Parallel Computing Mode in Homomorphic Encryption Using GPUs Acceleration in Cloud

Jing Xia*, Zhong Ma, Xinfa Dai
Wuhan Digital Engineering Institute, No.1 Canglong Bei Road, Jiangxia, Wuhan, P.R.China.

* Corresponding author. Email: 673718032@qq.com; xiajing_csic@163.com
Manuscript submitted January 18, 2019; accepted March 25, 2019.
doi: 10.17706/jcp.14.7.451-469

Abstract: As an important encryption algorithm of performing computations directly on ciphertext without any need of decryption and compromising privacy, homomorphic encryption is an increasingly popular research topic of protecting privacy of the data in cloud security research. However, it will be a heavy workload for resources as the high computational complexity of homomorphic encryption. Therefore, GPU acceleration is employed to speed up homomorphic encryption. Motivated by this observation, we utilize parallel computing mode in DGHV algorithm with GPUs acceleration based on CPU-GPUs hybrid system. Our main contribution is to present a parallel computing mode for large-scale data encryption based on CPU-GPUs hybrid system as fast as possible. Specifically, we applies parallel computing mode in DGHV with GPUs acceleration to reduce the time duration and provide a comparative performance study. We further design pipeline architecture of processing stream to accelerate the speed of DGHV algorithm. Furthermore, experimental results validate that different parallelism has the corresponding granularity in parallel computing mode. Experimental results show that our method gains more than 84% improvement (run time), 67% improvement (run time), and 80% improvement (run time) compared to the sequential data encryption, sequential homomorphic addition, and sequential homomorphic multiplication in DGHV algorithm respectively.

Key words: Cloud computing, data security, GPU, homomorphic encryption, parallel.

1. Introduction

It is a popular scenario that cloud customers utilize cloud computing platform to store, compute, and analyze their large-scale data on the third-party cloud providers, which have the risk of private data leakage. As is generally known, standard encryption algorithms help protect sensitive data from outside attackers, nevertheless, they cannot be utilized to compute ciphertext directly. The best way of utilizing homomorphic encryption presents a very useful way that can compute ciphertext directly. However, computational complexity of the homomorphic encryption is huge. Therefore, so far these studies have mainly concentrated on improving execution efficiency of homomorphic encryption in order to reduce the time duration. This is the motivation for us to construct parallel computing mode in DGHV algorithm using GPUs acceleration. In cloud computing platform, the clients are given the right of encrypting their sensitive information before sending it to the cloud server. Then the cloud server executes computation over the ciphertext directly without the need for the decryption key. For instance, in the fields of medicine and finance, user's privacy needs to be protected. Therefore, homomorphic cryptosystem is a good way of

ensuring security and privacy of data in cloud computing service mode.

Along with the rise of concept of full homomorphic encryption (FHE) first proposed by Rivest *et al.* [1], the FHE gradually become one of the most important topic in the field of cryptosystem, especially in cloud. Nevertheless, satisfactory results were not obtained for a long time before 2009. Only some semi-homomorphic or only finite-step full-homomorphic encryption schemes have been obtained, such as the RSA algorithm which only supports the multiplication homomorphism. Until 2009, Gentry [2] has proposed the realization of the FHE using bootstrapping. Subsequently, Gentry invented a fully homomorphic encryption scheme based on an ideal lattice in the same year [3]. In addition, the application value of homomorphic encryption has been started to receive attentions by scholars and entrepreneurs. A lot of variants based on Gentry's approach have emerged from that time. However, the FHE leads to high workload for computing resources and therefore making the cryptosystem computationally inefficient. In 2010, Dijk *et al.* [4] proposed another concise and easy-to-understand full-homomorphic encryption scheme, have been called integer (ring)-based full homomorphic encryption scheme (DGHV). The DGHV performs homomorphic encryption over integers and not on lattices. Accordingly, the DGHV is suitable for computer operations in theory. By contrast, the reality is that the DGHV is impractical for actual utilization, especially in real time applications [5]. For instance, the application of homomorphic encryption in a simple plaintext search will increase the amount of computing in trillions of times. At present, much parallel computing mode has been designed successfully [6], especially by using GPUs acceleration [7]. Under the background, the initial motivation is that we use the parallel computing mode in homomorphic encryption to create this paper's idea.

The goal of this paper is to improve the execution efficiency of homomorphic encryption by using GPUs acceleration. Specifically, the proposed parallel computing mode can benefit the improvement of the efficiency for the DGHV quantitatively. Our approach is mainly divided into three parts in this paper.

First, in order to find out the noninterference subtasks, the sequential processing stream is analyzed and the process unit is defined. In order to find out the bottleneck of the cryptosystem, we can analyze the serial homomorphic encryption based on the only CPU system which is referred to as Serial Homomorphic Encryption (SHE). Secondly, because the response time is the key consideration for the cryptosystem, we use GPUs to accelerate the DGHV in order to obtain a preferable response time. Furthermore, pipeline framework has been designed for stream processing as an optimization measure in parallel computing modes. In our study, according to utilizing the number of GPU in hybrid CPU-GPUs system which represents parallelism, we validate that different parallelism has its corresponding granularity. Finally, to verify the effectiveness of our method, we aim at test the performance of the CPU-4GPUs hybrid system by carrying out a set of experiments. We further do some comparative experiments in order to quantitatively test improvement of the execution time of the DGHV in the parallel computing mode.

The main contributions of this paper are the following. (1) The fast parallel homomorphic encryption with four GPUs acceleration denoted by FPHE is explored to improve the performance of the DGHV. (2) We find out the corresponding granularity of process unit with different parallelism. By utilizing the granularity of process unit, we further make efforts to design pipeline architecture of processing stream as an assisted acceleration of DGHV algorithm. (3) Extensive experiments are conducted to demonstrate the effectiveness of the FPHE in optimization of DGHV algorithm.

The rest of the paper is organized as follows. Section 2 reviews related work on homomorphic cryptosystem using various parallel systems. The core of our paper, Section 3, proposes a method that is used to accelerate the DGHV. Section 4 shows the experimental results. Finally, we draw some conclusions and discussions in the Section 5.

2. Related Work

Recent technological advances have sparked the popularity of cloud computing platform, which provides many services that support the storage and intensive computing of the customer's data. Nevertheless, these promising services bring many challenging security and privacy issues [8]. In recent years, much work has been done in the area of cloud security.

The reliability and security of data remain major concerns in cloud computing environment. Fugkeaw *et al.* [9] propose a hybrid access control model to support fine-grained access control for big data outsourced in cloud storage systems. In order to check the cloud data reliability of the user's uploads the data in server, Swathi *et al.* [10] propose an approach using certificate less public auditing scheme which is used to generate key value. Jiang *et al.* [11] propose a novel cloud security storage solution which is based on autonomous data storage, management, and access control. Moreover, due to loss of ownership of user's data, the reliability and security of data stored in the outsourcing environment give rise to a lot of concerns. Therefore, protecting user's information privacy in outsourcing environment is becoming increasingly important. For example, Shu *et al.* [12] propose a privacy-preserving task recommendation scheme which is used to encrypt information about tasks and workers before being outsourced to the crowdsourcing platform. Zhang *et al.* [13] providing a strong privacy protection to safeguard against privacy disclosure and information tampering. These techniques are utilized to protect privacy of user's data. Especially, in order to protect data privacy in cloud computing environment, the study of cryptosystem adapted to the cloud is the main approach. Data outsourcing is a typical use scenario of the cloud storage services, wherein data encryption is a good approach enabling the data owner to retain its control over the outsourced data [14]. Esposito *et al.* [15] describes a solution that applies encryption to protect data sovereignty in federated clouds.

Nevertheless, some encryption algorithms are not applicable for computing ciphertext directly in cloud computing platform. On the contrary, homomorphic encryption can execute operations of ciphertext directly, which called the feature as algebraic homomorphism [16]-[18]. However, as the high computational complexity of homomorphic encryption, many scholars concentrate on how to improve the execution efficiency for sake of the popularity of the cloud computing platform [19]. Jayapandian *et al*. [20] have tried to propose a method combines the characteristics of both probabilistic and homomorphic encryption techniques in order to improve execution efficiency from the point of homomorphic algorithm itself. Nevertheless, there is no obvious decrease in time complexity according to the experimental results. Thus, we did not improve the execution efficiency from the perspective of optimizing the homomorphic encryption itself.

Lately, scholars have researched many parallel computing modes in homomorphic encryption [21]-[23]. Sethi *et al.* [24] proposed a model to parallel homomorphic encryption based on MapReduce framework. Similarly, Zhaoe Min *et al.* [25] proposed another parallel homomorphic encryption scheme based on MapReduce environment with 16 cores and 4 nodes. The above two studies validate that a good parallel computing scheme can improve the run time of the homomorphic encryption. So we have focused on applying efficient parallel computing modes in homomorphic encryption to reduce the time complexity.

Because of the fact is that the privacy of sensitive personal information is an increasingly important topic as a result of the increased availability of cloud services, especially in the medical and financial field. Khedr *et al*. [26] propose a secure medical computation using GPU-Accelerated homomorphic encryption scheme. Accelerating somewhat homomorphic cryptography is also researched by many scholars. For example, Tian *et al.* [27] design a GPU-assisted homomorphic cryptograph for matrix multiplication operation. Moayedfard *et al.* [28] present the parallel implementations of somewhat homomorphic encryption scheme over integers using Open-MP and CUDA programming to reduce the time duration. Nevertheless, many

applications require both addition and multiplication operations, so we study the improvement of the executing efficiency for the DGHV using CPU-4GPUs. Wang Wei *et al.* [29] present the first GPU implementation of a full homomorphic encryption scheme.

However, to the best of our knowledge, there is no research trying to quantitatively research the acceleration improvement of multiple GPUs compared to a single GPU. Moreover, this paper present a parallel processing stream scheme for large-scale data encryption based on CPU-GPUs hybrid system as fast as possible, which is also different from the existing research. Experimental results also find out the corresponding granularity of process unit with different parallelism. More generally, the corresponding granularity of process unit with different parallelism further validate that granularity can be of significant benefit to optimize parallel computing mode, which is the main difference between our method and that of the works above.

3. Method Formulation and Analysis

In this section, we propose parallel computing modes in homomorphic encryption using GPUs acceleration.

We first describe the background knowledge of the DGHV. Then the serial computing mode of the DGHV is explained. Then, by analyzing the noninterference with each process unit, the parallelizable part is explained. Finally, based the parallelizable part, the parallel computing mode is illustrated.

3.1. Background Knowledge of the DGHV

In cloud computing scenario, the cloud server needs to process and respond to the user's query based on executing operation on ciphertext which is uploaded by users. Homomorphic encryption is a special encryption system that can effectively solve this problem. The key point is that the output of ciphertext is equivalent to the result of performing the same operation on the plaintext and then encrypting it. The definition of the homomorphic encryption scheme is as follows.

U is the plaintext algebraic system of data set *S*. The f_1, f_2, ..., f_n are a set of operations. The p_1, p_2, ..., p_n are a set of predicates. And the s_1, s_2, ..., s_n are a set of different constants. C denotes the cryptographic algebraic system of the data set *S*, and its corresponding operation operations, predicates, constants denoted by a set of f_1', f_2', ..., f_n', a set of p_1', p_2', ..., p_n', a set of s_1', s_2', ..., s_n' respectively. Encrypted data is mapped from *U* to *C*, and decrypted data is mapped from *C* to *U*. The entire cryptosystem can be represented as follows.

$$U =< S; f_1, f_2, ..., f_n; p_1, p_2, \dots, p_n; s_1, s_2, ..., s_n > \tag{1}$$

$$=< S'; f_1', f_2', ..., f_n'; p_1', p_2', ..., p_n'; s_1', s_2', \dots, s_n' > \tag{2}$$

Suppose the encryption function is $\Phi: S \rightarrow S'$, and the corresponding decryption function is $\Phi^{-1}: S' \rightarrow S$, $(a, b, \dots) \in S$, and $(d_1, d_2, \dots) \in S'$. The cryptosystem can operate directly on encrypted data. The decryption function Φ^{-1} must satisfy the homomorphism from C to U. The formalized description is as follows.

$$(\forall i)(a, b, c \ \dots)[f_i(a, b, \dots) = c] \Longrightarrow f_i(\Phi(a), \Phi(b), \dots) = \Phi(c) \tag{3}$$

$$(\forall i)(a, b, c \ \dots)p_i(a, b, \dots) \equiv p_i(\Phi(a), \Phi(b), \dots), \ 且\Phi^{-1}(s_i') = s_i \tag{4}$$

The function Φ has the homomorphism properties. For example, when we wants to obtain the function value $f_1(d_1, d_2)$, we can instead of compute the value of $f_1'(\Phi^{-1}(d_1), \Phi^{-1}(d_2))$. Because of the properties

of the function Φ, it can be known as equation five.

$$f_1(d_1, d_2) = f_1'(\Phi^{-1}(d_1), \Phi^{-1}(d_2)) \tag{5}$$

Specifically, different homomorphic operations can perform different homomorphic calculations on ciphertext. In our study, the DGHV both have the additive homomorphism and the multiplicative homomorphism. Generally, the homomorphic encryption algorithm has the homomorphic properties as follows.

Additive Homomorphism: If there is an effective algorithm that makes the plaintext x, y satisfies the equation as $E(x + y) = E(x) \oplus E(y)$ or $x + y = D(E(x) \oplus E(y))$, and do not reveal the plaintext x, y.

Multiplicative Homomorphism: If there is an effective algorithm that makes the plaintext x, y satisfies the equation as $E(x \times y) = E(x) \otimes E(y)$ or $x \times y = D(E(x) \otimes E(y))$, and do not reveal the plaintext x, y.

A homomorphic encryption scheme consists of four part, namely homomorphic key generation (KeyGen), homomorphic encryption (Enc), homomorphic decryption (Dec), and homomorphic assignment (Evaluate). All of the four parts are probabilistic polynomials time complexity.

The DGHV encryption scheme supports both addition and multiplication in the integer range, which is using only basic modular arithmetic, instead of using the ideal lattice concept in polynomial rings. The specific process is as follows.

KeyGen: The encryption select a random positive prime as the secret p and an n bit odd integer from an interval $p \in (2Z + 1) \cap [2^{n-1}, 2^n)$.

Enc: For every bit data, we define q and r as random positive integers, where q is a large positive integer and r satisfy the equation as $|2r| < p/2$. To encrypt a bit of plaintext m belong to the binary set, which describe as $m \in \{0, 1\}$. In order to generate ciphertext c in the DGHV algorithm is defined as $c = pq + 2r + m$, where q is a large random number and r is a small random number. In practical application, for a composite message denoted by $M = m_1 m_2 \ldots m_n$, where $n \in N$(the set of natural numbers). Subsequently, the ciphertext C is defined as follows. In our study, the encryption step is executed by using the four GPUs.

$$\boldsymbol{C} = E(p, M) = E(p, m_1)E(p, m_2) \ldots E(p, m_n) = c_1 c_2 \ldots c_n \tag{6}$$

$$c_i = E(p, m_i) = pq_i + 2r_i + m_i \tag{7}$$

Dec. The decryption algorithm uses the secret key and ciphertext to decrypt the ciphertext, which is described as the equation eight.

$$m = D(p, c) = (c \bmod p) \bmod 2 \tag{8}$$

Evaluate: Since the public key is open to the public, we can obtain the $c - pq$ by the equation $c - pq = m + 2r$. Due to the interference of noise denoted by r, it is impossible to distinguish the plaintext message denoted by m, so we call the right part of the equation denoted by $m + 2r$ as the noise. While performing the decryption, it is necessary to satisfy the condition denoted by $c \bmod p = m + 2r < p/2$. In detail, only satisfying the condition, the mode operation of noise $m + 2r$ can perform the correct decryption process described by $(m + 2r) \bmod 2 = m$. As a conclusion, the noise $m + 2r$ is the key to perform the correct decryption. Therefore, if we want to construct a full homomorphic scheme, it is necessary to eliminate the noise. The re-encryption technology provides a feasible way to reduce the noise. Then Evaluate(pk_2, Dec, s', c') is performed, and the obtained result is a new ciphertext. The ciphertext is obtained by performing the encryption over the plaintext information using the secret key pk_2. This is one of the most important characteristics of the DGHV's idea, which is the decryption of the homomorphism. In

fact, the ciphertext noise obtained by performing homomorphic decryption is still slightly larger than fresh ciphertext. Therefore, in order to reduce the noise, every time before we carry out the computational operation of the ciphertext, we apply the re-encryption operation over the ciphertext. Then the noise will remain in the controllable range. The result produced by the homomorphic decryption should not be discrepant with the plaintext.

3.2. The Sequential Computing Mode of the DGHV

In order to construct parallel computing mode, the sequential computing mode needs to be analyzed. For a given data set, the sequential computing mode in DGHV can be described as the sequential following steps: initialization, data partition, task distribution, synchronization, data encryption, homomorphic addition, homomorphic multiplication, data merging, result outputting etc.

3.2.1. Sequential homomorphic encryption

The sequential encryption in the DGHV is named Sequential Homomorphic Encryption (SHE). The executing flow of the SHE based on CPU is as shown in Fig. 1. In the SHE algorithm, CPU is the only execution unit in all of the steps. Specifically, GPUs are not utilized to accelerate the run time of the SHE.

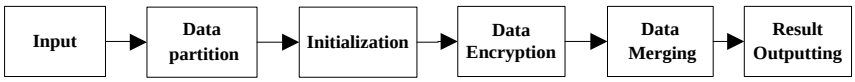

Fig. 1. The executing flow of the SHE based on CPU.

Given a composite message denoted by $M = m_1 m_2 \dots m_n$, where $n \in N$(the set of natural numbers). Encrypt M over a secret key p can be utilized by Method 1 named Sequential Homomorphic Encryption (SHE) in sequential computing mode.

Method 1: SHE
Function SHE( ) Input: $M = m_1 m_2 \dots m_n$, $m_i \in \{0,1\}$; Step1: p = KeyGen($p \in (2Z+1) \cap [2^{n-1}, 2^n)$); Step2: for each m_i In M do q = Random (q, 2^{n^5}) > 0, r = Random(r, 2^n) && $\lvert 2r \rvert < p/2$; $c_i = Enc(p, m_i) = pq_i + 2r_i + m_i$; Step3: $C = Enc(p, M) = Enc(p, m_1)Enc(p, m_2) \dots Enc(p, m_n) = c_1 c_2 \dots c_n$; Output($C = c_1 c_2 \dots c_n$);
End Function

Here, the function KeyGen is to generate the secret key p. And the function Random is the random number generating function which generates the q and r in the DGHV. The encryption process is carried out by the CPU execution in sequence. In Method 1, C denotes the ciphertext of the composite message M. In

fact, before encrypting the message M, the CPU need execute data partition subtask which spilt the data.

Step 1 utilizes the function KeyGen to generate a random number of positive prime as the secret key in DGHV. Step 2 gives the formula that calculates the ciphertext of each bit of the message M, which is denoted by c_i. Specifically, random numbers of p and r need to be generated before encryption of c_i. Step 3 outputs the encrypted result denoted by the composite message C. From the output, we can observe the ciphertext for preparing the homomorphic addition and homomorphic multiplication.

3.2.2. Sequential homomorphic addition

The sequential homomorphic addition in the DGHV is named Sequential Homomorphic Addition (SHA). The executing flow of the SHA based on CPU is as shown in Fig. 2.

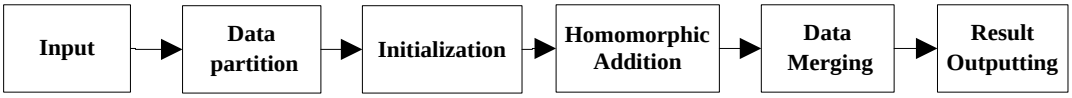

Fig. 2. The executing flow of the SHA based on CPU.

As shown in Fig. 2, when executing one of these steps, the other steps need to wait. In Special, the Method 2 presents implementation of sequential homomorphic addition between a pair of ciphertext with the same length. In sequential computing mode, all steps of the SHA are executed by CPU.

Method 2: Sequential Homomorphic Addition

Function SHA( )

Input: CA[ ] = $c_{a1}c_{a2}\ldots c_{an}$, CB[ ] = $c_{b1}c_{b2}\ldots c_{bn}$;

Step 1: N = n; C = Array[N], Sum = Array[N], i = 0;

Step 2: while N != 0 do

Temp = c_{aN} XOR c_{bN};

SUM[N] = Temp XOR i ;

Temp = Temp AND i;

i = c_{aN} AND c_{bN};

i = i XOR Temp;

N = N-1;

end while;

Output Sum;

End Function

Here, the parameter CA[ ] and CB[ ] denote two integer ciphertext with the length of n. Step 1 execute some initialization operations which assign values to corresponding variables. Step 2 introduces how to calculate the homomorphic addition over a pair of integer ciphertext. In Step 2, we use the primitive operations in order to realize the homomorphic addition, including XOR and AND operations. Step 3 outputs the sum of homomorphic addition.

3.2.3. Sequential homomorphic multiplication

The sequential homomorphic multiplication in the DGHV is named Sequential Homomorphic Multiplication (SHM). The executing flow of SHM based on CPU is as shown in Fig. 3.

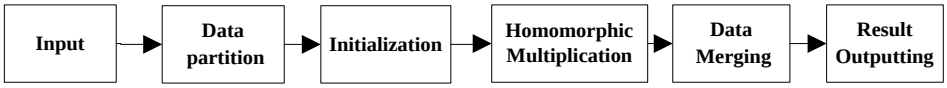

Fig. 3. The executing flow of the SHM based on CPU.

The Method 3 presents implementation of homomorphic multiplication on a pair of ciphertext. In sequential computing mode, all steps of SHM are carried out by CPU.

Method 3: Sequential Homomorphic Multiplication

```
Function    SHM( )
Input:  CA[ ] = c_a1 c_a2 ... c_an,  CB[ ] = c_b1 c_b2 ... c_bn  ;
Step 1: N = 2n; Mul = Array[N], AL = n; BL = n;
Step 2: while BL > 0 do
            C = Array[N];
            P1= n – BL;
            P2 = (AL + BL) – P1;
            while P1 > 0 do
                 C[AL + BL – P1] = 0;
                 P1 = P1 – 1;
                   end while;
                 AL = n;
                 while AL > 0 do
                                C[P2] = CB[BL] AND CA[AL];
                              P2 = P2 - 1;
                             AL = AL – 1;
                 end while
               if    Mul = = NULL then
                             Mul = C;
            else    Mul    = SHA(Mul, C);
            end if
                        BL = BL -1;
```

end while Step 3: Evaluate (Mul); Output(Mul);
End function

In the method given above, we describe the homomorphic multiplication between a pair of ciphertext with the same length in SHM. The method also applies to a pair of ciphertext with different lengths by modifying the parameters AL and BL denoted the length of ciphertext CA and the length of ciphertext CB respectively.

Here, Step 1 executes some initialization operations which assign values to corresponding variables. Step 2 introduces how to calculate the homomorphic multiplication between a pair of integer ciphertext. Step 2, we use the primitive operations in order to realize the homomorphic addition, including AND operation and SHA function. Step 3 is to remove the noise from the multiplication by calling the Evaluate function. Finally, Step 4 outputs the sum of homomorphic addition.

In the mentioned above three methods, initialization and data partition operations are not reflected in the description of the methods in sequential computing mode. We only illustrate critical steps by pseudo-code in sequential computing mode.

3.3. Proposed Method

In order to parallelize the sequential mode, we should find out the steps that can be paralleled in the mentioned three sequential methods. Generally, computing operation can be paralleled by different execution units. In the three methods, step 2 of each method can be processed by other execution unit paralleling with CPU's other operations. The CPU's other operations include initialization, data partition, data merging, and result outputting. For example, as is shown in Fig. 1, the executing flow of the SHE is analyzed by us in order to find out the bottleneck in the sequential computing mode. Moreover, we find out that except step 2 of the three methods, other steps are difficult to be parallelized. In practice, these three operations belong to computing operation of CPU. In special, when CPU executes computing operation, other steps of the three methods need to be stay waiting state. Furthermore, the execution of computing operation is time-consuming as the computational complexity of the DGHV is high. Therefore, computing operation is the pivotal bottleneck in the sequential computing mode. In fact, one of the reasons for the high time complexity is that the DGHV encrypt data by bits, which leads to the run time is insufferable.

3.3.1. The theoretical support of parallel computing model

As there is noninterference with each other among bit computing, the process of computing operation can be parallelized. The Amdahl's law proves that parallel algorithms can improve the computational efficiency in theory. In special, The Amdahl's law indicates that a parallel algorithm can obtain speedup ratio compared with the sequential version as the following equation nine.

$$S(m) = \frac{m}{1+(m-1)f} \tag{9}$$

where S is the speedup ratio, m is the number of processing units, and f indicates the proportion of sequential parts in the parallel algorithm. This law states that the speedup ratio can be up to $1/f$ when $m \to \infty$.

3.3.2. Overview of the proposed parallel computing mode

In the study of existing parallel computing methods, we found that using GPU-assisted to accelerate execution of algorithm is applicable and popular in many applications from the hardware perspective. Specifically, GPU has evolved into a massively parallel, multithreaded, many-core processor system because of its powerful computing power. In recent years, much work has been done in using CPU-GPU hybrid system to improve execution efficiency of various systems. Some scholars have concentrated on the utilization of GPU to improve efficiency of homomorphic encryption algorithms. However, the amount of processed data is too large, which leads to the CPU throughput drops and affects the parallel execution reduction. Therefore, the number of GPU affects the parallelism of the CPU-GPUs hybrid system which has not been studied up to now. Moreover, CPU-GPUs hybrid systems with different scales also have their corresponding process granularity, which is validated in the following experimental results chapter.

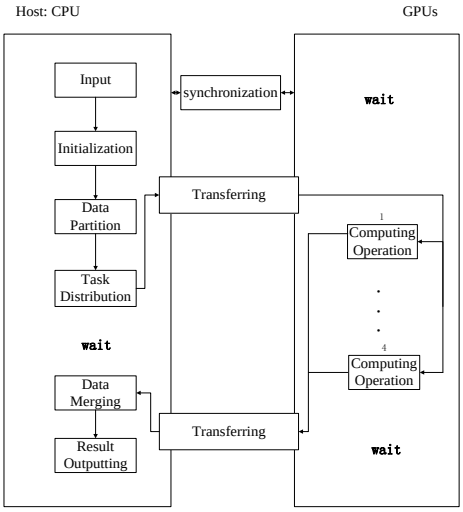

Fig. 4. The executing flow of the parallel computing modes in DGHV based on CPU-GPUs hybrid system.

The executing flow of parallel computing mode in DGHV based on CPU-GPUs hybrid system as shown in the Fig. 4. In the Fig. 4, the computing operation in the right part of the figure represents homomorphic encryption. Moreover, the right part determines the parallelism of the hybrid system. In our study, there are four kinds of parallel computing modes depending on the number of GPU. As is shown in the right part of the Fig. 4, the number of GPU can be selected from 1 to 4, which determines the parallelism. That is, we intend to find out the quantitative relationship between the number of GPU and the corresponding processing granularity. Furthermore, on account of our lab having up to 4 GPUs, our study adopts the CPU-4GPUs accelerated architecture at most. Fig. 4 shows the status of CPU, system bus and GPU in the parallel computing modes. As is shown in Fig. 4, CPU or GPU may need to wait for each other's data. In order to improve the efficiency of the hybrid system, the goal of our study is to make the waiting time shortest even going to zero.

Step 1: Preprocessing

This step mainly includes three subtasks, that is, initialization, data partition, and task distribution. First, the CPU gets the input data which contains many data blocks. In the next step, data partition task mainly splits the data block into smaller data unit for the step of preparing computing operation. In special, the task distribution mainly assign data unit to the GPUs in order to perform computing tasks in parallelism. These three steps are all processed in CPU and host memory. We combine them to a procedure called preprocessing.

Step 2: Transferring and Computing

GPUs can be used by s single host process at the same time. Transferring input data from host to the GPUs and transferring the results back from the GPUs to CPU relay on the system bus. Therefore, we need to consider the GPUs device and the system bus as the critical resources. In detail, while the GPUs execute computing in parallel, the CPU can still perform preprocessing step and the third step. This is the mainly reason for acceleration in our study. Moreover, the GPUs have been stay working state all the time. Also, the results from the GPUs are transferred back to the CPU for the third step.

Step 3: Merging and Outputting

Finally, the CPU will take the tasks that merge results into data blocks as output. These steps will be considered as a procedure called merging and outputting.

3.3.3. Overview of the pipeline framework

In order to minimize the waiting time to zero, we further propose a pipeline framework for CPU-GPUs hybrid system, as shown in Fig. 5, which transforms the above process into three procedures for accelerating DGHV algorithm.

The preprocessing is handled by CPU process unit, which is called Host_Proc1. The transferring and computing procedure is handled by the system bus (with CPU instructions) and GPUs, which is called Host_Proc2_GPUs. The merging and outputting procedure is handled by CPU process unit called Host_Proc3.

In Fig. 5, the granularity of process unit plays the key role in the pipeline framework. In the section of experiments, we will present the influence of different granularities on the throughput and response time based on the CPU-GPUs hybrid system.

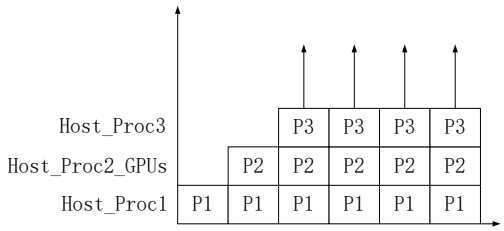

Fig. 5. Pipeline architecture of processing stream in CPU-GPUs hybrid system.

3.3.4. Parallel computing mode of the DGHV

In CPU-GPUs hybrid system, CPU is the core of the system and schedules various resources, including its own computing resources and GPU's computing resources. In theory, when implement a computing task, we can have it execute on the CPU or on the GPUs. In order to determine which device executes the computing operations, CUDA leaves the decision to the programmer by adding the modifier before the function to specify it. Therefore, we can package the step 2 of the above three methods into a function with adding modifier, which represents that these codes in the function executed by GPUs. Here, we define the encryption function named as _globe_ Enc( ), which indicates that this function is performed by GPUs. In the same way, the homomorphic addition function executed by GPUs can be defined as the _globe_ Add( ) function and the homomorphic multiplication can be defined as the _globe_ Mul( ) function. The prefix _globe_ indicates the function is carried out by GPUs.

Given a composite message $M = m_1 m_2 \ldots m_n$, where $n \in N$(the set of natural numbers). Encrypt M over a secret key p can be utilized by Method 4 named Parallel Homomorphic Encryption(PHE) in parallel computing mode, which calls the function _globe_ Enc( ).

Method 4: PHE( )

Function ( )

Input: $M = m_1 m_2 \dots m_n$, $m_i \in \{0,1\}$;

globe Enc(M);

Step1: p = KeyGen($p \in (2Z + 1) \cap [2^{n-1}, 2^n)$);

Step2: Enc<<< M >>>; || executed by GPUs

Step3: $C = Enc(p, M) = Enc(p, m_1)Enc(p, m_2) \dots Enc(p, m_n) = c_1 c_2 \dots c_n$;

Output($C = c_1 c_2 \dots c_n$);

End Function

Given two integer ciphertext with the length of n, where $n \in N$(the set of natural numbers). The parallel homomorphic addition (PHA) calls the _globe_ Add( ) function to implement addition of the two ciphertext.

Method 5: Parallel Homomorphic Addition

Function PHA( )

Input: CA[ ] = $c_{a1} c_{a2} \dots c_{an}$, CB[ ] = $c_{b1} c_{b2} \dots c_{bn}$;

globe Add(CA, CB);

Step 1: N = n; C = Array[N], Sum = Array[N], i = 0;

Step2: Add<<< CA, CB >>>; || executed by GPUs

Step 3: Output Sum;

End Function

Given two integer ciphertext with the length of n, where $n \in N$(the set of natural numbers). The parallel homomorphic multiplication (PHM) calls the _globe_ Mul( ) function to implement multiplication of the two ciphertext.

Method 6: Parallel Homomorphic Multiplication

Function PHM( )

Input: CA[ ] = $c_{a1} c_{a2} \dots c_{an}$, CB[ ] = $c_{b1} c_{b2} \dots c_{bn}$;

globe Mul(CA, CB);

Step 1: N = 2n; Mul = Array[N], AL = n; BL = n;

Step2: Mul<<< CA, CB >>>; || executed by GPUs

Step 3:	Evaluate (Mul);
Step 4:	Output(Mul);
End function	

In the above three parallel computing methods, the step 2 is executed by GPUs because of CUDA provides a convenient way of implementation. The time complexity of the six methods is shown in the following Table 1. Here, we mainly consider the time complexity of the host which represents the CPU's execution efficiency to some extent. A conclusion can be drawn from the analysis of time complexity, the execution efficiency of parallel computing mode in the DGHV is significantly improved compared to the three sequential counterparts.

Table 1. The Time Complexity of the Six Methods

Method	Time Complexity of Host
SHE	O(n)
SHA	O(n)
SHM	$O(n^2)$
PHE	O(1)
PHA	O(1)
PHM	O(1)

4. Experimental Results

4.1. Experimental Setup and Dataset

In our study, Table 2 shows the experimental environment. All the codes are implemented in C++ and CUDA, which is an SDK for NVIDIA GPU, and which is an integrated complier for C++ and CUDA.

Table 2. Experiment Environment

Item	Specification
CPU	Intel Core i5-5200U
#of CPU Cores	2
#of Threads	4
CPU Frequency	2.20 GHz
System Memory	12 GB RAM
Operating System	Windows 10
The maximum number of GPUs	4
GPU	NVIDIA Tesla C2050s

#CUDA Cores	448
GPU Core Frequency	1.15GHz
GPU Memory	3GB

In this experiment, we use a data set containing data block with size of 2048MiB. In detail, we have generated plaintexts as a series of random integers for experiment and used encrypted 64-bit representation for an integer. Thus, our data sets ranged from 256 MiB to 2048 MiB of plaintexts. Two kinds of sequential and the parallel tests are involved in our experiment. In sequential computing mode, encryption is directly conducted on the plaintexts without the need of data partition. On the contrary, the data needs to be partitioned into many data blocks in the parallel computing mode. We evaluate our method in performance and compare the parallel computing modes with the counterpart.

4.2. Granularity of Processing Unit in Different Parallelism

In our study, the granularity of process unit indicates the maximum number of data unit in different parallelism. This is a major factor that influences the performance of the proposed method. As it determines the time cost by the transferring and data encryption procedure, it will bring a great impact on whether the processing stream can flow smoothly or not at the certain parallelism. There are four parallelisms in our study, which respectively corresponding to CPU-GPU system, CPU-2GPUs system, CPU-3GPUs system, and CPU-4GPUs system. The four parallelisms are represented by PHE-1, PHE-2, PHE-3, and PHE-4 respectively. For instance, the PHE-1 represents using one GPU to accelerate homomorphic encryption in hybrid system. The other three represents the corresponding meanings. In fact, the number of GPU in hybrid system determines the parallelism. Generally, more GPUs used in CPU-GPU hybrid system means higher parallelism. We utilized encrypted 64-bit representation for an integer in this experiment to represent data unit for process unit. In specific, we confirm the granularities with different parallelism by analyzing the throughput and the response time of hybrid system. In order to avoid the throughput influenced by input queue, we further kept the queue of waiting tasks full, which contained the tasks to be processed.

Fig. 6 and Fig. 7 show the throughput and response time of the CPU-GPU hybrid system respectively with different number of data units. It has been numerically shown that the throughput of the system is low when the number of data unit is small. This is because resources cannot be effectively utilized in the hybrid system, and the most of time is wasted on waiting and communications. With the increase of the number of data units, the throughput rises. It means that, with increase of the number of data unit, the GPU resources can be utilized more effectively. However, we can see the throughput dropping down when the number of data unit is more than 55. The reason is that, when the number of data units goes beyond the optimum value, a large-size process unit will cost more time in GPU, while the host must spend more time on waiting. Fig. 7 shows the response time of the hybrid system. It has been numerically shown that the response time increases with the growth of the number of data unit. By analyzing, the reason is that the larger data size cost more time in the transferring and data encryption procedure. Nevertheless, the slope of the curve becomes larger and tends to be constant. This is because when the number of data unit is small, time is mostly consumed in data transferring step, and also when the number becomes large enough, response time will have an approximate linear increase with the data size. In addition, the response time of data encryption step that has linear relationship with the data size which will account for the majority of the response time.

In detail, compared with sequential computing mode, there are many steps added in parallel computing mode, including data distribution and transmission, interaction within and between the processing units.

These additional parts will affect the efficiency of the system, especially when the system is under low workload. Therefore, relatively large size data need to be utilized in the experiment.

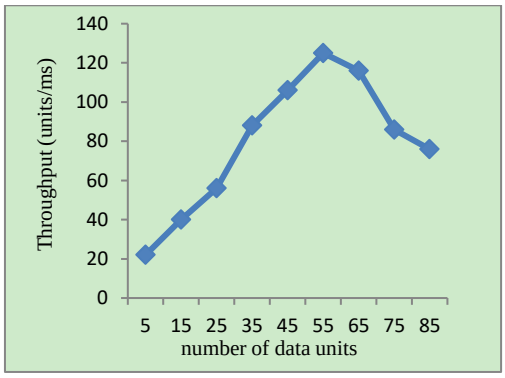

Fig. 6. Throughput under different load levels in PHE-1.

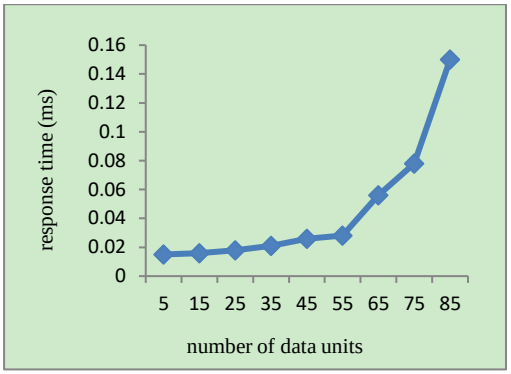

Fig. 7. Response time under different load levels in PHE-1.

Granularity of process unit is also called window size in hybrid system. By regulating the sliding window size, finally we adopt 55 as the granularity of process unit in CPU-GPU hybrid system. Because our main purpose is to improve the throughput of system and the throughput achieves maximum while the response time is also preferable. As a compromise solution, we choose the case of window size w=55 for our implementation, which is referred as granularity of process unit in CPU-GPU hybrid system.

We adopt the same analysis method as PHE-4 applying in the PHE-1, PHE-2, and PHE-3, which confirms the granularity of the other three parallel computing mode. Specifically, we adopt 106, 136, 198 as the granularities of process unit in PHE-2, PHE-3, and PHE-4 respectively. It can be concluded that different parallelism has different granularity. Therefore, we should first confirm the granularity of the system in practical parallel circumstance. The specific experimental data are shown in the following 3 figures.

4.3. Evaluation of Proposed Method

In order to evaluate how well the proposed method has practical application significance, we apply the run time as an indicator of evaluation and quantitatively confirming the improvement of data encryption, homomorphic addition, and homomorphic multiplication. In detail, we compare SHE with PHE-4 based on the run time of system. The aim of the experiment is not to determine which parallel mode is optimal. This is because the PHE-4 is definitely the most efficient parallel computing mode, which is also named as FPHE mentioned above. Therefore, we utilize FPHE to execute experiment. Fig. 8 indicates the encryption time of data with different size under the sequential and the parallel mode. Fig. 9 indicates the homomorphic

addition time of data with different size under the sequential and the parallel mode. Fig. 10 indicates the homomorphic multiplication time of data with different size under the sequential and the parallel mode.

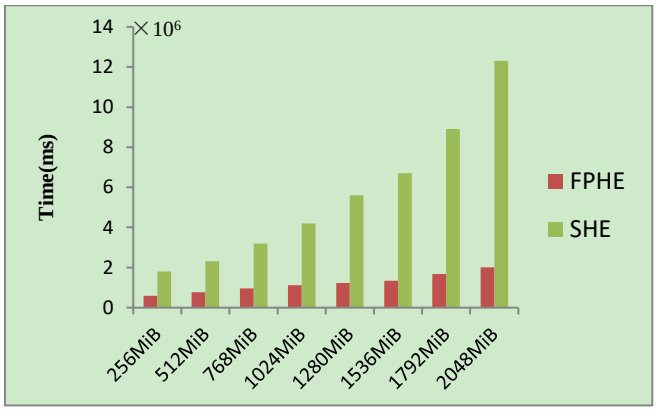

Fig. 8. The contrast diagram of data encryption.

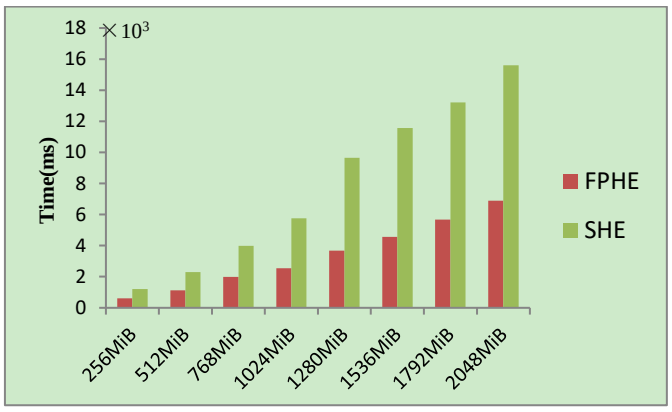

Fig. 9. The contrast diagram of homomorphic addition.

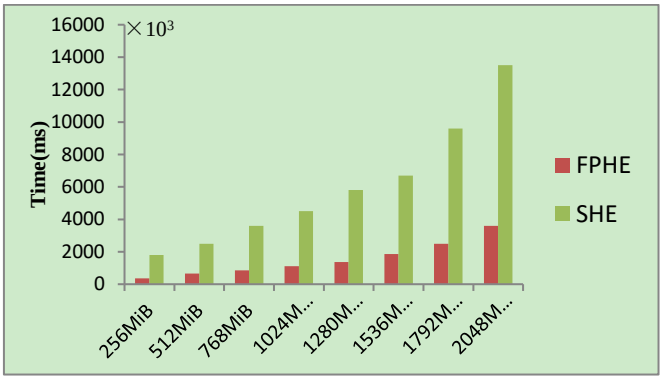

Fig. 10. The contrast diagram of homomorphic multiplication.

The following conclusions can be drawn from the experimental results shown in the above six figures.

(1) The time of the above three operations increase along with the increase of the data size in the parallel modes.

(2) The parallel mode can improve efficiency of system compared to the sequential mode. In special, FPHE provides more than 84% improvement (run time) over the SHE with encryption operation, and provides more than 67% improvement (run time) over the SHE with homomorphic addition, and provides more than 80% improvement (run time) over the SHE with homomorphic multiplication.

(3) Overall, the above experimental results indicate the fact that the CPU and the GPU are actually performing computing tasks concurrently in our proposed method. The run time of the FPHE has been parallelized using the 4GPUs is greatly shorter than the SHE.

5. Conclusions and Discussions

In this research, we proposed to use hybrid CPU-4GPUs as a parallel coprocessor to deal with the DGHV homomorphic encryption in which the CPU is responsible for the management, scheduling, distributing, merging, and input/output tasks and the GPU is responsible for encryption. We further adopt the pipeline architecture of processing stream in CPU-4GPUs hybrid system to accelerate the method in our study. In order to ensure the pipeline architecture really take effect, we need to find out the granularity of a certain parallel mode. Moreover, we also validate that different parallelism has its own corresponding granularity of processing unit.

The experimental results showed that our proposed method named FPHE can obtain higher efficiency on high workload level compared to the SHE. However, for further improving the performance of the proposed method in this paper, there are still several problems to further address. To further illustrate the effectiveness of the proposed method, decryption and evaluation algorithms of DGHV will be implemented on the hybrid CPU-4GPUs system.

Acknowledgments

The work is supported by National Science Foundation of China under the Grant No. 61502438.

References

[1] Rivest, R. L., Adleman, L., & Dertouzos, M. L. (1978). On data banks and privacy homeomorphisms. *Foundations of Secure Computation* (pp. 169-177).

[2] Gentry, C. (2009). *A Fully Homomorphic Encryption Scheme*. Stanford University Press.

[3] Gentry, C. (2009). Fully homomrphic encryption using ideal lattices. *ACM STOC 2009* (pp. 169-178).

[4] Dijk, M. V., Gentry, C., & Halevi, S. (2010). Fully homomorphic encryption over the integers. *Proceedings of International Conference on Theory and Applications of Cryptographic Techniques, Springer-Verlag* (pp. 24-43).

[5] Wu, Z., & Li, T. (2017). An improved fully homomorphic encryption scheme under the cloud environment. *Chinese Conference on Computer Supported Cooperative Work and Social Computing, ACM* (pp. 251-252).

[6] Hendrickson, B. (2013). Emerging challenges and opportunities in parallel computing: The cretaceous redux? *Health Policy & Planning*, *28(8),* 809-824.

[7] Buck, I., Foley, T., & Horn, D. (2004). Brook for GPUs: Stream computing on graphics hardware. *ACM SIGGRAPH, ACM* (pp. 777-786).

[8] Hendre, A., & Joshi, K. P. (2015). A semantic approach to cloud security and compliance. *Proceedings of International Conference on Cloud Computing* (pp. 1081-1084).

[9] Fugkeaw, S., & Sato, H. (2016). Privacy-preserving access control model for big data cloud. *Computer Science and Engineering Conference*, *IEEE* (pp. 1-6).

[10] Swathi, R., & Subha, T. (2017). Enhancing data storage security in cloud using certificateless public auditing. *Proceedings of International Conference on Computing and Communications Technologies* (pp. 348-352).

[11] Jiang, W., Zhao, Z., & Laat, C. D. (2013). An autonomous security storage solution for data-intensive cooperative cloud computing. *Proceedings of International Conference on E-Science, IEEE Computer*

Society (pp. 369-372).

[12] Shu, J., Jia, X., & Yang K. (2018). Privacy-preserving task recommendation services for crowdsourcing. *IEEE Transactions on Services Computing* (pp. 1-1).

[13] Zhang, J., Li, H., & Liu, X. (2017). On efficient and robust anonymization for privacy protection on massive streaming categorical information. *IEEE Transactions on Dependable & Secure Computing, 14(5)*, 507-520.

[14] Yang, Y. (2011). Towards multi-user private keyword search for cloud computing. *Proceedings of International Conference on Cloud Computing, IEEE Computer Society* (pp. 758-759).

[15] Esposito, C., Castiglione, A., & Choo, K. K. R. (2016). Encryption-based solution for data sovereignty in federated clouds. *IEEE Cloud Computing, 3(1),* 12-17.

[16] Li, J., Chen, S., & Song, D. (2013). Security structure of cloud storage based on homomorphic encryption scheme. *Proceedings of International Conference on Cloud Computing and Intelligent Systems* (pp. 224-227).

[17] Zhang, J., Yang, Y., & Chen, Y. (2017). A general framework to design secure cloud storage protocol using homomorphic encryption scheme. *Computer Networks,* 37-50.

[18] Rangasami, K., & Vagdevi, S. (2017). Comparative study of homomorphic encryption methods for secured data operations in cloud computing. *Proceedings of 2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT)* (pp. 1-6).

[19] Song, X. D., & Wang, Y. L. (2017). Homomorphic cloud computing scheme based on hybrid homomorphic encryption. *Proceedings of 2017 3rd IEEE International Conference on Computer and Communications (ICCC)* (pp. 2450-2453).

[20] Jayapandian, N., & Rahman, A. M. J. M. Z. (2017). Secure and efficient online data storage and sharing over cloud environment using probabilistic with homomorphic encryption. *Cluster Computing*, 1561-1573.

[21] Moayedfard, M., & Molahosseini, A. S. (2016). Parallel implementations of somewhat homomorphic encryption based on Open-MP and CUDA. *International Congress on Technology, Communication and Knowledge, IEEE* (pp. 186-190).

[22] Xia, J., Ma, Z., & Dai X. F. (2018). Fast homomorphic encryption based on CPU-4GPUs hybrid system in cloud. *Web Information Systems and Application* (pp. 79-90).

[23] Hayward, R., & Chiang, C. (2013). An architecture for parallelizing fully homomorphic cryptography on cloud. *Proceedings of Seventh International Conference on Complex, Intelligent, and Software Intensive Systems. IEEE* (pp. 72-77).

[24] Sethi, K., Majumdar, A., & Bera, P. (2017). A novel implementation of parallel homomorphic encryption for secure data storage in cloud. *Proceedings of 2017 International Conference on Cyber Security And Protection Of Digital Services (Cyber Security). IEEE Press, San Francisco* (pp. 1-7).

[25] Min, Z., Yang, G., & Shi, J. (2017). A privacy-preserving parallel and homomorphic encryption scheme. *Open Physics. De Gruyter Open,* 135-142.

[26] Khedr, A., & Gulak, G. (2017). SecureMed: Secure medical computation using GPU-accelerated homomorphic encryption scheme. *IEEE Journal of Biomedical and Health Informatics. IEEE Press*, 597-606.

[27] Tian, Y., Al-Rodhaan, M., & Song, B. (2015) Somewhat homomorphic cryptography for matrix multiplication using GPU acceleration. *Proceedings of 2014 International Symposium on Biometrics and Security Technologies (ISBAST), IEEE Press, Kuala Lumpur* (pp. 166-170).

[28] Moayedfard, M., & Molahosseini, A. S. (2015). Parallel implementations of somewhat homomorphic encryption based on open-MP and CUDA. *Proceedings of 2015 International Congress on Technology,*

Communication and Knowledge (ICTCK). IEEE Press, Mashhad (pp. 186-190).

[29] Wang, W., Hu, Y., & Chen L. (2012). Accelerating fully homomorphic encryption using GPU. *Proceedings of 2012 IEEE Conference on High Performance Extreme Computing, IEEE Press, Waltham* (pp. 1-5).

Jing Xia was born in 1982. She received her master's degree in computer application from Huazhong University of Science and Technology in 2011. She is currently studying for Ph.D at Wuhan Digital Engineering Institute. Her main research interests include information security, machine learning, cloud computing, and web search engine.

Zhong Ma was born in 1962, who is a PhD, professor and PhD supervisor. He is the president of Wuhan Digital Engineering Institute. His main research interests include high performance computing, cloud computing, information security, machine learning, and data mining.

Xinfa Dai was born in 1974, who is a PhD, associate professor and master's tutor. His main research interests include web mining, cloud computing security, and machine learning.