复现论文:Over 100x Faster Bootstrapping in Fully Homomorphic Encryption through Memory-centric Optimization with GPUs

摘要

摘要：本报告聚焦于完全同态加密（FHE）技术，这是一种允许在加密数据上进行无限次操作而无需解密的先进加密方法。尽管 FHE 提供了加强数据隐私的独特优势，但其应用受到高计算成本的显著限制，尤其是在进行连续操作时必需的引导（bootstrapping）过程。该过程旨在刷新累积噪声，却耗时极长，这限制了 FHE 在实际应用中的广泛采用。本研究通过在 GPU 上实现 FHE，特别是对 CKKS 方案的实现，展示了如何克服这些限制。CKKS 方案是一种支持近似数算术的有前途的 FHE 方案。我们的分析显示，FHE 操作的主要性能瓶颈在于对高主存带宽的需求，而这一需求在采用旨在减少计算量的现有优化时更加突出。因应此，我们广泛采用了内存中心的优化方法，如内核融合和重排主要功能。GPU 实现相对于最先进的 GPU 实现，在单个 FHE 乘法上表现出了 7.02 倍的加速，并且每比特的摊销引导时间为 0.423 微秒，相当于相对于单线程 CPU 实现的 257 倍速度提升。

关键词：完全同态加密；GPU；内核融合

1 引言

同态加密（HE）技术允许我们在不解密数据的前提下，对加密数据执行各种操作。这意味着，只有掌握秘密密钥的用户才能够解密这些操作的结果。同态加密的一个显著特点是，在计算过程中，除了数据的大小以外，它不会泄露任何有关输入或输出的信息。因此，同态加密被广泛认为是隐私保护计算等领域的一项核心技术。

在同态加密技术的早期发展阶段，存在对操作数量和类型的限制。然而，自从 Gentry 的开创性工作以来，这些限制被打破，从而诞生了完全同态加密（FHE）。FHE 通过引导操作（bootstrapping）允许进行无限数量的计算操作 [16]。

在这个领域的新兴关注点之一是 CKKS（Cheon-Kim-Kim-Song）方案，它因其高效的近似计算而备受瞩目 [10]。与其他 FHE 方案不同，CKKS 方案除了基础的加法和乘法（mult）操作外，还包括四舍五入操作，使得该方案能够高效地在密文上进行固定点算术运算。CKKS 方案在隐私保护数据分析和机器学习领域表现出巨大的优势。例如，在 iDash 安全基因组分析竞赛中，自 2017 年以来，大多数 HE 轨道的获胜者和亚军都采用了 CKKS 方案 [21] [19]。尽管如此，CKKS 方案的性能仍未能完全满足行业实践者的需求 [22] [24] [13] [6] [17]；特别

是在传统的 PC 环境中，使用 128 位安全参数集时，CKKS 中最为低效的引导操作甚至需要超过 60 秒的时间来完成 [9]。

在这篇研究论文中，作者们着手解决了一系列挑战，主要通过提高完全同态加密（FHE）方案的性能来实现。他们特别关注于优化基于剩余数系统（RNS）的 CKKS 方案 [10]，这一方案在多种应用中展现出了卓越的效率。更进一步，他们通过充分利用高性能的通用计算平台，即 GPU，致力于提升 FHE 操作的并行处理能力。具体来讲，CKKS 方案的所有操作，包括同态加密的乘法（HE Mult）和引导程序（bootstrapping）——分别为最常用和最耗时的操作——都已成功实现在 GPU 上运行。这一研究不仅展示了技术的创新，也为同态加密的实用应用提供了新的可能性。

2 相关工作

2.1 基于 CPU 的 CKKS 算法

CKKS 是由 Jung Hee Cheon, Andrey Kim, Miran Kim 和 Yongsoo Song 在 2017 年提出 [11]，同时开发了对应的 CPU 算法库 HEAAN。其特点主要有以下几点：1. 近似算术：CKKS 支持对加密的近似数值进行加法和乘法运算，这对于处理浮点数特别有用。2. 批处理：CKKS 可以对一组数值进行批处理，即同时对多个加密值进行运算，从而提高效率。3. 可扩展性：CKKS 允许调整参数以平衡安全性、性能和精度，使其适用于不同的应用需求。4. 噪声管理：同态加密中的每次运算都会引入一些噪声，CKKS 设计了有效的噪声管理机制，确保在整个计算过程中噪声保持在可控范围内。

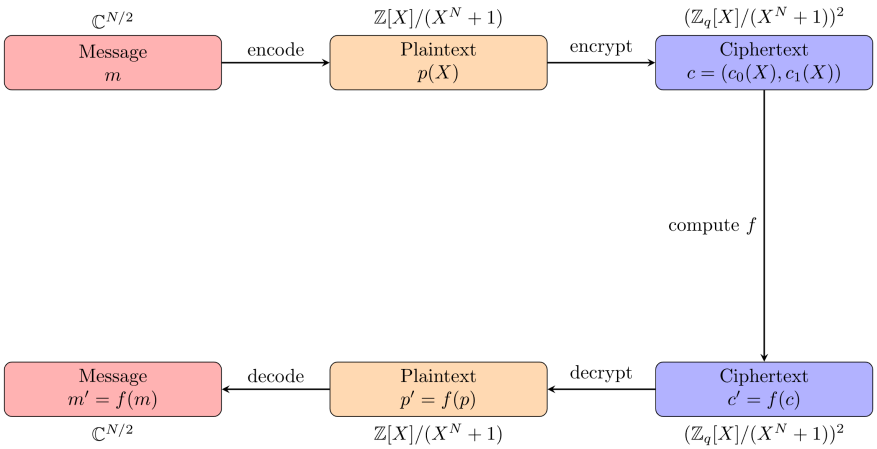

图 1. CKKS 算法流程

CKKS 算法步骤主要如图 3所示。编码和解码、加密和解密、同态操作。下面将对每个部分详细说明。

2.1.1 CKKS 编码和解码

编码

CKKS 利用整型多项式环的丰富结构实现其明文和密文空间。假设输入复数向量 $z \in \mathbb{C}^{\frac{N}{2}}$，将它编码成 $m(X) = \frac{Z[X]}{X^N+1}$，N 表示多项式模的次数，一般 N 取 2 的幂。其中 $\Phi_M = X^N + 1$

(M=2N) 叫做 M 次分圆多项式，分圆多项式是不可约的，即它们不能被分解为更低次数的多项式的乘积。明文空间即多项式环 $R = \frac{Z[X]}{X^N+1}$。此时 R 和 $\mathbb{C}^{\frac{N}{2}}$ 并不同构。但是 $\frac{\mathbb{R}[X]}{X^N+1}$ 和 $\mathbb{C}^{N/2}$ 是同构的。所以，编码的过程可以分为两步，第一步为先将输入的复数向量同构映射到实数多项式环 $\frac{\mathbb{R}[X]}{X^N+1}$，第二步是将第一步的结果映射到整数多项式环 $\frac{Z[X]}{X^N+1}$，从实数到整数这一过程会损失一定的精度。

定义如下映射 σ：$\forall m \in \frac{\mathbb{C}[X]}{X^N+1}$，$\sigma(m) = m(\xi), m(\xi^3), \ldots, m(\xi^{2N-1})) \in \mathbb{C}^N$, 这里的 ξ^{2i-1} 代表 $X^N + 1$ 的 N 个原根。这里的 σ 是双射同态，所以任何向量都将唯一地编码到对应的多项式之中。定义 $\mathbb{C}^N$ 的子环 $\mathbb{H} = \{z \in \mathbb{C}_N : z_j = \overline{z_{N-j}}\}$，自然投影映射 π：$\forall t \in \mathbb{H}, \pi(t) = (t_0, t_1 \ldots t_{\frac{N}{2}}) \in \mathbb{C}^{\frac{N}{2}}$，$\pi$ 和 σ 都是同构映射，两者的逆映射记作 π^{-1}，σ^{-1}。

第一步过程如下，取出复数向量 $z \in \mathbb{C}$，做自然投影映射的逆映射 $\pi^{-1}(z) \in \mathbb{H}$ ，为了保证精度，定义缩放因子 Δ，用得到的结果乘缩放因子放大整数部分，再做嵌入映射的逆映射，得到 $\sigma^{-1}(\Delta \cdot \pi^{-1}(z)) \in \mathbb{R}[X]/(X^N + 1)$，这里得到的结果即为实数多项式。

第二步是将得到的实数多项式转换成整数多项式，简单的说对系数进行四舍五入即可，CKKS 采用了一种坐标随机舍入（coordinate-wise randomized rounding）的方法，将多项式系数按照和相邻整数距离的远近分配概率，然后按这个概率决定向上取整还是向下取整。综上所述，编码过程可以表示为 $m(X) = \lfloor \sigma^{-1}(\Delta \cdot \pi^{-1}(z)) \rceil \in R$。

解码

解码过程基本上与编码过程相反，即计算 $\pi(\sigma(\Delta^{-1} \cdot m)) \in \mathbb{C}^{\frac{N}{2}}$。

2.1.2 CKKS 加密与解密

CKKS 使用的是一种公钥加密方案，其中生成一个私钥和一个公钥。公钥用于加密，可以共享，私钥用于解密。在加密前首先要进行初始化相关参数。

加密

首先初始化以下参数：安全等级 λ、深度上限 L、密文模数 N、一个特殊模数 P 以用于重缩放。选取一个误差分布 χ_e，和随机分布 χ_r 然后定义 $Q = q_0 \cdot p^L$，使得 N 满足安全等级 λ。

初始化一个私钥 sk，并计算公钥 $pk \in R_Q^2$，设置辅助密钥 $evk \in R_Q^2$，生成一个 $r \leftarrow \chi_r$ 和 $e_0, e_1 \leftarrow \chi_e$，对于明文多项式 m，输出密文 $c \leftarrow r \cdot pk + (m + e_0, e_1) mod$ Q

解密

对于密文 $c \in \mathcal{R}_q^2$，计算 $< c, sk > mod$ q。q 为当前层的模数，是 Q 的约数。

2.1.3 CKKS 同态操作

首先介绍一下 CKKS 加密算法中的一些限制，CKKS 是一个有限层次全同态加密，每一个密文对应了一个深度，深度一共有 L 个层。其中最大的层中的模数规模为 $Q = q_0 \cdot p^L$，它的下一层的模数规模为 $q_0 \cdot p^{L-1}$，以此类推，有 L 层，即选定 L 个素数，他们的大小与 p 在

同一数量级，不同的素数独立分配给不同的层用于计算，随着层的减小，模数规模下降。上限 L 规定了 CKKS 算法中乘法操作次数的上限，即当乘法操作次数超过 L，密文将无法被解密。

CKKS 支持的同态操作如下：加法、乘法、旋转、重缩放，下面将分别对每个操作进行介绍。

加法

加法运算的是将两个密文相加，给定一个复数向量加密生成的密文，将实部同实部相加，虚部同虚部相加，返回结果，伪代码如下：

Algorithm 1 HADD

$\mathbf{ct}_0 \to (a_0, b_0), \mathbf{ct}_1 \to (a_1, b_1)$
$d_0 \leftarrow (a_0 + a_1)$
$d_1 \leftarrow (b_0 + b_1)$
return (d_0, d_1)

乘法

乘法运算是将两个密文进行相乘，给定两个复数，实部同实部、虚部同虚部、实部同虚部做 Hadamard 乘法，得到三维的向量，通过 keySwithc 进行重线性化，将三维向量转换成二维向量，伪代码如下：

Algorithm 2 HMUL

$\mathbf{ct}_0 \to (a_0, b_0), \mathbf{ct}_1 \to (a_1, b_1)$
$d_2 \leftarrow (a_0 \odot a_1), d_0 \leftarrow (b_0 \odot b_1)$
$d_1 \leftarrow (a_1 \odot b_0 + b_1 \odot a_0)$
$(c_0', c_1') \leftarrow \mathrm{KeySwitch}(d_2, \mathbf{evk})$
return $c_{mul} = (d_0 + c_0', d_1 + c_1')$

重缩放

重缩放主要是用于应对 CKKS 乘法带来的规模增长问题，随着乘法运算次数增加，密文规模会增加，可能有超出模数范围的风险，所以引入了重缩放技术，将密文调到浅层，并缩小密文规模。伪代码如下：

Algorithm 3 RESCALE

$\mathbf{ct} \to ([a]_{\mathcal{C}_\ell}, [b]_{\mathcal{C}_\ell})$
$a'^{(j)} \leftarrow [q_\ell^{-1}(a^{(j)} - \mathrm{NTT}([\mathrm{iNTT}(a^{(\ell)})]_{q_j}))]_{q_j, j \in [0, \ell-1]}$
$b'^{(j)} \leftarrow [q_\ell^{-1}(b^{(j)} - \mathrm{NTT}([\mathrm{iNTT}(b^{(\ell)})]_{q_j}))]_{q_j, j \in [0, \ell-1]}$
return $([a']_{\mathcal{C}_{\ell-1}}, [b']_{\mathcal{C}_{\ell-1}})$

其中 NTT（Number Theoretic Transform） [1] 是快速数论变换，这是一种类似与 FFT 的变化方法，借助 NTT 变换可以将算法复杂度从 $\mathcal{O}(N^2)$ 降到 $\mathcal{O}(NlogN)$。

旋转

旋转操作主要是用于对密文进行旋转，目的是对原始的复数向量进行重排，首先根据旋转系数 sn 对原始密文的实部和虚部分别进行 FrobeniusMap 映射，即根据 sn 计算密文重排后的 index 并进行重排，将得到的结果同样使用 KeySwitch 进行重线性化，伪代码如下：

Algorithm 4 HROTATE

1: $\mathbf{ct} \rightarrow (a, b)$
2: $a' \leftarrow \text{FrobeniusMap}(a, sn)$
3: $b' \leftarrow \text{FrobeniusMap}(b, sn)$
4: $(a'', b'') \leftarrow \text{KeySwitch}(a', \mathbf{evk})$
5: **return** $ct' = (a'', b' + b'')$

2.2 基于 GPU 的 CKKS 算法实现

PrivFT [2] 是唯一一个利用 GPU 加速 RNS 变体 CKKS 的系统，它利用了他们之前的工作 [4] [3] 中的库，这些库实现了 Brakerski-Fan-Vercauteren (BFV) 方案 [7] [14] 的 RNS 部分 [5]。他们没有实现引导技术。

一个名为 cuFHE [8] 的开源库实现了 TFHE 方案 [12]，包括 TFHE 的引导技术，这是最快的引导技术，但每个密文最多只能容纳几个比特的明文。

3 本文方法

CKKS 算法在实际应用中，是编码加密只需要进行一次，之后是反复多次调用各种同态操作，所以用 GPU 加速的部分主要是各种同态操作。主要架构如下图所示：

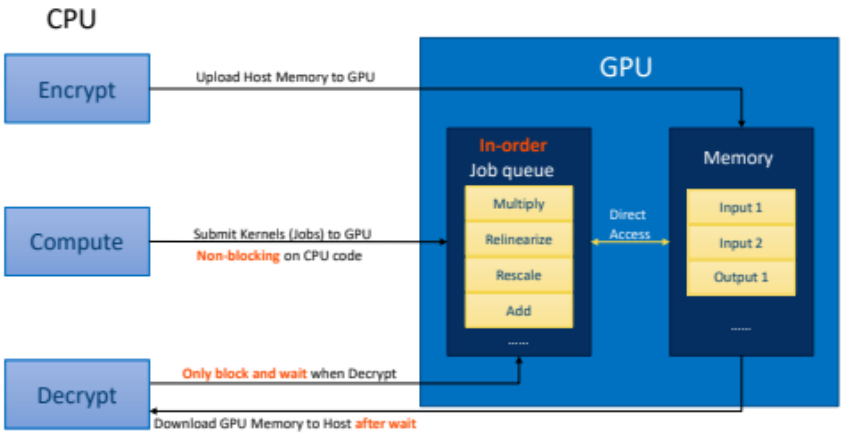

图 2. 主要框架

3.1 本文方法概述

本文主要在 GPU 上实现了以下函数：1) 元素级别 RNS 中的运算，如 NTT 域中的乘法、加法和减法；(2)NTT 和 iNTT；(3) ModUp 和 ModDown 中使用的快速基转换；(4) KeySwitch

密钥切换中的内积。

RNS 操作

本文将 RNS 运算称为二进制运算，它将残差作为输入，并对其执行元素顺向运算，例如多、加和减。

在 GPU 实现过程中，让每个 GPU 线程执行一次 RNS 操作，这样就能充分利用现代 GPU 的大规模线程级并行性，因为 GPU 可以同时运行数十万个线程。

NTT 和 iNTT

本文使用了 [18] 中的分层 NTT 实现，该实现大量利用了 GPU 中的共享内存。更具体地说，对于具有 N 个残差的 (i)NTT，我们使用 [18] 中描述的 8 个每线程 (i)NTT 内核，其中内核中的每个线程一次将 8 个残差加载到寄存器中。本文启动的内核分别执行 radix-256 或 radix-512 (i)NTT。它使用共享内存作为每个 (i)NTT 阶段时间输出的存储空间，因此每个内核只需从主存加载并存储 N 个残差一次。

ModUp 和 ModDown 中的快速基变换

快速基变换的目的是让密文在不同的数学域中进行快速转换，将密文从 S_1 的基转换到 S_2 的基。在 GPU 实现中，主要有两个 kernel，第一个 kernel 开启 $(\mathcal{S}_1|+1)\times N$ 个线程，每个线程的任务是将输入的残差与对应的常数 $\hat{Q}''^{-1}_j(\hat{Q}'''^{-1}_j)$ 相乘并 $\mathrm{mod}q_j$。第二个 kernel 主要开启 $(\mathcal{S}_2|+1)\times N$，每个线程将 $(\mathcal{S}_1|+1)$ 个余项累加，作为输出。

KeySwitch

GPU 实现部分主要有三个 kernel，一个是用于乘法，一个用于累加，还有一用于模块化的规约，伪代码如下：

Algorithm 5 KeySwitch

1: $\mathbf{evk}_i := (a_{\mathbf{evk}_i}, b_{\mathbf{evk}_i}), i = 0, 1, \ldots, \beta - 1$
2: $\mathbf{d} := \{d_2^{(0)}, d_2^{(1)}, \ldots, d_2^{(\beta-1)}\}$
3: **procedure** INNER-PRODUCT(accum, evk, d)
4: accum $= d_2^{(0)} \cdot \mathbf{evk}_0$
5: **for** each element i in set $[1, \beta)$ **do**
6: accum+ $= d_2^{(i)} \cdot \mathbf{evk}_i$
7: **end for**
8: $b'_{\mathrm{accum}} = \mathrm{BarrettModReduction}(b_{\mathrm{accum}})$
9: $a'_{\mathrm{accum}} = \mathrm{BarrettModReduction}(a_{\mathrm{accum}})$
10: **return** $(a'_{\mathrm{accum}}, b'_{\mathrm{accum}})$
11: **end procedure**

3.2 代码瓶颈分析

本文同时对 GPU 的实现进行 profile 进行分析，发现 GPU 的主要瓶颈在访存。大多数同态操作并不是受计算限制，而是受访存带宽的限制。下图为对单个同态乘法内部的各个 kernel 的执行时间的统计。可以发现 kernel 的 SM 的利用率都不高，最高才百分之七十，相反，DRAM 的利用率很高，说明两个问题：1. 单个 kernel 的计算量太小，无法占满 GPU。2. 计算无法掩盖访存延时，kernel 需要花费很多时间等待数据准备好才能进行计算。

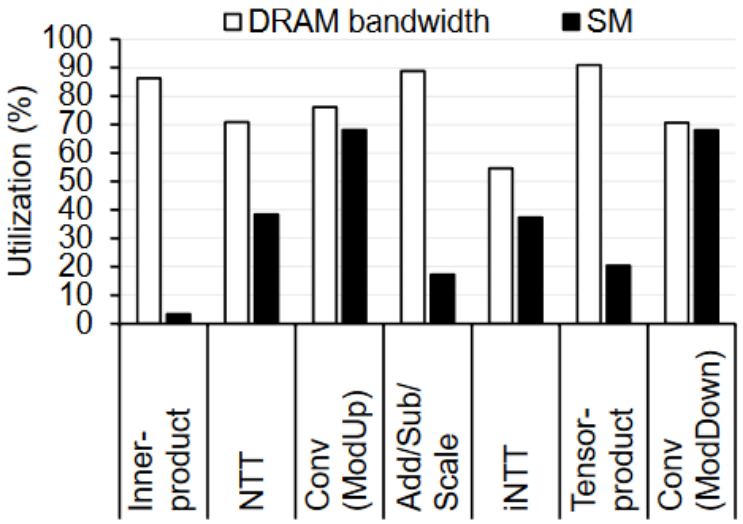

图 3. 在 GPU 实现上，单个同态乘法的执行时间、DRAM 利用率和 SM 利用率，相关参数 (N , L , dnum) = (216、45、45)

为了解决这个问题，本文提出了两个方法：1. Intra-FHE-operation Fusion 2. Inter-FHE-operation Fusion。

Intra-FHE-operation Fusion

在单个同态操作内，主要的 FHE 操作，如乘法、旋转和重新缩放，都有采用内核融合来节省内存访问次数提升性能的潜力。

ModUp 操作包括多个 iNTT 内核，每个内核都执行基于 FFT 的操作。当 dnum 足够大时，每个 iNTT 的输入大小会减小，无法充分利用 GPU 中的线程级并行性。因此可以把每次密钥切换的需要批量执行的 ModUp 的多次 iNTT 内核，融合为一个内核，以减少内存访问次数。此外，缩放内核是访存瓶颈而非计算密集型任务，可以将其与前一个 batch 的 iNTT 操作融合，来减少访存次数。

在 KeySwitch 操作中，执行内积操作时，需要对累加器进行多次读取和写入。累加器中的每个元素都是 128 位长，它占用了大部分的内存访问。为了减少内存访问次数，可以将算法中的乘法内核，乘加内核与简约内核这三种内核融合为一个单一的大内核，在该内核中执行所有的乘法、乘加和约简操作。使内核中的每个线程在其寄存器中保存部分和，直到最后一个乘加操作完成。这种简单而有效的方法显著减少了访存次数。

ModDown 中的两个逐元素操作为减法和缩放分别调用一个内核，类似 ModUp 中的内核融合，可以将其融合为一个内核，以减少内存访问次数，提高 ModDown 操作的性能。

Inter-FHE-operation Fusion

评估线性变换是 bootstrapping 的瓶颈，主要涉及 lotToCoefficient 和 CoefficientToSlot 函数分别对应同态解码与编码的过程，其核心算法为 BSGS [20]。对此，可以通过批处理明文-密文 MAC 方法来优化。

BSGS 算法中最内层的求和操作在 GPU 上存在严重的内存带宽瓶颈，可以将 CMULT 和 HADD 融合，从而减少总内存访问次数，这种融合即为批处理明文-密文 MAC。由于 CMULT 和 HADD 大多数情况下是内存绑定的，因此减少内存访问次数可以将性能提高到渐近 11/3 倍。

4 复现细节

4.1 与已有开源代码对比

论文作者开源了一部分代码，包括 NTT 与 iNTT 的 GPU 实现等，但是作者并没有将全部同态操作复现，本文在原代码的基础上，增加了如旋转、密文乘密文等同态操作的实现，以同时对相关 kernel 进行 fusion，缓解访存瓶颈。

4.2 实验环境搭建

代码使用 C++ 和 CUDA 编写，实验环境是 linux 系统，搭建 CUDA 环境。硬件配置如下，CPU：Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz；GPU：Tesla V100-PCIE 32G，driver version 12.1。

4.3 创新点

本文基于原文的基础上，实现了原仓库中未实现的同态操作，如密文旋转、密文乘以密文。同时实现了仓库中未完全实现的融合操作。除此之外，还对一些相同的 kernel 进行融合操作，即一次读入多个输入，开辟更多的空间以进行计算，减少了 GPU 的访存次数，通过增大单个 kernel 的计算量以达到提高了 GPU 的利用率的目的，实现进一步的加速。

5 实验结果分析

本文主要对比了同规模下 CPU CKKS 算法各个同态操作运行速度和论文中给出的结果以及复现的结果，论文中用到的硬件配置与复现代码用的硬件配置相同，结果如表 1 所示：

表中 CPU 的部分主要使用 HElib 这一开源库，GPU(paper) 这一列是原文中实验所得，GPU(test) 这一列为实验所得。原文中 GPU 的实现相较于 CPU 版本有着百倍以上的加速比，而在复现过程中，经过一定的优化，复现结果相较于原文结果，基本上有一定的加速。

运行时间 (ms)				
	CPU	GPU (paper)	GPU (test)	Speed up(compare to paper)
Add	1.609	0.208	0.211	0.98
Multiply (two ciphertexts)	3083	17.4	15.075	1.15
Multiply (plaintext ciphertexts)	168	0.177	0.177	1
Rotate	2890	16.83	10.784	1.560645401
Mult-Add	3188	——	0.365	
KeySwitch	——	——	2.231	
Rescale	145	0.846	0.625	1.3536

表 1. 参数规模 (N，L，dnum)=(65536,44,3) 时各个操作耗时

同时本文还对参数选择进行一定的分析，图 4 为选择不同 N 的大小时，各个操作的耗时分析。可以发现，在 N 的规模较小时，同态操作耗时差异并不大。当 N 增长到一定规模后，同态操作的耗时会随 N 规模增大而增大。N 的增量是指数级的。

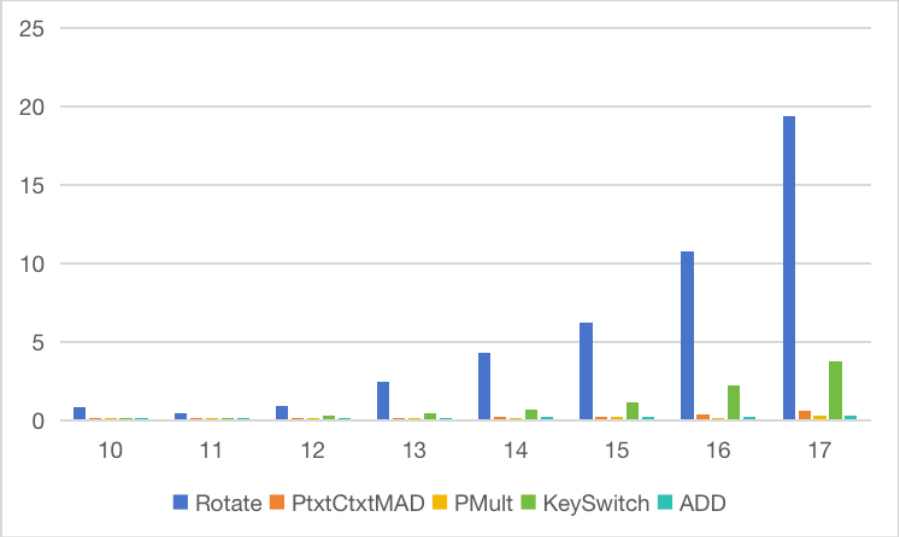

图 4. log N 的增长对各个同态操作运行时间的影响

6 总结与展望

本文是对同态加密算法 CKKS 的 GPU 实现的复现工作。在原文开放出一部分的代码的基础上，进行扩展和进一步的优化。最终取得的效果 xxx。这篇论文是第一篇在 GPU 端实现 CKKS 等同态加密算法的文章，后续有基于此研究使用 Intel GPU 对 CKKS 进行加速 [23]，还有使用 Nvidia GPU 的最新部件 Tensor core 进行加速的文章 [15]，都取得了不错的效果。这为对本代码的进行进一步的优化提供了广泛的思路。

参考文献

[1] Ramesh C Agarwal and C Sidney Burrus. Number theoretic transforms to implement fast digital convolution. *Proceedings of the IEEE*, 63(4):550–560, 1975.

[2] Ahmad Al Badawi, Louie Hoang, Chan Fook Mun, Kim Laine, and Khin Mi Mi Aung. Privft: Private and fast text classification with homomorphic encryption. *IEEE Access*, 8:226544–226556, 2020.

[3] Ahmad Al Badawi, Yuriy Polyakov, Khin Mi Mi Aung, Bharadwaj Veeravalli, and Kurt Rohloff. Implementation and performance evaluation of rns variants of the bfv homomorphic encryption scheme. *IEEE Transactions on Emerging Topics in Computing*, 9(2):941–956, 2019.

[4] Ahmad Al Badawi, Bharadwaj Veeravalli, Chan Fook Mun, and Khin Mi Mi Aung. High-performance fv somewhat homomorphic encryption on gpus: An implementation using cuda. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 70–95, 2018.

[5] Jean-Claude Bajard, Julien Eynard, M Anwar Hasan, and Vincent Zucca. A full rns variant of fv like somewhat homomorphic encryption schemes. In *International Conference on Selected Areas in Cryptography*, pages 423–442. Springer, 2016.

[6] Song Bian, Masayuki Hiromoto, and Takashi Sato. Hardware-accelerated secured naïve bayesian filter based on partially homomorphic encryption. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 102(2):430–439, 2019.

[7] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Annual Cryptology Conference*, pages 868–886. Springer, 2012.

[8] Gizem Selcan Cetin, Wei Dai, Bogdan Opanchuk, and Kitsu. cufhe, 2018. GitHub repository.

[9] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In *Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part I 37*, pages 360–384. Springer, 2018.

[10] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full rns variant of approximate homomorphic encryption. In *Selected Areas in Cryptography–SAC 2018: 25th International Conference, Calgary, AB, Canada, August 15–17, 2018, Revised Selected Papers 25*, pages 347–368. Springer, 2019.

[11] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*, pages 409–437. Springer, 2017.

[12] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster packed homomorphic operations and efficient circuit bootstrapping for tfhe. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 377–408. Springer, 2017.

[13] Jack Lik Hon Crawford. *Fully Homomorphic Encryption Applications: The Strive Towards Practicality*. PhD thesis, Queen Mary University of London, 2020.

[14] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive*, 2012.

[15] Shengyu Fan, Zhiwei Wang, Weizhi Xu, Rui Hou, Dan Meng, and Mingzhe Zhang. Tensorfhe: Achieving practical computation on encrypted data using gpgpu. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 922–934. IEEE, 2023.

[16] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.

[17] Kai Huang, Ximeng Liu, Shaojing Fu, Deke Guo, and Ming Xu. A lightweight privacy-preserving cnn feature extraction framework for mobile sensing. *IEEE Transactions on Dependable and Secure Computing*, 18(3):1441–1455, 2019.

[18] Sangpyo Kim, Wonkyung Jung, Jaiyoung Park, and Jung Ho Ahn. Accelerating number theoretic transformations for bootstrappable homomorphic encryption on gpus. In *2020 IEEE International Symposium on Workload Characterization (IISWC)*, pages 264–275. IEEE, 2020.

[19] Tsung-Ting Kuo, Xiaoqian Jiang, Haixu Tang, XiaoFeng Wang, Tyler Bath, Diyue Bu, Lei Wang, Arif Harmanci, Shaojie Zhang, Degui Zhi, et al. idash secure genome analysis competition 2018: blockchain genomic data access logging, homomorphic encryption on gwas, and dna segment searching, 2020.

[20] Daniel Shanks. Class number, a theory of factorization, and genera. In *Proc. Symp. Math. Soc., 1971*, volume 20, pages 415–440, 1971.

[21] XiaoFeng Wang, Haixu Tang, Shuang Wang, Xiaoqian Jiang, Wenhao Wang, Diyue Bu, Lei Wang, Yicheng Jiang, and Chenghong Wang. idash secure genome analysis competition 2017, 2018.

[22] Fan Yin, Yandong Zheng, Rongxing Lu, and Xiaohu Tang. Achieving efficient and privacy-preserving multi-keyword conjunctive query over cloud. *IEEE Access*, 7:165862–165872, 2019.

[23] Yujia Zhai, Mohannad Ibrahim, Yiqin Qiu, Fabian Boemer, Zizhong Chen, Alexey Titov, and Alexander Lyashevsky. Accelerating encrypted computing on intel gpus. In *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 705–716. IEEE, 2022.

[24] Chengliang Zhang, Suyi Li, Junzhe Xia, Wei Wang, Feng Yan, and Yang Liu. {BatchCrypt}: Efficient homomorphic encryption for {Cross-Silo} federated learning. In *2020 USENIX annual technical conference (USENIX ATC 20)*, pages 493–506, 2020.