28th IEEE International Conference on High Performance Computing, Data, and Analytics

A Programming API Implementation for Secure Data Analytics Applications with Homomorphic Encryption on GPUs

THE OHIO STATE UNIVERSITY

AUGUSTA UNIVERSITY

Shuangsheng Lou[§]   Gagan Agrawal[*]

Department of Computer Science and Engineering
[§]The Ohio State University, USA
[*]Augusta University, USA

Overview

- Introduction
- Challenges
- Preliminary
- Proposed approach
- Evaluation
- Conclusion

Introduction

Why secure data analytics applications?

- Cloud systems for rapidly increasing volume of data
- An attacker or adversary having an unauthorized access to the storage on the cloud can mine the data and retrieve large amounts of confidential data

Introduction

What is homomorphic encryption (HE) ?

A form of encryption that permits users to perform computations on its encrypted data without first decrypting it.

Three notable classes of scheme:

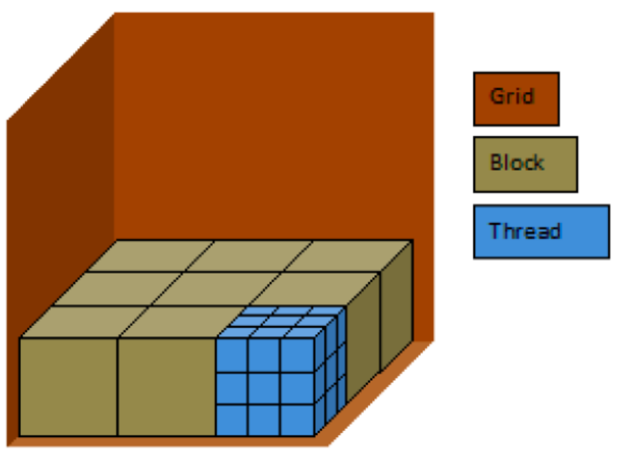
- Partially homomorphic encryption (PHE)
  PHE allows only one type of operation, either addition or multiplication, for an unlimited number of computation times
- Somewhat homomorphic encryption (SWHE)
  SWHE allows certain types of operations for a limited number of computation times
- Fully homomorphic encryption (FHE)
  FHE allows an unlimited number of operations for an unlimited number of computation times

Challenges

Goal

Secure data analytics application with HE

- With only a few operations supported using HE(multiplication and addition), data analytics algorithms can not be translated to encrypted versions without modification
- Programmability concern on HE-based data analytics on GPUs

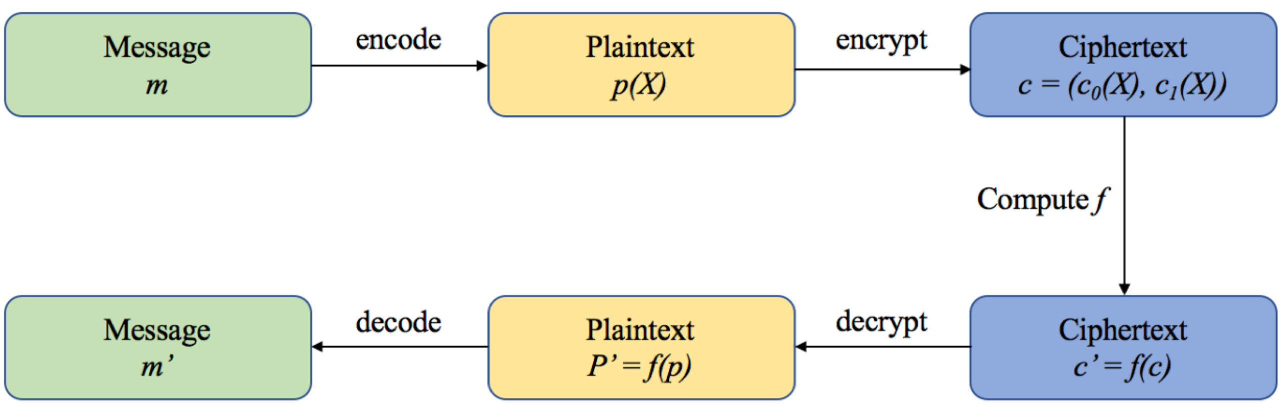


Preliminary

CKKS scheme and Microsoft Seal library

SEAL: provide high-performance and easy-to-use encryption functions that allow computations to be performed directly on encrypted data

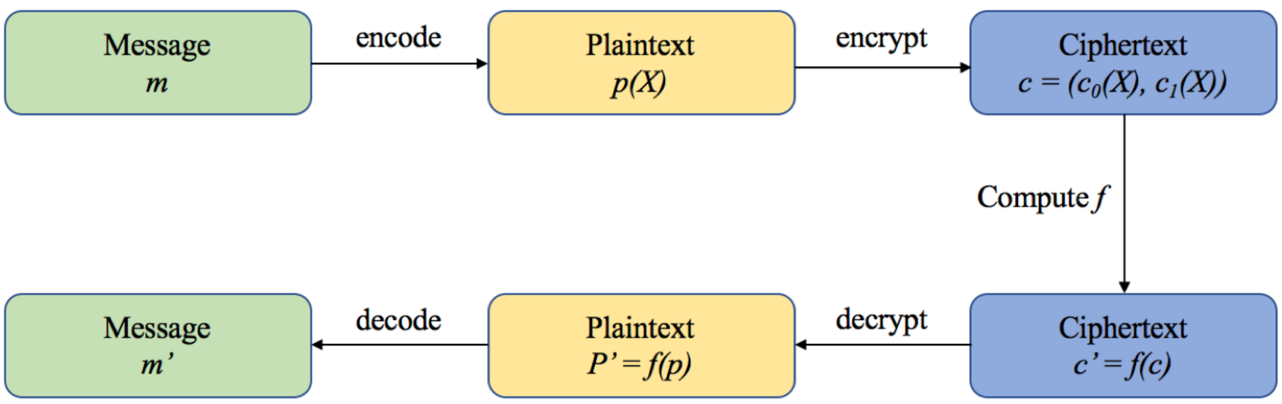
CKKS scheme: perform computations on vectors of complex values and yields approximate results



Preliminary

CKKS scheme

Message *m* : a vector of values
Plaintext *p(X)* : a plaintext polynomial
Ciphertext *c* : a couple of polynomials
Function *f* : a composition of homomorphic operations, such as addition, multiplication, and rotation



Preliminary

Key functions from Seal library

- Encode and Decode

  Transforms the plaintext to/from a polynomial
- Encryption and Decryption

  Encryption process transforms the plaintext to ciphertext and decryption process transforms the ciphertext to plaintext.
- Addition
- Plaintext-ciphertext multiplication
- Ciphertext-ciphertext multiplication
- Relinearization

  Relinearize the ciphertext size after ciphertext-ciphertext multiplication to keep a constant ciphertext size
- Rescaling

  Keep the scale constant, and also reduce the noise present in the ciphertext

Proposed Approach

Integrated functions

1)Ciphertext Multiply_Plain(Ciphertext x, Plaintext p)

It computes the multiplication between the ciphertext and plaintext, and then eliminate the scale effect of data

2)Ciphertext Multiply(Ciphertext x, Ciphertext x2)

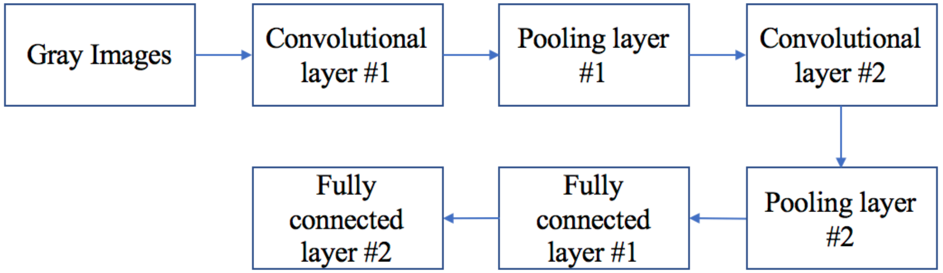
It encapsulate the relinearization and rescaling operations for the multiplication between two ciphertexts

3)vector<double> FloatToVector(double* Fvalue, size_t slot_count)

It converts the data of "**double**" type into "**vector**" type so that they can be operated with different homomorphic operations

CNN application

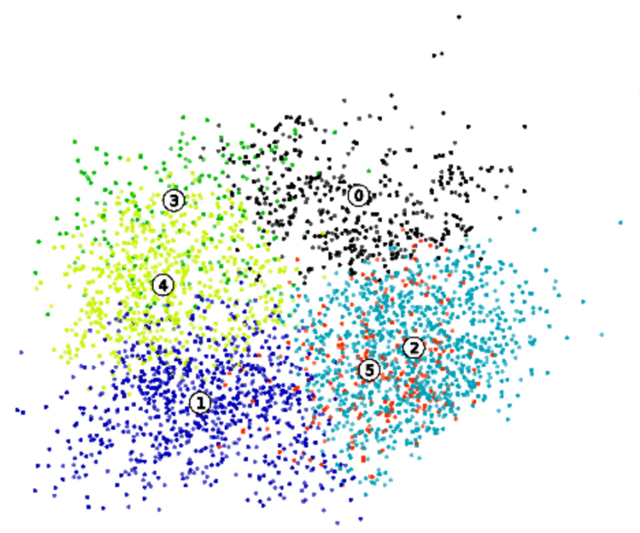
- Preprocessing layer
  Threads work concurrently on ***Block_size*** to unroll the matrix
- Convolutional layer
  matrix_multiplication is designed as a kernel function for tile-based matrix multiplication. Each block deals with a part of a number. Registers are used to store common indices
- Activation layer
  Square function is used for the activation function.
- Pooling layer
  A scaled-up version of average pooling is used to calculate the summation of values without dividing them by the number of values. We implement average pooling with addition only



K-means application

Algorithm

- Encryption
  Data points are encoded and encrypted for the following homomorphic encryption
- Assignment
  Get the squared Euclidean distance through a series of homomorphic encryption operations through ***SquaredEuclideanDistance*** function



K-means application

Algorithm

- CAM protocol

  It first decrypts the received encrypted Euclidean distance to obtain the plaintexts and runs the ***Min(.)*** to find the minimum distance
- Update

  New data centers updated
- Termination

  It verifies whether the number of iteration satisfies the pre-defined termination condition

Algorithm 2 Squared Euclidean Distance Computation with Encryption

Input: two data points t_1, t_2

Output: Squared distance of t_1, t_2

$t_2 \rightarrow$ Multiply_Plain(t_2, -1)

evaluator.add(t_1, t_2,result)

square(result, result2)

relinearize_inplace(result2, relin_keys)

rescale_to_next_inplace(result2)

KNN application

Algorithm

To identify ***k*** objects that are nearest to the given test point

Euclidean distance needs to be computed between the test data point and previous ***SquaredEuclideanDistance*** designed is applied.

Process:

- Encryption
- *SquaredEuclideanDistance*
- Find K nearest neighbors

Algorithm 3 Encrypted Version of KNN

Input: m data points $t_1, ..., t_m$ and test data point p
Output: class
1 Encode and encrypt m data points, test data point p
2 Set K
3 for i = 1 to m do
4 SquaredEuclideanDistance(t_i, p_j)
5 **end for**
6 Find K nearest neighbors using CAM protocol
$P = \{p_1, p_2, ..., p_k\}$ for $1 \leq k \leq m$
7 Identify the the p using k nearest neighbor's method.
8 Compute cluster center for c'_j

Evaluation Results

GPU configuration: Dual Intel Xeon 8268s in dual NVIDIA Volta V100 with 32GB GPU memory
CPU configuration: Dual Intel Xeon 8268s Cascade Lakes, which has 48 cores per node

1)Convolution Neural Network

Parameters:100 images as one data chunk

The **speedup** of GPU over CPU:
Without any encryption technique: 190x
Using the CKKS scheme: 81x

During matrix multiplication, we delay part of relinearization and rescaling operations to reduce their cost and performance increases 28%

Evaluation Results

2)KMeans

Parameters: number of data points is 100, number of attributes is 288 and the number of iterations is set to 100

The **speedup** of GPU over CPU:
Without any encryption technique: 170x
Using the CKKS scheme: 133x

By using CAM protocol, a part of the computation is performed directly on unencrypted data, which reduces the size of encrypted data greatly.

Evaluation Results

3)KNN

Parameters: each data chunk has 10000 points, number of labels is 3 and the value of k is set to 3

The **speedup** of GPU over CPU:
Without any encryption technique: 7x
Using the CKKS scheme: 6.7x

We delay part of relinearization and rescaling operations to reduce their cost and it achieves 11% reduction on its execution time.

Conclusion

- Programmability eased for data analytics applications with HE by designing secure advanced functions on the top of the SEAL library
- The development and introduction of integrated functions to perform comparison operation into CKKS scheme
- Demonstration of its functionality on GPUs by showing the development of three significant applications -- CNN, KMeans, and KNN
- Comprehensive experimental evaluations of three data analytics applications that compare the efficiency of the CPU-based to GPU-based implementation

Thanks for your attention!

Q?

- Shuangsheng Lou lou.125@osu.edu
- Gagan Agrawal gagrawal@augusta.edu