本科毕业论文

论文题目（中文） 基于 GPU 的 CKKS 算法优化

论文题目（英文） Gpu-based Homomorphic Encryption Computing System Optimization

学生姓名 何东杰

指导教师 赵志立

学　　院 信息科学与工程学院

专　　业 数据科学与大数据技术

年　　级 2019 级

兰州大学教务处

诚信责任书

本人郑重声明：本人所呈交的毕业论文（设计），是在导师的指导下独立进行研究所取得的成果。毕业论文（设计）中凡引用他人已经发表或未发表的成果、数据、观点等，均已明确注明出处。除文中已经注明引用的内容外，不包含任何其他个人或集体已经发表或在网上发表的论文。

本声明的法律责任由本人承担。

论文作者签名：_____ 日 期：_____

关于毕业论文（设计）使用授权的声明

本人在导师指导下所完成的论文及相关的职务作品，知识产权归属兰州大学。本人完全了解兰州大学有关保存、使用毕业论文（设计）的规定，同意学校保存或向国家有关部门或机构送交论文的纸质版和电子版，允许论文被查阅和借阅；本人授权兰州大学可以将本毕业论文（设计）的全部或部分内容编入有关数据库进行检索，可以采用任何复制手段保存和汇编本毕业论文（设计）。本人离校后发表、使用毕业论文（设计）或与该毕业论文（设计）直接相关的学术论文或成果时，第一署名单位仍然为兰州大学。

本毕业论文（设计）研究内容：

☑可以公开

□ 不宜公开，已在学位办公室办理保密申请，解密后适用本授权书。

（请在以上选项内选择其中一项打“√”）

论文作者签名：_____ 导师签名：_____

日 期：_____ 日 期：_____

基于 GPU 的 CKKS 算法优化

中文摘要

CKKS 算法是一种基于多项式环的同态加密算法，其无需解密就可对加密数据直接计算，在公有云隐私数据的计算中有着广泛的应用。然而，CKKS 算法在加密过程中会导致原本的数据位数以及数据量增加，存在运行效率较低的问题。因此，本文的主要目标在于寻找 CKKS 算法的性能瓶颈，同时基于 GPU 平台对其性能瓶颈进行优化，以提升其运行效率，主要工作有：

（1）CKKS 算法性能瓶颈分析。在 CPU 平台对 CKKS 算法各个算子的运行时间进行测算，发现同态密文乘法是 CKKS 算法的主要性能瓶颈。（2）同态密文乘法算法性能瓶颈分析。本文对同态密文乘法算法的运算过程进行了进一步的分解，通过运行时间对比发现多项式模乘为其核心性能瓶颈。（3）CKKS 算法性能优化。为了提高算法的效率，本文基于 GPU 平台对同态密文乘法进行了优化，采用常量内存和异步传输等机制，同时利用了 GPU 高并行性的特点提升算法的并行度，最终得到了更高效的算法。

本文通过逻辑回归模型验证了优化方案在真实场景下的可行性，并与未经过优化的算法进行了比较。实验结果显示，优化后的核心性能瓶颈即多项式模乘相比于基于 CPU 平台实现的初始算法实现了 10 倍以上的加速比，优化过后的同态密文乘法则实现了 2.5 倍的加速比。因此，本研究在 CKKS 算法的性能瓶颈探究以及优化方面做出了一定的贡献，为隐私数据的保护提供了一种相对高效且可行的解决方案。

关键词： CKKS 算法，同态密文乘法，GPU 优化，异步传输

Gpu-based Homomorphic Encryption Computing System Optimization

Abstract

The CKKS algorithm is an encryption algorithm based on polynomial rings that can achieve fully homomorphic encryption. It can perform calculations while encrypting data without decrypting ciphertexts to obtain the calculation results. This algorithm has a wide range of applications in scenarios where privacy data protection is required, such as cloud computing, finance, and healthcare. It can protect data while allowing for calculations, achieving the dual purpose of privacy protection and data processing. However, the encryption process can lead to an increase in the number and size of the original data, resulting in low efficiency. By testing the algorithm process on the CPU platform, we found that homomorphic ciphertext multiplication is the main performance bottleneck of the CKKS algorithm. We further decomposed the homomorphic ciphertext multiplication and found that the polynomial modular multiplication is the core performance bottleneck. To improve the efficiency of the algorithm, we optimized the homomorphic ciphertext multiplication based on the GPU platform, using mechanisms such as constant memory and asynchronous transmission and leveraging the high parallelism of the GPU. We finally obtained a more efficient algorithm. We verified the feasibility of the optimization scheme in real scenarios through a logistic regression model and compared it with the unoptimized algorithm. For the core performance bottleneck, our optimized algorithm achieved an acceleration ratio of more than 10 times. Therefore, this study has made contributions to the exploration and optimization of the performance bottleneck of the CKKS algorithm, providing a relatively efficient and feasible solution for the protection of privacy data.abcdQR

Keywords: CKKS; Homomorphic Ciphertext multiplication; GPU optimization; Asynchronous transmission

目　录

图 目 录

表 目 录

第一章 绪 论

1.1 研究背景及意义

在信息化时代，数据安全问题日益重要。为了保障数据隐私，同态加密技术应运而生。同态加密技术可以实现在不破坏数据加密状态的前提下进行计算，为数据隐私保护提供了有力的保障，其比较适合公有云领域针对隐私数据的 AI 模型的训练与推理。同态加密计算系统在现实生活中应用广泛，如金融行业、医疗行业、电子商务等。例如，在金融行业中，同态加密技术可以用于保护客户隐私，如同态加密技术可以对客户的信用评级进行加密，保护客户的个人信息不被泄露。在医疗行业中，同态加密技术可以用于保护患者隐私，如同态加密技术可以对患者的病历信息进行加密，保护患者的个人隐私。在电子商务领域，同态加密技术可以用于保护用户隐私，在不破坏用户隐私的前提下，对用户行为数据进行分析，提高电子商务平台的服务质量。

同态加密技术的发展历程可以追溯到 20 世纪 70 年代，最早的同态加密算法是 RSA 算法 [1]。然而，RSA 算法的同态加密性能较差，只能支持同态乘法，很难适应大规模数据场景的需求。20 世纪 90 年代末期，Paillier 又提出了可以实现同态加法的方案 [2]，但是也存在加密性能差、仅支持同态加法的问题，导致无法大规模应用。2009 年 Gentry 提出了首个基于理论的同态加密方案，打开了同态加密理论研究的新篇章。初期的同态加密方案存在误差累计问题，因此对于进行同态运算的次数存在限制，Gentry 引入了自举操作解除了这一限制, 该同态加密方案被称为完全同态加密 [3]。Gentry 提出的同态加密方案主要有两种，一种是基于理想格的同态加密方案 [4]，另一种是基于环同态的同态加密方案 [3]。其中，基于环同态的同态加密方案更加实用，被广泛应用于实际场景中。

CKKS(Cheon-Kim-Kim-Song) 算法 [5] 是一种基于环同态的同态加密算法，具有优秀的性能和可扩展性。CKKS 算法可以在不破坏数据隐私的前提下，支持多项式计算和向量计算，可以用于解决大规模数据场景下的同态加密计算问题。其中自举操作对同态加密算法进行优化，能够提升模数，避免因在密文乘法中的模数降低从而产生计算次数的限制，理论上能够对加密数据进行无限次的运算，但也消耗了大量的计算资源，同时其并没有消除误差累积的问题，在实际应用场景下，仍然具有运算次数的限制，不包含自举的 CKKS 在对模量进行合理规划的前提下，足以适应大部分场景。因此本文实现的 CKKS 算法不包含自举操作。

虽然同态加密技术在数据隐私保护方面具有重要意义，但是在大规模数据场景下，同态加密计算系统往往会出现性能瓶颈，导致系统效率低下。因此，如何提高同态加密计算系统的效率，使其能够适应大规模数据场景下的需求，成为了研究的热点问题。在不考虑自举操作的情况下，本文通过实验证明，密文乘法为主要的性能瓶颈。同时，GPU 拥有高并行性和强大的浮点计算能力，在图形处理和科学计算等领域得到了广泛应用。因此，一些

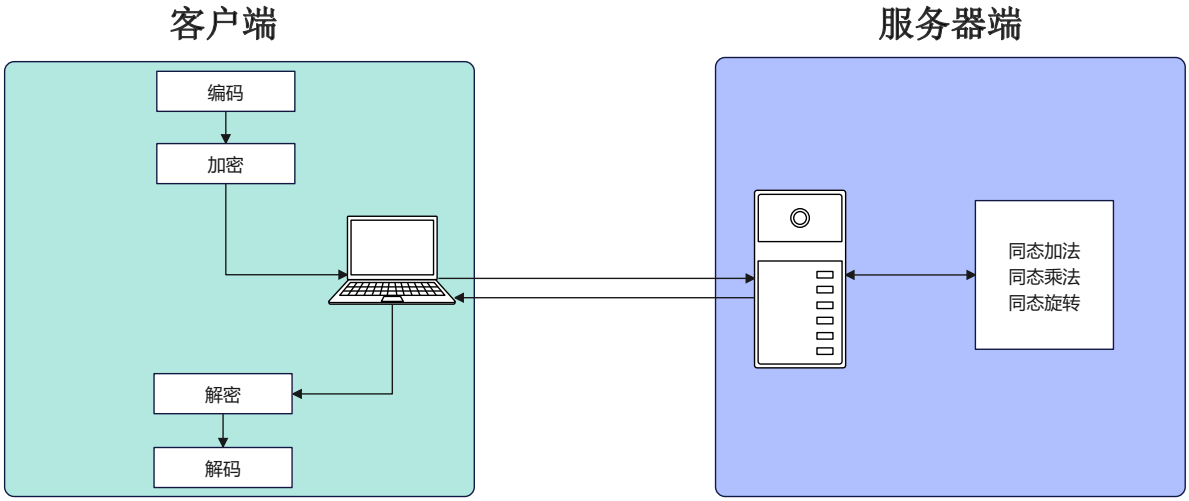


图 1.1 同态加密过程示意图

研究者开始尝试将 GPU 应用于同态加密技术的优化中，以提高同态加密技术的计算效率。因此基于 GPU 平台对同态加密计算系统进行优化具有较为重要的意义。如图1.1所示，同态加密主要包含编码、解码、加密、解密、同态加法，同态乘法，同态旋转等操作。编码、解码、加密、解密，均运行在客户端，因此无需对其进行基于 GPU 的优化。同态加法，同态乘法，同态旋转则运行在服务器端，需要对其进行优化。本文通过实验验证，发现密文乘法为主要的性能瓶颈，但是目前没有完整的基于 GPU 实现的完整的密文乘法的开源解决方案，因此本文对于同态加密乘法进行了 GPU 的实现，并应用于逻辑回归，对其性能进行验证。

1.2 研究现状

相比于其他的隐私保护技术，完全同态加密技术具有更广泛的应用场景，可以实现不同类型的应用程序，如机器学习、信息检索和基因组分析。然而较高的性能开销阻碍了完全同态加密的大规模应用。当前即使使用高度优化的完全同态加密库或者高端 CPU，计算仍然比基于未解密数据的等效应用程序多花费 10^4 到 10^5 倍的时间[6]。

因此，近年来提出了不少基于不同硬件的优化解决方案。100x[7] 提出了第一个基于 GPU 的 CKKS 实现方案，之后 TensorFHE[6] 则基于 GPU 中的 tensor core 对完全同态加密中的 NTT 进行了实现，使得性能进一步得到了优化。HEAX[8] 则基于 FPGA 实现了对 CKKS 的加速，其优化了 NTT 操作以及模量操作，然而由于芯片资源限制其并不支持自举，之后 Poseidon[9] 则基于 FPGA 采用了基于 radix 的优化方法优化了 NTT 操作，同时实现了自举操作。同时还有大量基于 ASIC 的 CKKS 实现，例如 F1+[10]，CraterLake[11]，BTS[12] 以及 ARK[13] 它们均实现了较为显著的性能提升，但是无一例外，它们的实现均依赖于大量且昂贵的芯片上的缓冲区或者寄存器，因此很难大面积普及。例如, F1+[10] 以及 CraterLake[11] 需要 256MB 芯片缓冲区, 同时 BTS[12] 以及 ARK[13] 则需要 512MB 寄存器

空间。其中 GPU 具有最好的通用性，以及具有高带宽、高并行度的特性，能够较好的应用于多种场景，因此基于 GPU 进行优化具有很高的价值。

1.3 本文的主要工作

在隐私保护领域，完全同态加密是应用场景最广的解决方案，CKKS 算法是近年来完全同态加密方案中主流使用的算法。本文主旨在探究 CKKS 算法的性能瓶颈，同时基于 GPU 平台对 CKKS 进行优化。主要工作如下：

1. 基于 CPU 平台从 CKKS 算法的整体流程的角度，测算各个算子的运行时间，分析 CKKS 算法的性能瓶颈，发现同态密文乘法为主要的性能瓶颈。之后对于同态密文乘法中的算子进行进一步的拆解，测算同态密文乘法中各个算子的运行时间，确定多项式模乘为主要的性能瓶颈，对其进行深度优化。
2. 将同态密文乘法中的多项式模乘基于 GPU 平台进行优化，其中 NTT 与 INTT 为算法复杂度最高的操作，采用 GPU 中的常量内存以保存 NTT 以及 INTT 中重用次数较多的数据，以提升 NTT 以及 INTT 的 DMA 效率。同时采用锁页内存分配 CPU 与 GPU 之间需要传输的数据并且将 CPU 与 GPU 之间的数据数据传输以及拷贝过程设计为异步传输方式，在一定程度上隐藏通信时间，最终将同态密文乘法中的各个算子进行整合，在 GPU 中实现完整的同态密文乘法。
3. 对优化后的 CKKS 算法的可靠性进行验证，并将其应用于逻辑回归模型验证优化方案在真实场景下是否可行，之后将基于 GPU 优化过后的密文乘法和没有优化过以后的密文乘法的运行效率进行比较，最后基于逻辑回归模型比较优化前后的 CKKS 算法的整体运行效率。

1.4 本文的组织结构

本文旨在基于半同态加密的场景下，探究 CKKS 算法的性能瓶颈，并基于 GPU 对其性能瓶颈进行优化。本文通过实验验证发现密文乘法为主要的性能瓶颈，并对其基于 GPU 进行实现，之后将其整合进 CKKS 算法，通过逻辑回归模型验证其可靠性以及执行效率的提升程度。

第一章 绪论 本章引出本文的主要研究内容，介绍 CKKS 算法的应用场景以及研究意义，并介绍了基于不同硬件的优化的研究成果，并由此引出了基于 GPU 对其进行优化的意义。

第二章 相关理论与技术 本章介绍了 CKKS 算法的相关概念及各种同态运算。主要包括多项式环的相关概念、CKKS 中多项式计算的相关理论，以及编码、解码、加密、解密、同态加法、同态乘法、同态旋转等相关运算。

第三章 CKKS 算法性能瓶颈分析 本章首先对完整的同态加密运算过程进行分析，确

认同态密文乘法为主要的性能瓶颈，之后对于同态密文乘法进行进一步拆解，确认其中的多项式乘法为核心性能瓶颈。

第四章 基于 GPU 的同态密文乘法实现 本章详细描述了基于 GPU 的同态密文乘法的优化策略，描述了同态密文乘法中不同算子的 CPU 和 GPU 的划分方案，并详细描述了异步传输方案。

第五章 实验结果与分析 本章首先对实验环境以及参数设置原则进行描述，之后对于优化过后的同态密文乘法的可靠性进行验证，最终对优化前后的性能进行分析，测算其加速比。

第六章 总结与展望 本章是对本文主要工作的概括总结。总结了基于 GPU 实现的同态乘法相比于 CPU 实现存在哪些性能提升，并通过逻辑回归模型对全过程的性能提升进行进一步总结，同时总结文章的不足之处并对未来工作出展望。

第二章 相关理论与技术

2.1 同态加密的相关概念

以 2 为底的对数，后文将简单使用 $\log(\cdot)$ 表示。我们采用 $\langle\cdot,\cdot\rangle$ 两个向量的点积，对于实数 r, 采用 $\lfloor r \rceil$ 表示距离 r 最接近的整数,$[Z]_q$，采用 $x \leftarrow D$ 表示 x 从分布 D 中采样。对于整数 q, 我们定义 $\mathbb{Z} \cap \left(-\frac{q}{2}, \frac{q}{2}\right]$ 作为 Z_q 的完全剩余系的代表，并使用 $[z]_q$ 表示 z 对 q 取模后的余数。

我们采用 $\Phi_M(X) = X^N + 1$ 表示第 M 个分圆多项式（$M = 2N$）。$R = \mathbb{Z}[X]/(\Phi_M(X))$。我们将 R/qR 定义为 R 模 q 的剩余环。集合 $P = R[X]/(\Phi_M(X))$ 中的任意元素将表示为多项式 $a(X) = \sum_{j=0}^{N-1} a_j X^j$，其幂指数严格小于 N，并定义其系数向量为 $\mathbf{a} = (a_0, ..., a_{N-1}) \in R^N$ 。同时，我们采用系数向量 $\mathbf{a}$ 的相关范数定义 $\|a\|_1$ 以及 $\|a\|_\infty$。

对于实数 $\sigma > 0$，$DG(\sigma^2)$ 表示在 Z^N 上的分布，其从方差为 σ^2 的离散高斯分布独立采样构成。对于正整数 h，$HWT(h)$ 表示在 $\{\pm 1\}^N$ 上的二进制均匀分布，其汉明权重恰好为 h 。对于实数 $0 \le \rho \le 1$，分布 $ZO(\rho)$ 表示从向量 $\{0, \pm 1\}$ 中提取元素，其值等于 -1 和 $+1$ 的概率为 $\rho/2$，其值等于零的概率是 $1 - \rho$。

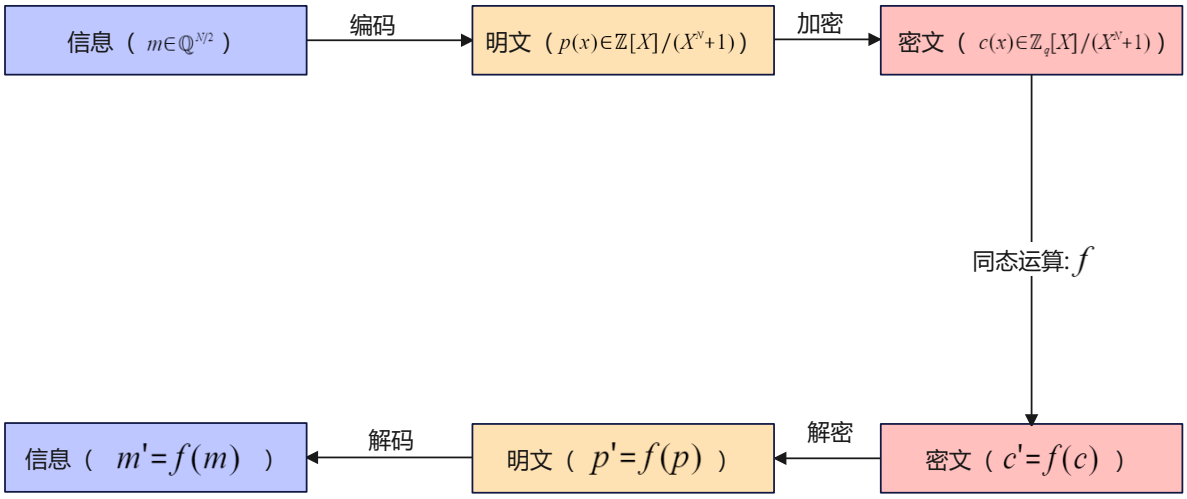


图 2.1 CKKS 算法的流程

CKKS 算法流程如图2.1所示。其首先需要将位于 Q 的向量表示的信息经过编码生成位于 Z 的明文，之后生成私钥 sk 和公钥 pk，之后通过公钥对明文进行加密获得密文，然后将获得的密文进行同态运算得到结果，再用私钥对其进行解密获得明文并将明文解码获得需要的信息。其涉及的相关函数如下所示：

- 编码以及解码：由于输入为 $z \in \mathbb{Q}^{N/2}$，但是 CKKS 算法则是基于整数多项式环进行，因此需要对其进行编码操作，解码为编码的逆过程，因此不对其进行重复叙述，编码的详细过程如下：

  - 扩充到 N 维向量：$\pi^{-1}(z) = (z_1, z_3, ..., z_{N-1}, z_{N+1}, ..., z_{2N-1}), z_{N+i} = \bar{z}_{N-i}, i = 1, 2, ...$，取共轭可以消去虚数部分，并保留虚数部分有效信息。
  - 乘以 Δ 以保证精度：由于 CKKS 密文和明文都是整数类型，此时乘的 Δ 相当于保留的小数位数。$\Delta\pi^{-1}(z) = (\Delta z1, \Delta z3, ..., \Delta z_{2N-1})$
  - 投影到 $\sigma(R) : \lfloor \Delta\pi^{-1}(z) \rceil_{\sigma(R)} = (z_1, z_2, ..., z_{N-1})$，即得到 N 组整数域的坐标，$z_i = \frac{<z, b_i>}{||b_i||^2}$，同时因为需要得到整数环，需要对 z 进行近似。
  - 编码到实数环：$\sigma^{-1}(\lfloor \Delta\pi^{-1}(z) \rceil_{\sigma(R)}) = (z_1, z_2, ..., z_{N-1}) \in R$

- 密钥生成函数分为初始密钥生成函数 $(keyGen(1^\lambda))$ 以及转换密钥生成函数 $(KSGen_{sk}(s'))$，其中 λ 表示安全级数。对于一个底数为 p 和一个整数 L，令 $q_\ell = p^\ell$，其中 $\ell = 1, ..., L$。给定安全参数 λ，选择 M 为 2 的幂指数，整数 h，整数 P，以及一个实数 $\sigma > 0$，用于实现 λ 比特安全级别的 RLWE 问题。具体细节如下所示：
  - $keyGen(1^\lambda)$：选取 $s \leftarrow HWT(h), a \leftarrow U(R_{qL})$ 和 $e \leftarrow DG(\sigma^2)$。将私钥设置为 $sk \leftarrow (1, s)$，将公钥设置为 $pk \leftarrow (b, a) \in R_{qL}^2$，其中 $b \leftarrow -as + e \pmod{q_L}$
  - $KSGen_{sk}(s')$：对于 $s_0 \in R$，选取 $a_0 \leftarrow U(R_{P \cdot qL})$ 以及 $e \leftarrow DG(\sigma^2)$。输出转换密钥为 $swk \leftarrow (b', a') \in R_{P \cdot qL}^2$，其中 $b' \leftarrow -a's + e' + Ps' \pmod{P \cdot q_L}$。
- 加密 $(Enc_{pk}(m))$：对于 $m \in R$, 选取 $v \leftarrow ZO(0.5)$ 以及 $e_0, e_1 \leftarrow DG(\sigma^2)$。输出 $v \cdot pk + (m + e_0, e_1) \pmod{q_L}$
- 解密 $(Dec_{sk}(ct))$：对于 $c_t = (b, a) \in R_{q\ell}^2$, 输出 $m = b + a \cdot s \pmod{q_\ell}$。

2.2 CKKS 中的多项式计算

CKKS 中的密文表示为一组位于多项式环 $R_Q = \mathbb{Z}_Q[X]/(X^N + 1)$, 幂指数小于 N, 系数不大于 Q 的多项式，同态运算由许多多项式操作构成。其中 N 和 Q 都是很大的数，N 的取值范围一般是 2^{16} 到 2^{18},$\log Q$ 的取值一般在 3000，因此会带来较大的计算开销。例如，基于加密数据的同态乘法会比未加密数据慢数千倍 [14]。因此在 CKKS 算法中进行多项式的计算需要对其进行一些特殊的处理。

2.2.1 快速数论变换以及逆快速数论变换

多项式直接相乘的时间复杂度为 $o(N^2)$，当多项式系数较小时并不会带来过高的计算开销，但是由于 CKKS 中的密文多项式长度均在 2^{16} 到 2^{18}，其较高的时间复杂度就变成了不可忽视的问题。因此 CKKS 采用了快速数论变换（number-theoretic transform, NTT）和快速数论逆变换（inverse number-theoretic transform, INTT）来完成多项式乘法，其可以将时间复杂度由 $o(N^2)$ 降低到 $o(N \log N)$。

给定一个正整数 m，我们可以在模 m 意义下找到一个原根 g，满足 $\forall i \in [1, m-1], \exists j$ 使

得 $g^j \equiv i \pmod{m}$。我们通常会选择一个素数 p 作为模数 [15]。给定一个长度为 N 的整数序列 $A = (a_0, a_1, \ldots, a_{N-1})$，我们希望找到一个整数序列 $Y = (y_0, y_1, \ldots, y_{N-1})$，满足

$$y_k = A(\omega^k), \quad k = 0, 1, \ldots, N-1,$$

其中 ω 是模 p 意义下的 N 次单位根。这样，我们可以利用蝶形运算实现高效的 NTT 变换 [16]。

蝶形运算是一种高效实现离散线性变换的方法，类似于 FFT 中的蝶形运算。NTT 的蝶形运算可以在 $O(N \log N)$ 的时间复杂度内完成，具体步骤如下：

1. 将输入序列 A 的元素按照二进制逆序排列；
2. 对于每一个 $1 \le s \le \log_2 N$，执行以下步骤：
   (a) 将输入序列划分为长度为 2^s 的子序列；
   (b) 对于每个子序列，将其分为两部分，前半部分为 A_1，后半部分为 A_2；
   (c) 计算 A_1 与 A_2 的元素之间的蝶形运算，即

$$\begin{cases} A_1[j] \leftarrow A_1[j] + \omega^{N/2^s} A_2[j], \\ A_2[j] \leftarrow A_1[j] - \omega^{N/2^s} A_2[j], \end{cases}$$

   其中 $j = 0, 1, \ldots, 2^{s-1} - 1$。

经过以上步骤，我们可以得到 NTT 变换后的序列 Y。

为了从 NTT 变换后的序列 Y 恢复原序列 A，我们需要进行逆 NTT 变换。逆 NTT 变换的过程与正向 NTT 变换类似，但需要使用模 p 意义下的 ω^{-1}。具体步骤如下：

1. 将输入序列 Y 的元素按照二进制逆序排列；
2. 对于每一个 $1 \le s \le \log_2 N$，执行以下步骤：
   (a) 将输入序列划分为长度为 2^s 的子序列；
   (b) 对于每个子序列，将其分为两部分，前半部分为 Y_1，后半部分为 Y_2；
   (c) 计算 Y_1 与 Y_2 的元素之间的蝶形运算，即

$$\begin{cases} Y_1[j] \leftarrow Y_1[j] + \omega^{N/2^s} Y_2[j], \\ Y_2[j] \leftarrow Y_1[j] - \omega^{N/2^s} Y_2[j], \end{cases}$$

   其中 $j = 0, 1, \ldots, 2^{s-1} - 1$。
3. 对于每个元素 y_i，计算 $y_i \cdot N^{-1} \pmod{p}$，得到原序列 A 的元素。

2.2.2 余数系统

由于系数过大带来的巨大计算开销，基于 RLWE 实现的 CKKS 算法为解决这一问题，实现了余数系统（Residue Number System，RNS）。其使用中国剩余定理（Chinese Remainder Theorem, CRT）将每一个较大的质数转化为一组余数，每一个余数都是其对应的系数模一个不同的质数得到的，其可以从整数域推广至多项式域。

CRT 是数论中的一个重要定理，它可以解决同余方程组的问题。CRT 已经有着悠久的历史，最早可以追溯到公元 3 世纪的中国，它在密码学、编码理论和计算机科学等领域都有着广泛的应用 [17]。

给定一组两两互质的正整数 $n_1, n_2, \ldots, n_k$，以及一组多项式 $a_1(x), a_2(x), \ldots, a_k(x)$，则存在一个唯一的整数 x，满足以下同余方程组：

$$\begin{cases} f(x) \equiv a_1(x) \pmod{n_1}, \\ f(x) \equiv a_2(x) \pmod{n_2}, \\ \cdots \\ f(x) \equiv a_k(x) \pmod{n_k}. \end{cases}$$

多项式 $f(x)$ 可以表示为：

$$f(x) \equiv \sum_{i=1}^{k} a_i(x) M_i M_i^{-1} \pmod{N},$$

其中 $N = n_1 n_2 \cdots n_k$，$M_i = N/n_i$，M_i^{-1} 是模 n_i 意义下 M_i 的逆元。

求解中国剩余定理的同余方程组的算法如下：

1. 计算 $N = n_1 n_2 \cdots n_k$；
2. 对于每个 $i = 1, 2, \ldots, k$，计算 $M_i = N/n_i$；
3. 对于每个 $i = 1, 2, \ldots, k$，计算模 n_i 意义下 M_i 的逆元 M_i^{-1}；
4. 计算 $x = \sum_{i=1}^{k} a_i M_i M_i^{-1} \pmod{N}$。

2.3 CKKS 中的同态运算

同态运算主要包括同态加法，同态乘法以及同态旋转，其中同态乘法又包括密文和密文的乘法以及密文和明文的乘法, 密文相乘以后模量会扩张则需要进行重缩放操作。由于密文乘法以及旋转不是线性的运算会带来密钥的改变，因此需要使用密钥转换函数对其改变以后的密钥转换为运算前的密钥，以保证运算结果的准确性。其涉及相关操作的函数如下所示。

- 密文间的同态加法 $(Add(ct_1, ct_2))$: 对于输入 $ct1, ct2 \in R_{q_\ell}^2$, 输出 $ct_{add} \leftarrow ct_1 + ct_2 \mod q_\ell$
- 密文和明文间的同态乘法 $(Mult(ct, m))$：对于 $ct = (b, a)$ 以及信息 m, 输出 $c_{mult} \leftarrow (m \cdot b, m \cdot a)$

- 密文间的同态乘法 ($Mult_{evk}(ct_1, ct_2)$):对于 $ct_1 = (b_1, a_1), ct_2 = (b_2, a_2) \in R^2_{q\ell}$,令 $(d_0, d_1, d_2) = (b_1b_2, a_1b_2 + a_2b_1, a_1a_2) \pmod{q_\ell}$。输出 $ct_{mult} \leftarrow (d_0, d_1) + \lfloor P^{-1} \cdot d_2 \cdot evk \rceil \pmod{q_\ell}$
- 重缩放 ($RS_{\ell \to \ell'}$)：对于 $ct \in R^2_{q\ell}$ 的密文，输出 $ct' \leftarrow \lfloor P^{\ell'-\ell} \cdot ct \rceil \pmod{q_\ell}$。当 $\ell = \ell'$ 时，我们将省略下标 ($\ell \to \ell'$)。
- 密钥转换 ($KS_{swk}(ct)$): 输入密文 ct 以及转换密钥 swk, 输出 $ct = (b, 0) + \lfloor P^{-1} \cdot a \cdot swk \rceil \pmod{q_\ell}$.
- 密文的同态旋转 ($Rot(ct; r)$):$Rot(ct; r)$ 表示密文 ct 旋转 r 位，首先生成旋转密钥 $rk_r \leftarrow KSGen_{sk}(5^r(s))$, 并通过密钥转换函数获得密文 $KS_{rk_r}(5^r(ct))$

2.4 基于 CKKS 算法的逻辑回归实现

逻辑回归模型因为其原理简单、可解释性强以及可并行化的优点从而得到了广泛应用。其采用 sigmod 函数作为分类函数，sigmod 函数表示为：

$$\frac{1}{1+e^{-x}} \tag{2.1}$$

逻辑回归模型表示为：

$$p(y_i|x_i) = \frac{1}{1+e^{-y_i\omega^T x_i}} \tag{2.2}$$

然而由于 sigmod 函数不是一个线性函数，并且如果采用非线性函数对于密文进行映射，会破坏密文之间原本的同态关系。因此我们不能直接采用 sigmod 函数作为分类函数，而是采用泰特展开对其进行近似，其三阶泰勒展开如下所示：

$$sigmoid(x) = \frac{1}{2} + \frac{x}{4} - \frac{x^3}{48} + O(x^5) \tag{2.3}$$

本文采用三阶泰勒展开对其进行近似，采用梯度下降法对模型参数进行求解，并分别对其可靠性进行验证。

在逻辑回归中，我们需要实现对输入特征的线性变换即 **wx**，由于直接对密文进行进行矩阵乘法，会改变原本密文槽的对应关系，因此我们采用如下的公式进行计算 [18]:

$$X \cdot w = \sum_{0 \le j < N/2} (u_j \odot Rot(w; j)) \tag{2.4}$$

其中 $u_j = (X_{0,j}, X_{1,j+1}, \ldots, X_{\frac{N}{2}-j-1,\frac{N}{2}-1}, X_{\frac{N}{2}-j,0}, \ldots, X_{\frac{N}{2}-1,j-1}) \in \mathbb{Q}^{N/2}$，其中 $\odot$ 表示向量间对应元素的哈达玛积，同态进行的线性变换如算法1所示：

算法 1 同态线性变换算法

输入： $ct \in R_q^2, u_j \in \mathbb{Q}^{N/2}$

输出： ct'

1: $ct' \leftarrow \lfloor \tau^{-1}(u_0) \cdot ct \rceil \pmod q$

2: **for** $j = 1$ to $N/2 - 1$ **do**

3: $\quad ct_j \leftarrow \lfloor \tau^{-1}(u_j) \rceil \cdot Rot(ct; j) \pmod q$

4: $\quad ct' \leftarrow Add(ct', ct_j) \pmod q$

5: **end for**

6: **return** ct'

2.5 算法评估指标

为评估优化过后的 CKKS 算法的可靠性，本文采用了多个评价指标，分别从同态加密算子运算结果，逻辑回归训练过程参数变化情况以及逻辑回归模型训练结果三个方面对训练结果进行评价。

平均绝对值误差（Mean Absolute Error，MAE）, 是衡量预测误差的指标，它计算的是每个样本的绝对误差的均值。

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

其中，y_i 表示第 i 个样本的真实值，$\hat{y}_i$ 表示第 i 个样本的预测值，n 表示样本总数。

平均绝对百分比误差（Mean Absolute Percentage Error，MAPE）是另一种衡量预测值与真实值误差的指标，它计算的是每个样本的百分比误差的均值。

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \times 100\% \tag{2.5}$$

其中，y_i 表示第 i 个样本的真实值，$\hat{y}_i$ 表示第 i 个样本的预测值，n 表示样本总数。

准确率（Accuracy）是评估模型分类效果的基本指标，定义为模型正确纠正的错误数占总错误数的比例。准确率越高，表示模型的纠错能力越强。准确率的计算方法公式如下

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.6}$$

其中，TP 表示真正例（True Positive），即模型预测为正例且真实值也为正例的样本数量；TN 表示真反例（True Negative），即模型预测为反例且真实值也为反例的样本数量；FP 表示假正例（False Positive），即模型预测为正例但真实值为反例的样本数量；FN 表示假反例（False Negative），即模型预测为反例但真实值为正例的样本数量。

第三章 CKKS 算法性能瓶颈分析

3.1 基于整体流程的性能瓶颈分析

完整的基于服务器端的同态运算共包括同态加法、同态旋转、同态乘法三种同态运算。其中同态乘法包括密文间的同态乘法以及密文和常数间的乘法。本节希望测算不同算子的运行时间以确定哪个算子为性能瓶颈，并对其进行优化。因为在进行同态密文乘法过后，必须要进行重缩放操作，大部分情况下会将重缩放操作视为乘法的一部分，但是本文希望对不同算子的运行时间进行更精准的评估，因此在进行性能测算时，本文将重缩放操作视作独立的算子而非乘法的一部分。本文首先测算同态加法、同态旋转、同态密文乘法、同态密文-常数乘法以及重缩放五个算子的运行时间。本文在进行单个算子的性能分析时，为保证数据的有效性，将参数统一设置为 $\log N = 16$，$logQ = 1200$，$logP = 30$。其中 $\log N$ 表示多项式的最大的幂指数的位数，$logQ$ 表示模数的位数，$logP$ 表示模数削减或增加的位数，关于参数的设置将在5.1节进行详细描述。

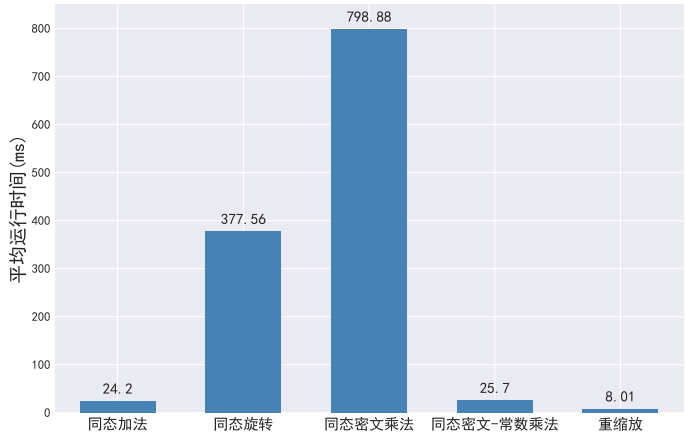


图 3.1 同态算子的运行时间

本节采用的输入数据为一个长度为 16，实部和虚部取值范围均为 (-1,1) 的随机生成的复数向量。由于生成的向量为随机值，且密文大小和明文值相关，为保证数据的可靠性，我们在进行单个算子的性能分析时，测算每一个算子的运行时间，并且连续运行五次求取平均值，其结果如图3.1所示。由实验结果可知，同态密文乘法的运行时间为同态旋转运行时间的两倍以上，为同态加法以及同态密文-常数乘法运行时间的 30 倍以上，约为重缩放的运行时间的 100 倍。因此在所有的同态算子中，同态密文乘法为主要的性能瓶颈。

3.2 基于同态密文乘法的瓶颈分析

正如3.1描述的那样，同态密文乘法为整个同态运算的性能瓶颈，因此本文需要对同态密文乘法的计算过程进行进一步详细的梳理，并将同态密文乘法内部的运算过程拆解为不同的算子评估其运行时间，以方便对其进行进一步的优化。

正如2.1节描述的那样，密文结构为形如 (b,a) 的二元组，其中 b 和 a 均为模 Q 意义下的最大幂指数为 N 的多项式。同态密文乘法需要首先输入两个密文，分别为 $ct_1=(b_1,a_1)$ 以及 $ct_2=(b_2,a_2)$。其首先通过 CRT 构建 RNS，即将原本模 Q 意义下的的多项式划分为 k 个模 q 意义下的多项式，其中 $q \ll Q$，得到 $ct_1=(rb_1,ra_1)$ 以及 $ct_2=(rb_2,ra_2)$。之后计算 ct_1 和 ct_2 的乘积，计算乘积需要首先采用 NTT 算法将原本的多项式从系数表示转化为点值表示，然后计算得到 $(rb_1\cdot rb_2,(ra_1+rb_1)\cdot(ra_2+rb_2),ra_1\cdot ra_2)=(rbb,rab,raa)$，并将计算好的结果进行进行 INTT 变换得到多项式的系数表示。然后进行逆 CRT 变换得到 (bb,ab,aa)，正如2.3节描述的那样，还要令 aa 乘以转换密钥以保证用以解密的密钥不发生增长，并将结果与 ab 相加。其乘法过程与密文乘法过程类似，同时由于模数发生改变，需要重新进行 CRT 变换以及逆 CRT 变换。最后将变换结果进行大数之间的加法以及减法操作，得到最终结果。

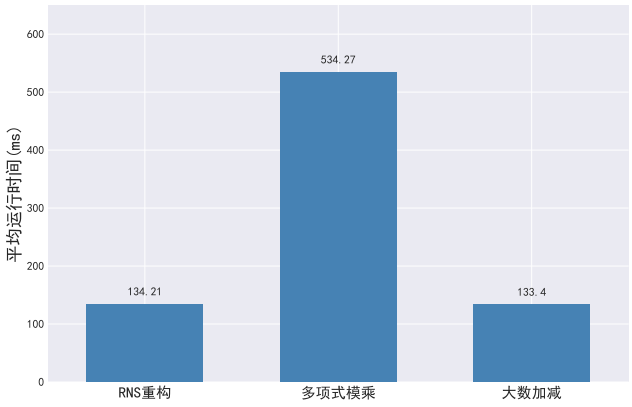


图 3.2 同态密文乘法内部算子运行时间

根据同态密文乘法完整的实现方案，本文将同态密文乘法过程划分为 RNS 重构，多项式模乘以及大数加减三个算子。其中 RNS 重构主要包括 CRT 变换以及逆 CRT 变换，多项式乘法主要包括 NTT 变换，INTT 变换，密文之间的乘法以及密文和转换密钥之间的模乘，大数加减主要包括。我们采用和3.1节一致的参数设置计算五次，分别统计这三个算子的平均运行时间，其结果如图3.2所示。由结果可知，多项式模乘的运行时间为 RNS 重构以及大数加减的四倍左右，因此多项式模乘为主要的性能瓶颈，本文将对多项式模乘进行进一步的详细优化。

第四章 基于 GPU 的同态密文乘法实现

4.1 整体实现方案

在3.2节中，我们对同态密文乘法进行梳理，并且将同态密文乘法划分为三个算子分别是 RNS 重构，多项式模乘以及大数加减三个算子，并且经过对运行时间的测算，发现多项式模乘为主要的性能瓶颈。因此我们的划分依据是将多项式模乘尽可能转移至 GPU 进行执行，并且在进行多项式模乘的过程中本质是对向量进行操作，其数据依赖少，可以进行并行计算，与 GPU 高并行度的特点。

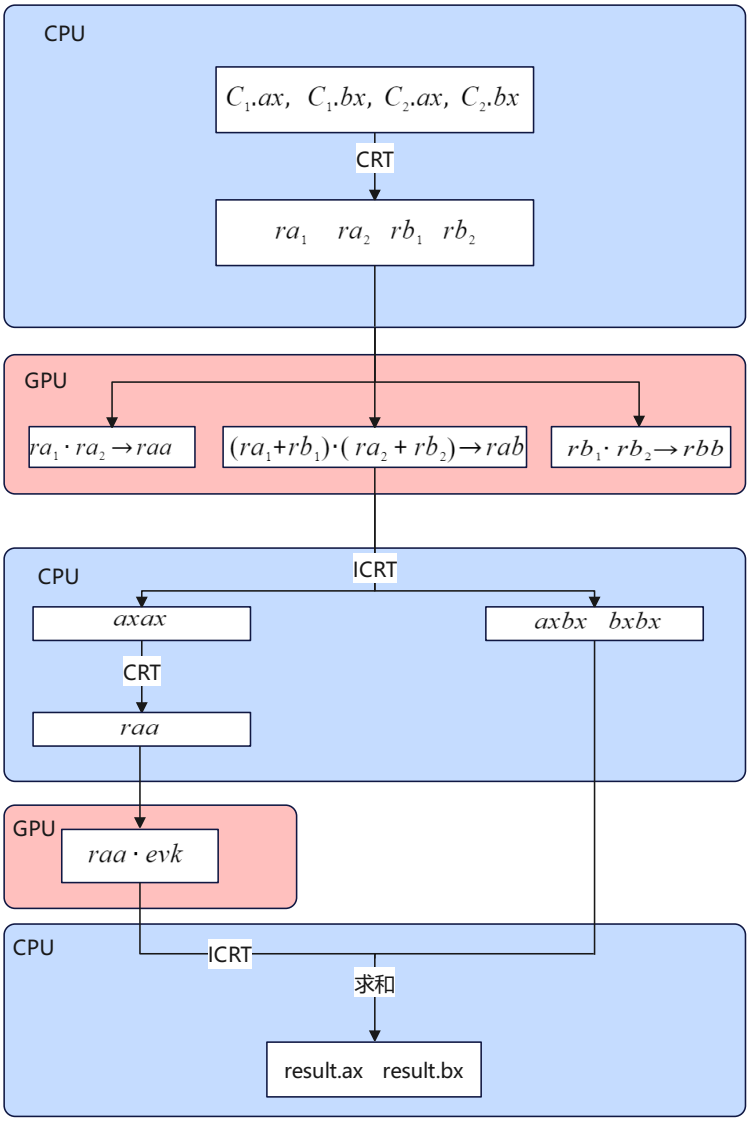


图 4.1 同态密文乘法的 GPU 实现方案

其划分方案如图4.1所示，我们将针对两个密文进行的 CRT 变换，针对密文多项式乘法结果的逆 CRT 变换，针对密文乘法结果中的 ax 部分的 CRT 变换，密钥转换结果的逆 CRT 变换以及最后的大数加法以及减法均放置于 CPU 中进行执行。将密文间的多项式模

乘，以及密文与转换密钥之间的多项式模乘均放置于 GPU 中进行优化，以并行的方式进行执行。其中 NTT 与 INTT 变换在每次规约的过程，由于需要对数据进行两两处理，其并行度为 N/2。在进行对应元素点积的过程中，由于数据之间没有依赖性，其对每个通过点值进行表示的多项式进行相乘的过程中，最大的并行度为 N。在 GPU 中，常量内存其保存的数据只可读不可写，因其具有常量缓存，所以拥有比全局内存更高的访问速度，但是其数量有限仅有 64KB，由于 NTT 与 INTT 的计算复杂度为 $O(NlogN)$ 而多项式点乘的复杂度为 $O(N)$，为保证 NTT 与 INTT 能够实现更高的效率，我们将 NTT 与 INTT 中需要重复访问的变量，分配以常量内存，以更好的提升其运行效率。

4.2 传输方案设计

大整数运算本身会带来高额的计算开销，因此正如2.2节所述，需要在 CPU 端通过 CRT 构建 RNS，对大整数多项式进行分解将其分解为多组小整数多项式。分解以后降低了计算开销以及使计算变得可行，但是增加了传输的数据量，从而增加了通信开销，为避免因为通信开销增加，从而降低程序的运行效率，因此本文设计了四个 cuda 流进行传输于采用异步数据拷贝方式完成 CPU 与 GPU 之间的数据传输，以实现计算与数据传输之间的重叠，从而实现了通信时间的隐藏，其结构如图4.2所示，由于计算时间比通信时间更长，因此能够更好的隐藏通信时间。

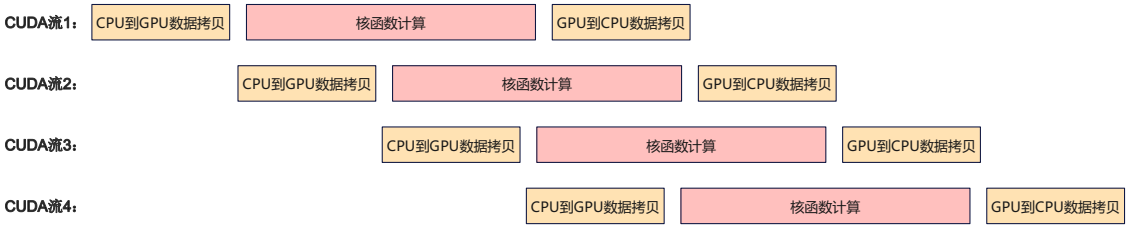


图 4.2 异步数据传输示意图

同时，由于异步传输由 GPU 中的 DMA 直接实现，不需要主机直接参与。而如果采用同步的数据传输函数，主机在向一个 CUDA 流发出数据传输的命令后，将无法立刻获得控制权，必须等待数据传输完毕，因而无法同时从另一个 cuda 流中调用核函数。在使用异步传输时，需将主机内存定义为锁页内存，以保证在程序运行期间其物理地址保持不变。对于数据的异步传输, 本文采用 cuda::memcpy_async 函数对其进行实现。

第五章 实验结果与分析

5.1 实验环境及参数设置

本文所有的实验采用的服务器相关配置如表5.1所示。其运行需要安装的软件依赖以及本服务器运行的版本如表5.2所示。其中 CUDA 提供 GPU 编程的相关支持，gcc 提供编译的相关支持，GMP 和 NTL 则提供了 CPU 多线程编程的支持。

表 5.1 服务器配置

设备	型号	数目
CPU	6 核 Intel(R) Xeon(R) Gold 5320 CPU @ 2.20GHz	4
GPU	NVIDIA Tesla A100	1
磁盘	7.86TB Intel P5510	1

表 5.2 软件依赖

软件包	版本
CUDA	11.7
gcc	7.5.0
GMP	6.1.2
NTL	11.5.1

实验过程中需要设置的参数为 $\log N$，$logQ$，$logp$，$\log N$ 为密文多项式幂指数的最大位数，$\log Q$ 表示密文多项式的系数的最大位数，$logp$ 为缩放比例。正如2.1节所描述的那样，$\log N$ 和 $\log Q$ 和密文的安全位数相关，而 $logp$ 则和明文向量以及模数增加或减少的位数相关。本文使用的所有参数均满足安全位数 $\lambda > 80$ 规定的参数，其由 LWE 评估方案计算得出 [19]。本文将 $\log N$ 设置为 18，由于本文的逻辑回归在训练过程并没有采用自举操作，因此需要设计足够大的 Q 来确保 Q 不会在逻辑回归结束前耗尽。在逻辑回归场景下，$logQ$ 的计算公式如下所示 [19] 。

$$logQ = wBits + lBits + iter(3 * wBits + 2 * pBits) \tag{5.1}$$

其中 lbits 表示噪声所需的位数, iter 表示迭代次数，wBits 表示密文模数 Q 的小数部分所需的位数，pBits 表示多项式系数的小数部分所需的位数。

5.2 可靠性验证

本节首先针对同态运算的单个算子的可靠性进行验证，之后对逻辑回归的整体流程的可靠性进行验证。验证方式为将输入的明文向量进行加密，加密以后进行相关的同态运算，之后将运算的结果进行解密。最后将初始明文的运算结果和解密以后的结果进行比较，计算初始明文的运算结果和经过同态加密后的运算结果的误差，观察误差是否在合理范围内。

5.2.1 单个算子的可靠性验证

同态运算包括同态加法、同态乘法以及同态旋转。其中同态乘法又包括密文的同态乘法以及密文和常数之间的同态乘法。正如2.3节描述的那样，同态乘法之后需要进行重缩放操作以保证结果的正确性，由于本节是对同态运算的可靠性验证，而不是对其进行性能的评估与分析，因此本节将重缩放操作视为同态乘法的一部分。因此共包括同态加法、同态旋转、同态密文乘法、同态密文-常数乘法四个基础算子。本节进行可靠性验证时输入的数据与 节保持一致，对其进行加密后完成同态运算, 将同态运算的结果进行解密，将明文运算结果与采用同态加密数据的运算结果进行比较，分别计算实部和虚部的 MAE，其结果如表5.3所示，其中同态旋转产生的误差最大，其原因在于完成旋转以后需要进行密钥转换操作，而旋转密钥是一个较大的多项式。由结果可知，初始明文的运算结果和经过同态加密后的运算结果的实部和虚部的 MAE 的数量级均在在 10^{-6} 至 10^{-9} 之间，因此可以认为经过同态加密以后的密文进行同态运算以后的结果是可靠的。

表 5.3 同态算子的可靠性验证

同态算子	实部 MAE	虚部 MAE
同态加法	1.90×10^{-8}	1.72×10^{-8}
同态旋转	8.29×10^{-7}	1.22×10^{-6}
同态密文乘法	2.1×10^{-8}	1.9×10^{-7}
同态密文-常数乘法	1.06×10^{-8}	9.63×10^{-9}

之后本文将基于 GPU 实现的同态密文乘法的运算结果进行解密，将明文运算结果与采用同态加密数据的运算结果进行比较，分别计算了实部和虚部的 MAE 值，并同基于 CPU 实现的同态密文乘法进行比较，其结果如表5.4 所示，二者不存在明显差异，因此可以认为基于 GPU 实现的同态密文乘法是可靠的。

表 5.4 同态密文乘法的可靠性验证

	CPU	GPU
实部 MAE	2.1×10^{-8}	1.92×10^{-7}
虚部 MAE	1.9×10^{-7}	6.7×10^{-8}

5.2.2 逻辑回归场景下的可靠性验证

由于同态运算会造成误差累积，其中主要的误差累计来自于同态密文乘法与同态旋转。由于真实的应用场景下，通常不会只进行一次运算而是将同态加密以及同态运算操作应用于机器学习或深度学习模型。因此本文在逻辑回归的应用场景下，对同态加密的可靠性进行验证以验证真实场景下的同态加密的可靠性。对于逻辑回归整体的验证，本文采用信用卡违约数据集，该数据集共有 23 个特征，最后一列为标签，其值为 0 或者 1 表示是否发生信用卡违约行为，该数据集共包含 30000 个样本。本文将其中 90% 的数据用于训练，10% 的数据用于测试，本文为采用密文训练的逻辑回归与采用明文训练的逻辑的参数设置相同的初值并设置相同的超参数，将模型的训练的迭代次数为 10，学习率设置为 0.8。在迭代次数为 10 次且满足安全位数不小于 80 的情况下，本文将 $logN$ 设置为 17，将 $logQ$ 设置为 1545。

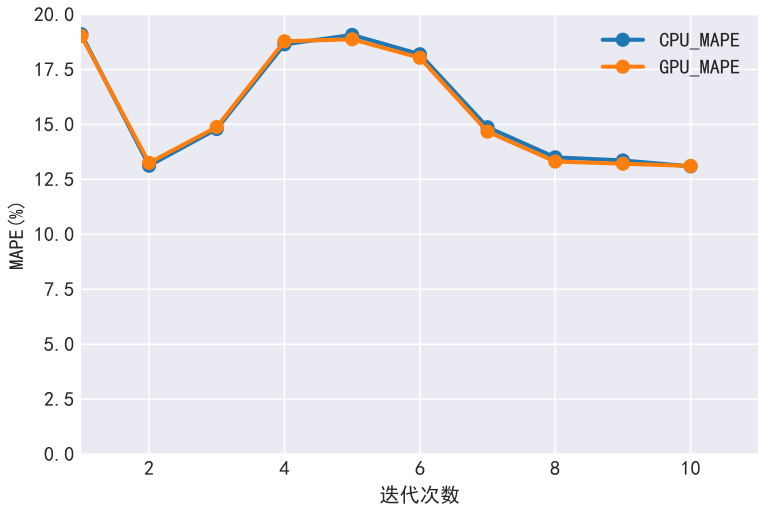


图 5.1 明文训练参数和密文训练参数的 MAPE 变化情况

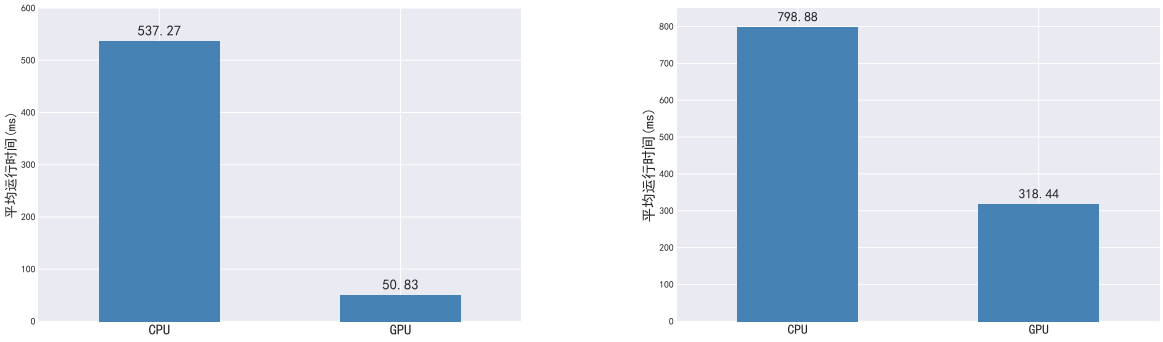
为验证逻辑回归场景中的 CKKS 算法的可靠性，本文在保证输入数据、参数初始化，以及超参数设置完全一致的情况下，分别对基于明文实现的逻辑回归算法以及基于密文实现的逻辑回归算法进行训练，并比较每一次迭代过程中的参数，由于参数不断在更新，因此采用对单个算子进行验证时采用的 MAE 进行评估不能准确反映出误差的变化，且参数不

存在非常接近于 0 的情况，因此本文在评估基于密文训练得到的参数与基于明文训练得到的参数之间的误差时没有采用 MAE，而是采用 MAPE，其通过百分比表示，与数值本身的大小无关，更适合评估参数之间的误差。无论是 CPU 实现方案还是 GPU 实现方案，基于密文训练得到的参数与基于明文训练得到的参数的 MAPE 均没有超过 20%, 且随着模型的收敛其值还有减小的趋势，并且 CPU 实现方案与 GPU 实现方案的变化过程较为相似，其变化过程如图 5.1 所示，因此在实际模型的训练过程中，不会存在因为误差累计而导致参数出现异常的情况。在第十次迭代完成时，本文在信用卡违约数据集的测试集中分别计算了基于密文训练得到的模型与基于明文训练得到的模型的预测准确率，其准确率均为 77.75%。因此可以认为经过同态加密以后的密文在真实场景下得到的结果是可靠的。

5.3 性能分析

5.3.1 同态密文乘法的性能提升

在基于 CPU 的多项式模乘方案中，我们采用 CRT 将原本的多项式，分解为 K 个多项式，K 值与此时密文多项式此时的 $\log q$ 值相关，因此我们将基于 CPU 的多项式模乘的并行度设置为 K，其值为 20 到 40 之间。本节首先比较了基于 GPU 优化过后的同态密文乘法中的多项式模乘与基于 CPU 实现的同态密文乘法中的多项式模乘之间的性能差距。其结果如图5.2a 所示，其运行时间从 537.27 毫秒缩短至 50.83 毫秒，性能加速比达到了 10 倍以上，取得了良好的效果。



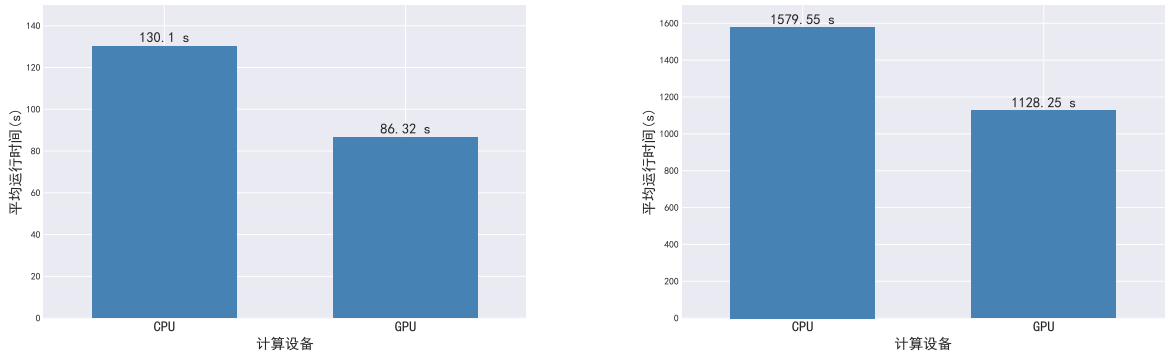
(a) 多项式模乘性能提升 (b) 同态密文乘法性能提升

图 5.2 优化前后同态密文乘法的性能比较

本节之后比较了基于 GPU 优化过后的同态密文乘法与原始的同态密文乘法之间的性能差距，其结果如图5.2b所示，其运行时间时间从 798.88 毫秒缩短至 318.44 毫秒，性能加速比为 2.5 倍，因此对于性能瓶颈进行优化是有效的，同时此时的性能瓶颈已经不再是多项式模乘操作。

5.3.2 逻辑回归场景下的性能提升

对于基于 CPU 的逻辑回归实现，由于采用的数据的特征数目为 23 个，本节将其并行度设置为 23。本节首先分别测算添加了 GPU 实现的同态密文乘法的基于同态加密的逻辑回归模型与原始的基于同态加密的逻辑回归模型的平均迭代时间，比较优化前后的性能差距，其结果如图5.3a所示，其平均迭代时间从 130.1 秒下降至 86.32 秒，其加速比达到了 1.5 倍。



(a) 多项式模乘性能提升 (b) 同态密文乘法性能提升

图 5.3 优化前后同态密文乘法的性能比较

本节之后分别测算添加了 GPU 实现的同态密文乘法的基于同态加密的逻辑回归模型与原始的基于同态加密的逻辑回归模型的完整训练时间，其包含了加密与解密过程，比较优化前后的性能差距，其结果如图5.3b所示，其平均迭代时间从 1579.55 秒下降至秒 1131.25 秒，其加速比为 1.4 倍左右。

第六章 总结与展望

6.1 总结

同态加密技术能够比较好的应用于公有云领域针对隐私数据的 ai 模型的训练与推理，其在金融行业、医疗行业、电子商务等领域均有比较好的应用。但是由于其加密过程会对原有的数据量以及数据位数进行扩充，使得计算变得非常复杂，计算量被大大提升，从而大大增加了其运行时间，降低了运行效率。因此分析同态加密过程中的性能瓶颈并对其进行优化与加速，使其算法与底层硬件能够更好的适配从而获得更好的性能提升，变得非常有意义。针对以上问题，本文以同态加密算法中的 CKKS 算法为例，测算其不同算子的平均运行时间，以分析其性能瓶颈，并结合 GPU 的特性对其性能瓶颈进行加速，得到了以下结论：

1. 在 CKKS 算法中共包括同态加法、同态旋转、同态密文乘法、同态密文-常数乘法以及重缩放五个用于同态运算的算子，其中同态密文乘法为主要的性能瓶颈。在同态密文乘法中，主要包括 RNS 重构，多项式模乘以及大数加减三个算子，其中多项式模乘为关键性能瓶颈。
2. 利用 GPU 的高并行性，能够较好的加速依赖程度比较低的数据，通常能够实现 10 倍以上的加速比，同时 GPU 中含有不同类型的内存组织，更合理的分配不同层次的内存的使用，能够使得访存更加高效。
3. 在进行 CPU 与 GPU 之间的数据拷贝过程中，应当优先考虑进行异步的数据传输，因为当计算时间大于传输时间时，采用计算采用异步传输能够较好的隐藏传输时间，而当传输时间大于计算时间时能够较好的隐藏计算时间，从而提升运行效率。

6.2 展望

CKKS 算法有着非常广泛的应用场景，为机器学习以及深度学习的相关场景下的隐私保护提供了解决方案。如何进一步降低数据依赖性实现更深度的优化以及基于 GPU 进一步对自举过程进行优化，以实现针对完整 CKKS 算法过程的 GPU 优化方案，使其适应于更多的应用场景是本文未来的研究方向。

参考文献

[1] Rivest R L, Shamir A, Adleman L. A method for obtaining digital signatures and public-key cryptosystems[J]. *Communications of the ACM*, 1978.

[2] Paillier P. Public-key cryptosystems based on composite degree residuosity classes[J]. *Proc. EUROCRYPT'99, Czech Republic, May*, 1999.

[3] Gentry C. A fully homomorphic encryption scheme[M]. A fully homomorphic encryption scheme, 2009.

[4] Gentry, Craig. Fully homomorphic encryption using ideal lattices[C]. 2009. 169--178.

[5] Cheon J H, Kim A, Kim M, et al. Homomorphic encryption for arithmetic of approximate numbers[C]. 2017.

[6] Fan S, Wang Z, Xu W, et al. Tensorfhe: Achieving practical computation on encrypted data using gpgpu[C]. 2023. 922--934.

[7] Jung W, Kim S, Ahn J H, et al. Over 100x faster bootstrapping in fully homomorphic encryption through memory-centric optimization with gpus[J]. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021, 2021, Issue 4:114--148.

[8] Riazi M S, Laine K, Pelton B, et al. Heax: An architecture for computing on encrypted data[C]. New York, NY, USA: Association for Computing Machinery. 2020. 1295–1309.

[9] Yang Y, Zhang H, Fan S, et al. Poseidon: Practical homomorphic encryption accelerator[C]. 2023. 870--881.

[10] Samardzic N, Feldmann A, Krastev A, et al. F1: A fast and programmable accelerator for fully homomorphic encryption[J]. 2021.

[11] Samardzic N, Feldmann A, Krastev A, et al. Craterlake: A hardware accelerator for efficient unbounded computation on encrypted data[C]. New York, NY, USA: Association for Computing Machinery. 2022. 173–187.

[12] Kim J, Lee G, Kim S, et al. Ark: Fully homomorphic encryption accelerator with runtime data generation and inter-operation key reuse[C]. 2022. 1237--1254.

[13] Kim J, Lee G, Kim S, et al. Ark: Fully homomorphic encryption accelerator with runtime data generation and inter-operation key reuse[C]. 2022. 1237--1254.

[14] Chen H, Laine K, Player R. Simple encrypted arithmetic library - seal v2.1[C]. Cham: Springer International Publishing. 2017. 3--18.

[15] Perez-Gonzalez, Fernando, Pedrouzo-Ulloa, et al. Number theoretic transforms for secure signal processing[J]. *IEEE transactions on information forensics and security*, 2017.

[16] Harvey D, van der Hoeven J. Faster polynomial multiplication over finite fields using cyclotomic coefficient rings[J]. *Journal of Complexity*, 2019, 54:101404.

[17] Rosen K H. Elementary number theory and its applications[J]. *Addison-Wesley*, 2000.

[18] Halevi S, Shoup V. Algorithms in helib[C]. Berlin, Heidelberg: Springer Berlin Heidelberg. 2014. 554--571.

[19] Albrecht M R, Player R, Scott S. On the concrete hardness of learning with errors[Z]. Cryptology ePrint Archive, Paper 2015/046, 2015. https://eprint.iacr.org/2015/046. https://eprint.iacr.org/2015/046.

致　　谢

光阴荏苒，岁月如梭。事实上，因为降级转专业的原因，我在兰大待的时间会比别人待的久一点。但临近毕业也还会有诸多不舍，因为所谓毕业意味着与一种新的身份告别，未来将以一种新的身份继续学习工作以及生活。在大学数年，我获得了诸多成功，实现了一个又一个梦想，但我深知自己的失败远比成功更多，我也懂得这些所谓的成功也绝不是我一人的功劳。首先，我想感谢我的母校兰州大学，虽然她有诸多不完美，也有许多需要进步的空间。但是，她在我高考失利时选择了我，以及如果没有她的培养没有给我提供一个 985 的平台，我远不可能获得今日的成功。

之后，我想感谢在我本科期间为我提供过重大帮助的老师们。赵志立老师是我科研路上的引路人，如果没有他的帮助我断不可能发表一篇 SCI 二区论文，除了科研他还教给了我很多学习上的方法以及生活上的经验。杨民强老师是我计算机方向的启蒙者，没有他的引导，我断不可能转专业进入计算机。闫德飞老师是我刚来兰大时遇到的知心人，没有他的排忧解惑，我不可能走出许多思想上的桎梏。李宝刚老师则是我价值观以及人生观上的引路人，在他的影响下，我渐渐形成了如今对世界的态度。之后我想感谢那些帮助过我的挚友们，尹邦睿和杜鑫在我无数次深夜的抑郁与焦虑中，在电话中为我送来关心听我的吐槽给我建议。郑子安则在我刚刚转入计算机时，一路带我打怪升级，让我迅速适应了转专业之后的生活，我们也一起聊人生聊理想，在他的毕业时，对我说你是我大学期间遇到的最懂我的人，听到这句话时，我感到震惊与欣喜。高世平则陪我度过了瓶颈期，我们相互扶持，共同进步，在学业上共同取得了许多成就，前些日子他说我大学最幸运的一件事是遇见了你，为此我深感荣幸。

然后我还想感谢我那些在我保研路上提供支持与重要建议的师长与朋友，程大钊老师，徐欣学姐，孙瑞，巧玲姐，三姨，宋秀杰学长，蔡潮鑫学长，陈威学长。没有你们的帮助，我来浙大的概率会小很多。

就这篇论文而言，我最应该感谢的是王则可老师，赵志立老师，刘奇师兄，孙杰师兄，阿里的陆文杰大哥，亚琛工业大学的 OChicken 大哥，他们或为我的论文框架提供建议并帮我修订论文，或陪我一起梳理同态加密的相关原理，或给我代码的 debug 过程提供建议，或教给了我许多编程的技术。

最后我想感谢我的父母和我自己，我的父母在我成年后仍然提供了许多的经济帮助与支持，虽然我的家庭非常普通，但是他们在给我很多建议的同时，又几乎没有对我有什么约束，从而让我更有勇气去做很多别人不敢做的事。我曾经看过黄国平的博士毕业论文的致谢中说到的，“我走过了很多的路,吃过了很多的苦，才将这份博士毕业论文送到你面前。”我深知自己离一名合格的博士还有很远的路，也明白自己也没有吃过他那么多的苦，不过如今我这个相对优秀的自己，还是想感谢曾经那个坚持，拼搏，勇敢，果断的自己，在一次次失败后没有放弃，亦做了许多在当时很多人不理解，现在看来无比正确的事，达到

了在相同成长环境下，很少有人能取得的成功。希望自己在未来的博士生活中仍然能够不断进步，不要让 2022 年 9 月 28 日成为自己的顶点。

毕业论文（设计）成绩表

导师评语

建议成绩_____ **指导教师（签字）**_____

答辩小组意见

答辩委员会负责人（签字）_____

成绩_____ **学院（盖章）**_____

年 月 日