并行进阶要求选题–多项式乘法及其 NTT 优化

1 选题背景

多项式乘法作为基础的数学运算，在信号处理，计算机图形学，密码学等领域有着广泛的应用。在该并行选题中，我们重点关注同态加密中的多项式乘法，同态加密可以在密文上进行运算，对运算后的密文进行解密的结果等于明文直接运算的结果，这在安全领域中是非常重要的技术，而同态加密的其中一项关键组成部分便是多项式乘法。

2 问题定义

在本次选题中，我们并不需要关注同态加密的复杂数学原理，而只关注其中的多项式乘法部分及其 NTT 优化。下面为多项式乘法的基本定义：给定多项式 $f(x) = a_{n-1}x^{n-1} + ... + a_1x + a_0$ 和多项式 $g(x) = b_{m-1}x^{m-1} + ... + b_1x + b_0$，设 $h(x) = f(x)g(x)$，则有 $h(x)$ 的 x^k 项系数 h_k 为：

$$h_k = \sum_{i=0}^{k} a_i b_{k-i}$$

朴素的多项式乘法计算需要 $O(n^2)$ 的时间复杂度，这在实际应用中是无法接受的，所以实际应用中一般通过 FFT 或者 NTT 来加速多项式乘法的计算，由于本选题重点关注同态加密领域中的多项式乘法，所以我们重点研究 NTT 算法的并行加速。但为了方便 NTT 的理解，下文我们首先对 FFT 和其逆变换进行简要介绍。

2.1 FFT 和 IFFT

设 $\omega_n^k = \cos\frac{2k\pi}{n} + \mathrm{i}\sin\frac{2k\pi}{n}$（下文简称 ω^k）

设矩阵 W(大小为 $n * n$) 为：
$$\begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 & \cdots & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^4 & \cdots & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^8 & \cdots & \cdots & \omega^{2(n-1)} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & \cdots & \cdots & \cdots & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix}$$

由于 $\omega^n = 1$，故矩阵 W 可以仅使用 $\omega, \omega^2, \omega^3, ..., \omega^n$ 表示。

化简后的矩阵 W 为：
$$\begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & \cdots & \cdots & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \cdots & \cdots & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \cdots & \cdots & \cdots & \omega^{n-2} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & \omega^{n-1} & \omega^{n-2} & \omega^{n-3} & \cdots & \cdots & \cdots & \cdots & \omega \end{bmatrix}$$

假定 n 维向量 $X=[x_0,x_1,\cdots,x_{n-1}]^T$，则向量 X 的傅里叶变换为 $Y=WX$。

使用矩阵-向量乘法可以在 $\Theta(n^2)$ 的时间复杂度内计算出结果,但实际上可以使用分治算法在 $\Theta(nlogn)$ 的时间复杂度内求解。设 n 为 2 的次幂，若实际求解中 n 不是 2 的次幂，可以对向量进行补 0 扩充至 2 的次幂。设矩阵 W 的第 i 行，第 j 列 (设最上面一行为第 0 行，最左边一列为第 0 列) 的值为 $W(i,j)$，则 W 具有如下性质:

1. 当 i 为偶数且 $0\le i\le n-1$ 时, $W(i,j)=W(i,n/2+j)(0\le j\le n/2)$;
2. 当 i 为奇数且 $0\le i\le n-1$ 时, $W(i,j)=-W(i,n/2+j)(0\le j\le n/2)$。

所以当计算 $Y=WX$ 时，有如下结论:

1. 当 i 为偶数且 $0\le i\le n-1$ 时, $y_i=\sum_{j=0}^{n/2-1}W(i,j)\left(x_j+x_{j+n/2}\right)$;
2. 当 i 为奇数且 $0\le i\le n-1$ 时, $y_i=\sum_{j=0}^{n/2-1}W(i,j)\left(x_j-x_{j+n/2}\right)$。

所以向量 $[y_0,y_2,y_4\cdots,y_{n-2}]$ 是向量 $\left[x_0+x_{n/2},x_1+x_{n/2+1},\cdots,x_{n/2-1}+x_{n-1}\right]$ 的傅里叶变换; 向量 $[y_1,y_3,y_5\cdots,y_{n-1}]$ 是向量 $\left[x_0-x_{n/2},\omega(x_1-x_{n/2+1}),\cdots,\omega^{n/2-1}(x_{n/2-1}-x_{n-1})\right]$ 的傅里叶变换。由上述两式即可得到向量 $Y=[y_0,y_1,y_2,\cdots,y_{n-1}]$，Y 即为 X 的傅里叶变换。

又当 n = 2 时，向量 $[x_0,x_1]$ 的傅里叶变换直接等于向量 $[x_0+x_1,x_0-x_1]$。所以向量 X 的傅里叶变换可以利用分治算法求解。

傅里叶逆变换即已知傅里叶变换的结果 $Y=[y_0,y_1,y_2\cdots,y_{n-1}]$,求原始向量 $X=[x_0,x_1,\cdots,x_{n-1}]$。即求解矩阵 W 的逆矩阵 W^{-1}。经过推导可得 W^{-1} 为 $\frac{1}{n}\begin{bmatrix}1&1&1&\cdots&1\\1&(\omega^{n-1})^1&(\omega^{n-1})^2&\cdots&(\omega^{n-1})^{n-1}\\1&(\omega^{n-2})^1&(\omega^{n-2})^2&\cdots&(\omega^{n-2})^{n-1}\\\vdots&\vdots&\vdots&\ddots&\vdots\\1&(\omega^1)^1&(\omega^1)^2&\cdots&(\omega^1)^{n-1}\end{bmatrix}$

2.2 运用 FFT 加速多项式乘法

给定多项式 $f(x)=a_{n-1}x^{n-1}+...+a_1x+a_0$ 和其系数组成的向量 $X_f=[a_0,a_1,...a_{n-1}]^T$，对向量 X_f 进行傅里叶变换可以得到 $Y_f=[A_0,A_1,...,A_{n-1}]$，有傅里叶变换的矩阵向量乘法定义可知，$A_k=f(w^k)$。同时我们知道 n 点可以唯一确定一个 $n-1$ 次的多项式，那么在 FFT 之后，我们已经有了 $(w^0,f(w^0)),\cdots,(w^{n-1},f(w^{n-1}))$ 这 n 个点。同理，对应多项式 $g(x)$，我们也可以得到 $(w^0,g(w^0)),\cdots,(w^{n-1},g(w^{n-1}))$ 这 n 个点，那么我们可以得到

$$(w^0,f(w^0)g(w^0)),\cdots,(w^{n-1},f(w^{n-1})g(w^{n-1}))$$

而这实际上是多项式乘法结果 $h(x)$ 上的 n 个点，那么我们只要对这 n 个点做一次快速傅里叶逆变换即可得到多项式 $h(x)$ 的各项系数。

2.3 NTT

FFT 利用了复数单位根的特殊性质来对傅里叶变换进行优化，但存在着浮点误差，浮点计算量大等问题。而 NTT 通过了拥有与复数单位根相同性质的另一数学工具来避免了浮点运算（使用 Z_p 上的单位根来代替复数单位根）。首先我们介绍 Z_p（p 为质数）的性质：$Z_p=0,1,2,\cdots,p-1$，对于 $a,b\in Z_p$，两数的加法结果为 $(a+b)mod\ p$, 减法，乘法和加法同理。Z_p 上同样可以进行除法，设 a，b 除法结果为 x，则 x 满足 $a\equiv bx(mod\ p)$，x 的求解可以通过费马小定理来完成，如果在高中阶段没有学习过数论，大家可以自行查阅资料学习。

和 FFT 的处理思路一样，我们令 n 为 2 的次幂，但在 NTT 中，我们还需要 $n|(p-1)$（这可以通过选择一个合适的 p 来完成）。令 g 为 p 的一个原根[1]，则 Z_p 上的单位根定义为：

$$g_n = g^{\frac{p-1}{n}}$$

容易证明,g_n 和单位根 w_n 具有相同的性质，所以将之前 FFT 中所有的 w_n 换成 g_n，将所有复数运算换成有限域上的运算即为 NTT。

3 算法简介

NTT 朴素的实现方式分为递归和迭代两种，其中迭代运行需要使用蝶形变换进行预处理，大家在实验时可以两者均实现然后比较两者的性能，同时可以在这两种实现方式上分别做并行优化。但实际上目前均是在迭代实现上做并行优化，助教也推荐大家重点在迭代实现的 NTT 上进行优化，所以接下来的讲解均是对迭代实现的 NTT 的并行优化，在阅读下面的资料之前，建议大家先能做到用 C++ 实现正确的串行迭代 NTT。

3.1 SIMD

朴素的 SIMD 优化即对蝶形运算过程中的加法和减法进行向量化。这部分较为简单，结合 NTT 的代码很容易便能实现。但是 NTT 中的一个耗时非常长的操作为模乘操作，但由于朴素的模乘无法利用 SIMD 进行优化[2]，所以这里首先介绍一个模乘的优化算法：Montgomery 模乘。

Montgomery 模乘的主体思想是选择一个有关模数的参数 r，使得计算 $a \times b \times r^{-1}\ mod\ p$ 这个操作可以通过一系列加减乘法和位运算来替换掉原本的取模运算, 大家可以自行查阅资料进行学习。

同时可以考虑更多的 NTT 优化，例如四分 NTT 等可以更好利用 SIMD 特性的 NTT 算法。也可探索其他的模乘算法。同时可以根据模数大小设计不同的 NTT 优化算法，例如模数可以用 16 位无符号整型存下时可以更好地利用 SIMD 指令的特性。

3.2 pthread/openmp

由于蝶形运算之间互相独立，朴素的多线程优化为在分治过程中将蝶形运算平均划分到各线程中去，每个线程负责一部分运算。

一个稍微复杂一些的多线程优化 NTT 的场景是在同态加密场景中，有可能模数的长度非常大[3]，无法直接进行运算。此时往往通过将模数拆解为多个小模数，在每个小模数上分别进行 NTT，最后再将每个小模数上的结果通过中国剩余定理合并，同学们可以尝试对该场景下的 NTT 进行并行化。

大家也可以自行调研一些 NTT 多线程优化相关论文然后进行实现。

3.3 MPI

MPI 与多线程的优化思路基本一致，大家可以仿照多线程的思路进行，也可自行查阅资料探索更多的内容。

[1]原根的定义大家可以自行查阅资料，一个最广为人知的模数 $p = 998244353 = 2^{23} * 7 * 17 + 1$，且 3 为 998244353 的原根，实际上 998244353 最多只能支持结果长度为 2^{23} 的多项式乘法。

[2]这里对模乘的优化指的是利用 SIMD 同时计算多个模乘，但 AVX 指令集和 Neon 指令集都不支持直接模运算的 SIMD 操作

[3]模数如果大于 2^{64}，那么 Z_p 上的运算都必须实现高精度类，运算效率大幅降低。但为了大家代码编写方便，我们假定模数均小于等于 2^{60}，但是大家在处理时需要假设无法直接进行该范围的数值运算。

3.4 GPU

GPU 优化可以采用和多线程类似的思路进行，大家也可以自行调研一些专门针对 GPU 优化的 NTT 算法。

另外由于 NTT 本质上还是矩阵向量乘法，大家可以尝试使用 GPU 对暴力矩阵乘法进行优化。

3.5 期末报告

1. 在之前实验基础上探索更多的 NTT 优化算法。
2. 将之前的实验和代码整理完善，形成一个完整的工作。
3. 对比不同并行环境下不同算法的性能。

4 代码框架说明

4.1 代码测试

测试方法类似 ann 的测试, 脚本使用、文件输入输出方法相同, 注意多项式乘法的最终结果不需要输出, 正确性可以通过函数 fCheck 判断。如果需要查看中间结果或最终结果, 可以调用函数 fWrite 写入文件, 禁止调用 std::cout 输出大量调试信息。

```
# 提交测试，第一个参数为作业编号，SIMD - 1，pthread/openmp - 2，mpi - 3
bash test.sh 1 1
```

4.2 数据说明

保证输入的所有模数的原根均为 3, 且模数都能表示为 $a \times 4^k + 1$ 的形式不同输入文件只有输入模数不同, 输入模数分别为 7340033 104857601 469762049 263882790666241 其中, 第四个模数超过了整型表示范围, 如果实现此模数意义下的多项式乘法需要略微修改框架对第四个模数的输入数据不做必要要求, 如果要自行探索大模数 NTT, 请在完成前三个模数的基础代码及优化后实现大模数 NTT(目前的同态加密中经常会出现较大模数)。

输入文件共五个, 第一个输入文件 n = 4, 其余四个文件分别对应四个模数, n = 131072 在实现快速数论变化前, 后四个测试样例运行时间较久, 推荐调试正确性时只使用输入文件 0。

4.3 框架内容

框架里只给出了多项式乘法的朴素算法, 需要先自行实现快速数论变化 (NTT) 代替原有的多项式乘法。

本选题基础要求即实现多项式乘法朴素算法的并行化算法, 进阶要求即实现 NTT 的并行化要求。

框架代码只要求正确性测试、数据读取函数和计时方法不允许更改, 其余代码仅供参考, 最终测试会测试五个输入。

如果在实验报告中进行性能对比, 需要保存一份多项式乘法朴素算法的运行结果 (运行时间较长脚本可能会自动截断, 推荐每个模数单独测试)。