据规模的计算时间差异较大，无法直接进行横向比较，因此本文以 GPU 的计算时间为基准，对各平台的计算时间进行了归一化处理. 从图 10 中可以看出，数据流处理器在计算时间上相比于 GPU、HEAX[18] 和文献 [24] 表现出显著的优势，计算时间显著缩短，计算时间开销有效减小. 尤其是在数据规模较大时，计算时间减少更为显著. 本文设计的数据流众核架构支持的最大片上计算数据规模为 2^{13}，在此数据规模下，计算资源的利用率达到最大，优化效果最为明显. 与 GPU 相比，本文设计的数据流众核架构在各数据规模下均实现了平均 8.96 的加速比，并与目前最先进的 NTT 加速器对比时，取得了 1.37 的加速比. 这表明，本文设计的架构在提高计算性能方面具有显著优势，尤其在大规模数据计算时展现了显著的加速能力.

为了验证数据流众核架构在加速快速傅里叶变换(FFT)蝶式运算方面的可扩展性，本文分别对优化读写模式和优化数据流在加速 FFT 计算的有效性进行了验证. 由于 FFT 与 NTT 的蝶式计算部分在运算结构上具有相似性，两者的数据流和计算模式也完全一致. 因此，针对 NTT 蝶式计算模式的加速方法同样适用于 FFT 的加速. 图 11 展示了在数据流众核架构上运行 FFT 蝶式运算时的加速比. 优化读写方式的方案在加速 FFT 计算中平均加速比为 1.34；而优化

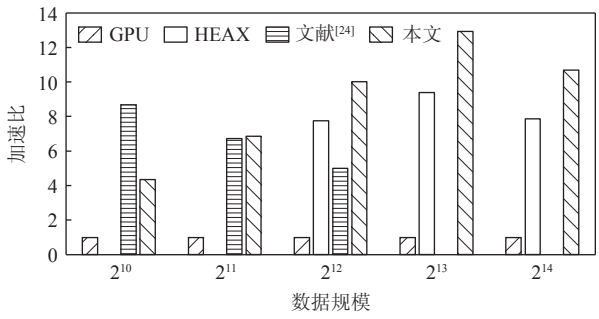


Fig. 10 Comparison chart of the time for calculating

图 10 计算时间对比图

数据流方案的平均加速比为 3.39. 综合这 2 项优化, 整体加速比达到 4.54. 这表明, 数据流众核架构在优化 FFT 蝶式运算的过程中, 能够有效提高计算性能.

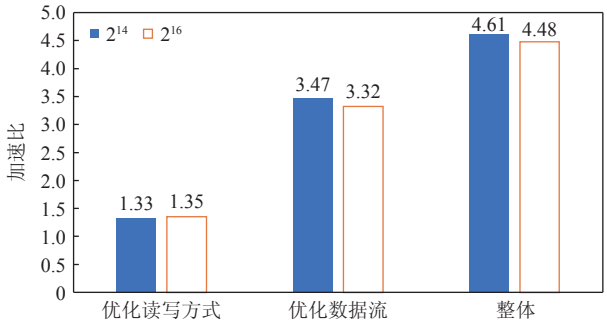


Fig. 11 Performance gain of FFT

图 11 FFT 性能收益

5.3 能效分析

图 12 展示了本文设计的数据流众核处理器与 GPU 之间的能效比. 结果表明, 本文数据流众核架构在性能上均优于 GPU, 并且每次计算的能耗显著低于 GPU. 具体而言, 本文设计的数据流众核处理器在计算过程中的平均计算资源利用率为 87.82%, 而 GPU 的峰值计算资源利用率仅为 18%~28%, 平均计算资源利用率为 4%~7%. 数据流处理器在能效方面稳定优于 GPU, 并且随着数据规模的扩大, GPU 的功耗会逐渐增加, 导致能效优化效果愈加显著, 最高可达到 10 倍以上. 整体来看, 数据流众核处理器的平均能效优化效果为 8.53 倍. 这表明, 本文设计的数据流众核处理器不仅具有更好的计算性能和更稳定的运行表现, 而且随着计算数据规模的增大, 其优势也更加突出.

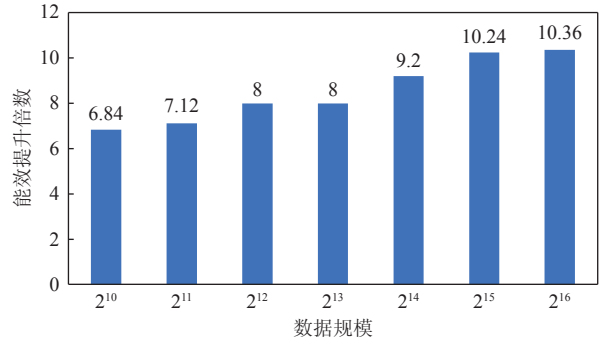


Fig. 12 Analysis diagram of energy efficiency optimization

图 12 能效优化分析图

5.4 功耗与面积分析

本文研究使用 Synopsys 工具和 Verilog 语言实现了 RVFHE 扩展单元: 模加/减单元和模乘单元. 通过使用 Synopsys Design Compiler 对模加/减和模乘单元进行综合, 本文选用了与 F1 相同的 12 nm 工艺, 采用 TT 工艺角, 电压设置为 0.8 V, 温度设定为标准运行状态下的 85 °C, 时钟频率为 1 GHz. 在综合后, 获得了计算单元的面积和能耗数据. 在每个 NTT 计算核中, 扩展了 32 套模加/减器和模乘器, 能够并行计算 32 组模加、模减和模乘运算, 从而显著提高了运算效率和并行度.

表 4 展示了模加/减单元和模乘单元各部分的面积. 模加/减单元的组合逻辑面积为 4 646.52 nm^2, 占比为 78.53%; 缓冲器和反相器面积为 356.66 nm^2; 非组合逻辑面积为 1 270.70 nm^2, 占比为 21.47%; 总面积为 5 917.23 nm^2. 模乘单元的组合逻辑面积为 117 720.76 nm^2, 占比为 95.58%; 缓冲器和反相器面积为 13 063.68 nm^2; 非组合逻辑面积为 5 353.21 nm^2, 占比为 4.42%; 总面积为 123 160.23 nm^2.

Table 4 Areas of Modulo Adder/Subtracter and Modulo Multiplier

表 4 模加/减器和模乘器面积

单元名称	组合逻辑面积/nm^2	缓冲器和反相器面积/nm^2	非组合逻辑面积/nm^2	总面积/nm^2
模加/减	4 646.52	356.66	1 270.70	5 917.23
模乘	117 720.76	13 063.68	5 353.21	123 160.23

表 5 展示了模加/减单元、模乘单元各部分运行时的功耗. 模加/减单元中, 寄存器的功耗占比为 42.7%, 组合逻辑的功耗占比为 57.3%. 模乘单元中, 时钟网络的功耗占比为 0.05%, 寄存器的功耗占比为 4.27%, 组合逻辑的功耗占比为 95.68%.

Table 5 Energy Consumption of Modulo Adder/Subtracter and Modulo Multiplier

表 5 模加/减器和模乘器能耗

单元名称	部件名称	短路功耗/mW	翻转功耗/mW	漏电功耗/mW	总功耗/mW	占比/%
模加/减	寄存器	0.032	0.120	0.001	0.154	42.7
	组合逻辑	0.080	0.092 9	0.034	0.206	57.3
	总功耗	0.113	0.213	0.035	0.360	100
模乘	时钟网络	0.002	0.001	0.001	0.003	0.05
	寄存器	0.148	0.147	0.022	0.317	4.27
	组合逻辑	2.859	2.826	1.424	7.109	95.68
	总功耗	3.009	2.974	1.447	7.430	100

表 6 展示了本文扩展指令和部件所带来的面积和功耗开销. 在 12 nm 工艺下, 扩展部件的面积为 6.29 mm^2, 且在 1 GHz 时钟频率下的最大功耗为 1 486 mW. 计算阵列在加速器中所占面积和功耗的比

例最大，分别占 57.71% 的面积和 51.56% 的功耗. 在每个计算单元中，执行引擎（包括计算单元、控制单元和指令与数据存储）所占比例最大.

Table 6 Area and Power Overhead of RVFHE Extended Part

表 6 RVFHE 扩展部分的面积与功耗

组成部分		面积/mm^2（占比）	功耗/mW（占比）
RVFHE 扩展	计算单元	0.125(54.97%)	21.92(45.73%)
	控制单元	0.033(14.60%)	2.69(5.62%)
	指令存储	0.015(6.62%)	1.73(3.60%)
	数据存储	0.054(23.84%)	21.59(45.05%)
	总和	0.227	47.93
阵列扩展总和		3.63(57.71%)	766(51.56%)
片上网络		1.13(17.92%)	194(13.07%)
数据缓存		1.10(17.56%)	400(26.94%)
配置缓存		0.16(2.51%)	82(5.50%)
DMA		0.27(4.30%)	44(2.93%)
总和		6.29	1 486

6 总　　结

本文提出了一种基于数据流架构的 NTT 蝶式计算加速方案，针对 NTT 蝶式计算中的核心模运算、分治蝶变和数据依赖等挑战，提出了相应的优化策略. 通过设计 RVFHE 指令集，成功加速了核心模运算. 引入数据重洗和蝶式地址生成方法，在数据预处理阶段提高了计算并行性. 提出的基于数据流的众核架构有效地将数据依赖作为计算触发机制，从而降低了计算延迟. 实验结果表明，与 GPU 相比，本文设计的加速架构在各数据规模下实现了平均 8.96 倍的加速效果；与现有最先进的 NTT 加速器相比，取得了 1.37 倍的加速提升. 此外，在能效方面，本文架构的能效（性能功耗比）优化效果达到了 GPU 的 8.53 倍以上.

未来的研究可以从 3 个方向进行拓展. 首先，硬件优化与集成方面，未来可以进一步优化数据流众核架构的硬件设计，探索更低功耗的硬件实现，以进一步提高计算性能和降低能耗. 其次，算法与架构的协同优化也是未来研究的重要方向，通过结合 NTT 算法的进一步优化和硬件架构特性，发展自适应算法，以提升计算性能和系统的灵活性，尤其是在不同规模数据集的计算中，能够根据实际需求动态调整资源分配. 最后，随着计算平台的多样化，异构计算将成为未来的趋势，未来的工作可以通过与 FPGA、专用加速器等硬件平台结合，进一步提升 NTT 计算的加速效果，扩大其应用领域.

作者贡献声明：石泓博提出了本文核心架构设计和实验验证并撰写论文；范志华、李文明负责架构设计和论文修改；张志远、穆宇栋负责实验和论文撰写；叶笑春、安学军负责论文总体的设计和讨论.

参　考　文　献

[1] Gentry C. A fully homomorphic encryption scheme[D]. Palo Alto, CA: Stanford University, 2009

[2] Feldmann A, Samardzic N, Krastev A, et al. An architecture to accelerate computation on encrypted data[J]. IEEE Micro, 2022, 42(4): 59−68

[3] Gentry C. Fully homomorphic encryption using ideal lattices[C]//Proc of Symp on the Theory of Computing. New York: ACM, 2009: 169−178

[4] Smart N, Vercauteren F. Fully homomorphic encryption with relatively small key and ciphertext sizes[C]//Proc of Public Key Cryptography–PKC 2010. Berlin: Springer, 2010: 420−443

[5] Brakerski Z, Gentry C, Vaikuntanathan V. (Leveled) Fully homomorphic encryption without bootstrapping[J]. ACM Transactions on Computation Theory-Special Issue on Innovations in Theoretical Computer Science 2012- Part II, 2014, 6(3): 1−36

[6] Bos J, Lauter K, Loftus J, et al. Improved security for a ring-based fully homomorphic encryption scheme[C]//Proc of Cryptography and Coding. IMACC 2013. Berlin: Springer, 2013: 45−64

[7] Brakerski Z. Fully homomorphic encryption without modulus switching from classical GapSVP[C]//Advances in Cryptology–CRYPTO 2012. Berlin: Springer, 2012: 868−886

[8] Gentry C, Halevi S. Implementing Gentry's fully-homomorphic encryption scheme[C]//Advances in Cryptology–EUROCRYPT 2011. EUROCRYPT 2011. Berlin: Springer, 2011: 129−148

[9] Erlingsson L, Pihur V, Korolova A. Rappor: Randomized aggregatable privacy-preserving ordinal response[C]//Proc of ACM Conf on Computer and Communications Security (CCS). New York: ACM, 2014: 1054-1067

[10] Gentry C. Computing arbitrary functions of encrypted data[J]. Communications of ACM, 2010, 53(3): 97−105

[11] Gentry C, Halevi S, Smart N. Fully homomorphic encryption with polylog overhead[C]//Proc of Annual Int Conf on the Theory and Applications of Cryptographic Techniques. Berlin: Springer, 2012: 465–482

[12] Gentry C, Sahai A, Waters B. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based[C]//Advances in Cryptology–CRYPTO 2013. Berlin: Springer, 2013: 75−92

[13] Cheon J, Kim A, Kim M, et al. Homomorphic encryption for

arithmetic of approximate numbers[C]//Advances in Cryptology–ASIACRYPT 2017. Berlin: Springer, 2017: 409−437

[14] Fan Shengyu, Wang Zhiwei, Xu Weizhi, et al. TensorFHE: Achieving practical computation on encrypted data using GPGPU[C]//Proc of 2023 IEEE Int Symp on High-Performance Computer Architecture (HPCA). Piscataway, NJ: IEEE, 2023: 922−934

[15] Akleylek S, Özgur D, Zaliha Y. On the efficiency of polynomial multiplication for lattice-based cryptography on GPUs using CUDA[C]//Proc of Int Conf on Cryptography and Information Security in the Balkans. Berlin: Springer, 2015: 155−168

[16] Badawi A, Veeravalli B, Mun C, et al. High performance FV somewhat homomorphic encryption on GPUs: An implementation using CUDA[J]. Transactions on Cryptographic Hardware and Embedded Systems, 2018(2): 70−95

[17] Dai W, Sunar B. cuHE: A homomorphic encryption accelerator library[C]//Proc of Cryptography and Information Security in the Balkans (BalkanCryptSec 2015). Berlin: Springer, 2015: 169−186

[18] Riazi M, Laine K, Pelton B, et al. HEAX: An architecture for computing on encrypted data[C]//Proc of the 25th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2020: 1295−1309

[19] Roy S, Turan F, Jarvinen K, et al. FPGA-based high-performance parallel architecture for homomorphic computing on encrypted data[C]//Proc of 2019 IEEE Int Symp on High Performance Computer Architecture (HPCA), Piscataway, NJ: IEEE, 2019: 387−398

[20] Pöppelmann T, Naehrig M, Putnam A, et al. Accelerating homomorphic evaluation on reconfigurable hardware[C]//Proc of Cryptographic Hardware and Embedded Systems (CHES 2015). Berlin: Springer, 2015: 143−163

[21] Feldmann A, Samardzic N, Krastev A. F1: A fast and programmable accelerator for fully homomorphic encryption[C]//Proc of the 54th Annual IEEE/ACM Int Symp on Microarchitecture (MICRO 2021). New York: ACM, 2021: 238−252

[22] Samardzic N, Feldmann A, Krastev A. CraterLake: A hardware accelerator for efficient unbounded computation on encrypted data[C]//Proc of Int Symp on Computer Architecture. New York : ACM, 2022: 173−187

[23] Karabulut E, Aysu A. RANTT: A RISC-V architecture extension for the number theoretic transform[C]//Proc of the 30th Int Conf on Field-Programmable Logic and Applications (FPL). New York : ACM, 2020: 26−32

[24] Paludo R, Sousa L. NTT architecture for a Linux-ready RISC-V fully-homomorphic encryption accelerator[J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2022, 69(7): 2669−2682

[25] Lu Zhaojun, Yu Weizong, Xu Peng, et al. An NTT/INTT accelerator with ultra-high throughput and area efficiency for FHE[C]//Proc of the 61st ACM/IEEE Design Automation Conf (DAC'24). Association for Computing Machinery. New York: ACM, 2024: 1−6

[26] Dennis J B. First version of a dataflow procedure language[C]//Proc of Programming Symp. Berlin: Springer, 1974, 19: 362−376

[27] Dijk M, Gentry C, Halevi S, et al. Fully homomorphic encryption over the integers[C]//Proc of Int Conf on Theory & Applications of Cryptographic Techniques. Berlin: Springer, 2010: 24−43

[28] Gentry C, Halevi S. Implementing Gentry's fully-homomorphic encryption scheme[C]//Advances in Cryptology–EUROCRYPT 2011. Berlin: Springer, 2011: 129−148

[29] Krendelev S, Tormasov A. Method for protecting data used in cloud computing with homomorphic encryption: US10116437B1 [P]. 2018-10-30

[30] Zhang Y, Dai W, Jiang X, et al. FORESEE: Fully outsourced secure genome study based on homomorphic encryption[J/OL]. BMC Medical Informatics & Decision Making, 2015[2025-03-01]. http://doi.org/10.1186/1472-6947-15-s5-s5

[31] Lagendijk R, Erkin Z, Barni M, Encrypted signal processing for privacy protection: Conveying the utility of homomorphic encryption and multiparty computation[J]. IEEE Signal Processing Magazine, 2013, 30(1): 82−105

[32] Gentry C, Halevi S, Smart N P. Homomorphic evaluation of the AES circuit[C]//Advances in Cryptology–CRYPTO 2012. Berlin: Springer, 2012: 850−867

[33] Asanović K, Avižienis R, Bachrach J, et al. The rocket chip generator[R]. Berkeley: University of California, 2016: 1−11

[34] Ye Xiaochun, Fan Dongrui, Sun Ninghui, et al. SimICT: A fast and flexible framework for performance and power evaluation of large-scale architecture[C]//Proc of the Int Symp on Low Power Electronics and Design (ISLPED). New York: ACM, 2013: 273−278

[35] Fan Zhihua, Li Wenming, Tang Shengzhong, et al. Improving utilization of dataflow architectures through software and hardware co-design[C]//Proc of Parallel Processing (Euro-Par 2023). Berlin: Springer, 2023: 245−259

Shi Hongbo, born in 2002. Bachelor. His main research interests include dataflow architecture and high-performance computing.

石泓博, 2002 年生. 学士. 主要研究方向为数据流架构、高性能计算.

Fan Zhihua, born in 1996. PhD, assistant professor. His main research interests include dataflow architecture, programming model, and reconfigurable architecture.

范志华, 1996 年生. 博士. 助理研究员. 主要研究方向为数据流体系结构、编程模型、可重构体系结构.

Li Wenming, born in 1988. PhD, associate professor. His main research interests include high-throughput processor architecture, dataflow architecture, and software simulation.

李文明, 1988 年生. 博士. 副研究员. 主要研究方向为高吞吐量处理器体系结构、数据流体系结构、软件仿真.

Zhang Zhiyuan, born in 2000. PhD candidate. His main research interests include high-performance computing, dataflow, and reconfigurable architecture.

张志远, 2000 年生. 博士研究生. 主要研究方向为高性能计算、数据流、可重构架构.

Mu Yudong, born in 2001. PhD candidate. His main research interests include dataflow architecture, dataflow graph mapping, and reconfigurable architecture.

穆宇栋, 2001 年生. 博士研究生. 主要研究方向为数据流架构、数据流图映射、可重构架构.

Ye Xiaochun, born in 1983. PhD, professor. His main research interests include high-performance computer architecture and software simulation.

叶笑春, 1983 年生. 博士, 研究员. 主要研究方向为高性能计算机体系结构、软件仿真.

An Xuejun, born in 1966. PhD, professor. His main research interests include programming model and processor architecture.

安学军, 1966 年生. 博士. 研究员. 主要研究方向为编程模型、处理器体系结构.