Cybersecurity

SURVEY **Open Access**

Practical solutions in fully homomorphic encryption: a survey analyzing existing acceleration methods

Yanwei Gong[1], Xiaolin Chang[1*], Jelena Mišić[2], Vojislav B. Mišić[2], Jianhua Wang[1] and Haoran Zhu[1]

Abstract
Fully homomorphic encryption (FHE) has experienced significant development and continuous breakthroughs in theory, enabling its widespread application in various fields, like outsourcing computation and secure multi-party computing, in order to preserve privacy. Nonetheless, the application of FHE is constrained by its substantial computing overhead and storage cost. Researchers have proposed practical acceleration solutions to address these issues. This paper aims to provide a comprehensive survey for systematically comparing and analyzing the strengths and weaknesses of FHE acceleration schemes, which is currently lacking in the literature. The relevant researches conducted between 2019 and 2022 are investigated. We first provide a comprehensive summary of the latest research findings on accelerating FHE, aiming to offer valuable insights for researchers interested in FHE acceleration. Secondly, we classify existing acceleration schemes from algorithmic and hardware perspectives. We also propose evaluation metrics and conduct a detailed comparison of various methods. Finally, our study presents the future research directions of FHE acceleration, and also offers both guidance and support for practical application and theoretical research in this field.

Keywords Acceleration, Bootstrapping, FPGA, Fully homomorphic encryption, GPU, NTT

Introduction

The exponential-growing volume of data and the rapid development of cloud computing facilitate outsourced computation of big data (Li et al. 2022; Hanafizadeh 2020). However, the collected data contains a large amount of sensitive and private information, which can lead to privacy disclosure. There exist cryptographic technologies to safeguard privacy, among which is fully homomorphic encryption (FHE). FHE is being explored for protecting data privacy and then is applied to many application scenarios, especially those involving sensitive data, such as healthcare, finance, and government (Bos et al. 2014; Jiayi et al. 2020 Deviani 2022). It plays an important role in the field of privacy protection (Dijk 2020; Torres et al. 2014).

The concept of FHE was first proposed by Rivest et al. (1978) in 1978, but the first FHE scheme was proposed by Gentry (Craig Gentry: A fully homomorphic encryption scheme[M]. 2009) in 2009. He also proposed a method for constructing an FHE scheme, which means that a somewhat homomorphic encryption (SWHE) scheme can become an FHE scheme by using bootstrapping to add the noise refresh process. Based on this, researchers carried out a lot of researches on various methods of constructing an FHE scheme. The most representative schemes are BGV (Brakerski and Gentry 2014), FV (Fan and Vercauteren 2012), GSW (Gentry and Sahai 2013), and CKKS (Homomorphic Encryption for Arithmetic of Approximate Numbers 2017). In fact, FHE is just one

*Correspondence:
Xiaolin Chang
xlchang@bjtu.edu.cn
[1] Beijing Key Laboratory of Security and Privacy in Intelligent Transportation, Beijing Jiaotong University, Beijing, China
[2] Ryerson University, Toronto, ON, Canada

of homomorphic encryption (HE), which also includes partially homomorphic encryption (PHE) (Rivest et al. 1978; Gamal 1985; Paillier 1999) and (SWHE) (Andrew Chi-Chih Yao 1982; Sander et al. 1999; Boneh et al. 2005). FHE allows infinite calculation and supports both homomorphic addition (HAdd) and homomorphic multiplication (HMult) on the ciphertext. The key strength of FHE is that it offers cryptographically-strong privacy guarantees, but these guarantees come at the cost of massive computational overhead. Thus, many researchers have shifted their study attention to FHE acceleration. Although the FHE acceleration schemes have made good progress, there is still a lack of a summary of related works to support further development.

There are surveys about FHE. Moore et al. (2014) gave a brief introduction to the existing FHE acceleration schemes, which only include those based on the graphics processing unit (GPU) and field programmable gate array (FPGA). They did not make an analysis about them, and there was no discussion of CKKS acceleration scheme because it was not proposed at that time. Acar et al. (2018) introduced the development history of HE and concluded the theories and details of various typical algorithms of HE. At the same time, this paper also sorted out and compared the implementation libraries of some HE algorithms. Alaya et al. (2020) summarized different HE application ways and provide the application scenarios, such as medical treatment, image, and other fields. Wood et al. (2021) provided an overview of the application of FHE in medicine and bioinformatics, along with descriptions of how it can be realized by considering their different characteristics. Zhang et al. (2022) took the first initiative to conduct a systematic study on the 14 FHE accelerators. They established a qualitative connection between different accelerators and performed testbed evaluations of representative open-source FHE accelerators to provide a quantitative comparison on them. But, similar to Moore et al. (2014), the summary of FHE acceleration scheme is insufficient. Marcolla et al. (2022) introduced the basic knowledge and security attributes of HE and further summarized the application scenarios of HE, such as machine learning, fog computing, and cloud computing. Moreover, it also introduced some libraries and tools for HE. However, these reviews neither referred to the FHE acceleration schemes nor provided a comprehensive summary of FHE acceleration schemes.

To bridge this gap, we collate existing studies and provide a comprehensive review of FHE acceleration schemes. In addition, we summarize and compare the related work in Table 1, which fully demonstrates the necessity of this survey. It is because no other survey has collated content similar to it.

The main limitation of the application of FHE is its performance bottleneck, which means the huge cost of FHE can not satisfy the demand for practical application. As a result, a large number of studies began to study how to accelerate it. At present, there mainly exist two ways. On the one hand, it focuses on algorithm optimization to accelerate the FHE scheme itself. On the other hand, it uses hardware to accelerate it, such as central processing unit (CPU), GPU, FPGA, etc. Figure 1 displays the distribution of papers focusing on FHE acceleration across

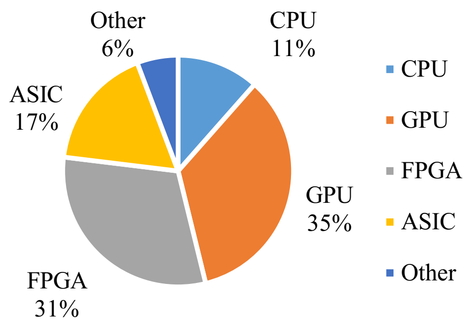

Fig. 1 The distribution of papers focusing on FHE acceleration across various hardware platforms during the last four years

Table 1 Comparison of related works

References	Year	Theory summary	Scheme summary	Application summary	Acceleration summary	
					Algorithm optimization	Hardware optimization
Moore et al. (2014)	2014					Insufficient
Acar and Hidayet Aksu (2018)	2018	√	√			
Alaya et al. (2020)	2020			√		
Wood and Najarian (2021)	2020			√		
Zhang et al. (2212)	2022					Insufficient
Marcolla et al. (2022)	2022	√		√		
Ours	2023				√	√

The '√' indicates that the work satisfies the attribute represented in this column. The same is true for the meaning of '√' in the other tables presented in this paper various hardware platforms during the last four years. Notably, a significant portion of the papers center on FHE acceleration via GPU and FPGA. It is speculated that while GPU may not represent the optimal hardware platform for FHE acceleration, GPU can be used relatively simply to accelerate FHE. Therefore, in the early stages of FHE acceleration research, it was more common for researchers to utilize GPU and achieve superior acceleration effects compared to using CPU. Figure 2 highlights the trend of papers on FHE acceleration based on different hardware platforms in recent years, thereby lending further support to the aforementioned speculation. The earlier stages of FHE acceleration research featured a greater emphasis on acceleration schemes based on GPU. However, later FPGA and ASIC were found to be more suitable hardware for FHE acceleration in terms of acceleration efficiency. This has resulted in an increasing number of papers on FHE acceleration based on them year after year.

We review and summarize the current mainstream FHE acceleration schemes proposed in the last four years, with the following contributions:

- In this paper, we present a comprehensive summary and synthesis of the latest research findings on accelerating FHE between 2019 and 2022. The primary objective is to provide valuable insights for researchers interested in the state-of-the-art developments and future directions of FHE acceleration.
- Our study provides a comprehensive classification of existing acceleration methods for FHE from two perspectives: algorithmic acceleration and hardware acceleration. Further classification is conducted based on these two main categories, which is detailed in Sects. "Algorithm acceleration schemes" and "Hardware acceleration schemes". Finally, we propose corresponding evaluation metrics to conduct a detailed comparison of various acceleration methods, aiming to offer guidance and inspiration for future research in this field. In particular, algorithmic-based methods achieve optimization by reducing the number of operations required for encryption, decryption, and homomorphic operations. Meanwhile, hardware-based methods focus on designing specialized hardware that can perform FHE operations more efficiently. Through this comprehensive comparison, we aim to provide a deeper understanding of the strengths and weaknesses of different FHE acceleration schemes.
- This paper explores the future research directions of FHE acceleration and provides guidance and support for practical application and theoretical research in this field. We identify various potential research directions, such as exploring new FHE algorithms, designing novel hardware architectures, and investigating hybrid acceleration schemes that combine algorithmic and hardware-based methods. By highlighting these potential research directions, we aim to encourage further advancements in FHE acceleration and its application to various fields for privacy preservation.

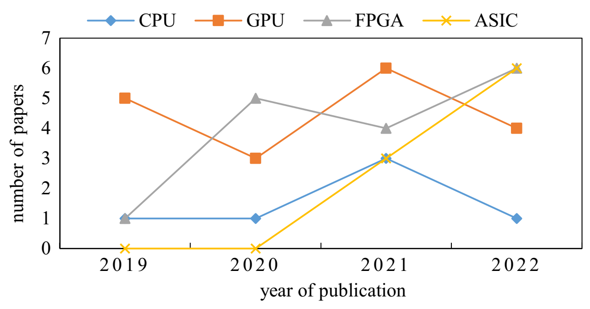

Fig. 2 The trend of papers on FHE acceleration based on various hardware from 2019 to 2022

The rest of this paper is organized as follows. Sect. "Preliminary" introduces the basic knowledge of FHE acceleration schemes, including the knowledge related to algorithms and hardware. Sects. "Algorithm acceleration schemes" and "Hardware acceleration schemes" respectively present the algorithm-based and hardware-based acceleration schemes. Following this, Sect. "Challenges and future research directions" gives the challenges and future research directions of FHE acceleration. Finally, Sect. "Conclusion" provides the conclusion.

Preliminary

This section introduces the basic knowledge of FHE acceleration, including algorithms and hardware, in order to better understand FHE acceleration schemes discussed in Sects. "Algorithm acceleration schemes" and "Hardware acceleration schemes". Sect. "Algorithms about FHE" focuses on those operations related to FHE performance bottlenecks. Sect. "Hardware platforms about FHE" focuses on hardware platforms exploited to accelerate FHE schemes, including CPU, GPU, FPGA, and application specific integrated circuit (ASIC).

Algorithms about FHE

This section describes operations related to the performance bottlenecks of FHE, including polynomial addition, polynomial multiplication, number theoretic transform (NTT), and bootstrapping.

(1) Polynomial multiplication

Polynomial multiplication is an important homomorphic operation and the multiplication of very large degree polynomials is one of the major performance bottlenecks for the FHE implementations. NTT can be explored for FHE acceleration, which is done by first converting inputs to the NTT domain. In the NTT domain, the polynomial operation can be converted to a coefficient-wise multiplication, which is also called dot multiplication. The coefficient-wise multiplication has high parallelism and is very suitable for executing on the hardware platform, which has a lot of parallel computing resources. The overhead of coefficient-wise multiplication is negligible when being compared to NTT and inverse NTT (INTT). They respectively represent the transformation of the polynomial multiplication input into the NTT domain and the reversal of the dot multiplication result, the overhead of coefficient-wise multiplication.

(2) Polynomial addition

Like polynomial multiplication, polynomial addition is another important homomorphic operation in the homomorphic evaluation and is the second most frequently used operation after polynomial multiplication. Besides, the accumulation part of relinearization also needs polynomial addition. The polynomial addition can be carried out in the Chinese remainder theorem (CRT) domain, which provides sufficient parallelism so that the hardware with parallel computing resources can be used to accelerate it.

(3) NTT

NTT is extensively employed for FHE implementation because it enables fast polynomial multiplication by reducing the complexity of polynomial multiplication from $O(n^2)$ to $O(nlogn)$. NTT is defined as the discrete fourier transform over $\mathbb{Z}_q$. An N-point NTT operation transforms an n element vector $\boldsymbol{a}$ to another an n element vector $\widetilde{\boldsymbol{a}}$. And the NTT can be naturally used for fast cyclic convolution.

(4) Bootstrapping algorithm

Bootstrapping algorithm is used to refresh the noise of ciphertext after a homomorphic evaluation when the entire noise budget of a ciphertext is consumed. It reduces the noise back to a lower level by running decryption homomorphically in order to allow an infinite number of computations of ciphertext. This is also the difference between FHE and SWHE. The bootstrapping operation consists of three major steps, including a linear transform, a polynomial evaluation, and another linear transform. All these steps consist of the same homomorphic operations, such as HAdd, HMult, and rotation.

Hardware platforms about FHE

Sect. "Hardware platforms about FHE" describes the hardware used to accelerate FHE schemes. It analyses the characteristics of hardware in order to determine the optimal hardware platform for accelerating FHE schemes.

(1) CPU

Compared with GPU, FPGA, and other hardware platforms, the acceleration effect achieved by CPU is not obvious. CPU itself is not designed to perform heavy data computing tasks. It can provide a degree of parallel computing power through multithreading, but this is far from enough to meet the needs of FHE acceleration. Therefore, most studies prefer to combine it with GPU or FPGA to realize the acceleration of FHE. However, CPU has the advantage of being a more general hardware platform with a wide range of applications.

(2) GPU

The customization flexibility that GPU can provide is between CPU and FPGA. General-purpose computing on GPU yields greater efficiency when standardized by price compared to FPGA and ASIC. GPU is a powerful but highly specialized device that requires careful coding to take full advantage of the large amount of parallelism it offers. Specifically, the programming model and memory organization are quite different from the CPU. Compared with FPGA, it is easier to develop programs. In addition, there is a compute unified device architecture (CUDA) toolkit to facilitate program development.

(3) FPGA

FPGA is a chip that can be reconfigurable circuitry. It is a hardware reconfigurable architecture, so it can provide more customization flexibility. Both CPU and GPU belong to the von Neumann structure, while FPGA is a no-instruction and no shared memory architecture. This structure makes FPGA much more energy efficient than CPU or even GPU. The function of each logic unit in an FPGA is determined during reprogramming, without the need for instructions. FPGA has both pipeline and data parallelism, whereas GPU only has data parallelism (pipeline depth is limited). Due to the flexibility of FPGA, more researchers focus on the acceleration of FHE using FPGA.

(4) ASIC

ASIC is capable of achieving maximum acceleration effects concerning FHE, while it requires customization, and therefore their application scope is limited. The biggest advantage of ASIC is that it can be designed corresponding circuits completely according to the needs of computing tasks, thus providing maximum acceleration capability. However, ASIC needs to go through many processes from being designed to being put into use, and it is also costly. So its application in FHE acceleration is more in the simulation stage. But the idea of using ASIC to accelerate FHE can be borrowed and applied to other hardware platforms.

Each FHE accelerator has its own strengths and weakness. CPU is the most common computer processor and can be used to implement FHE schemes. However, due to its single-instruction pipeline architecture, its parallel performance and processing speed are relatively low, making it less suitable for high-performance homomorphic operations. GPU has excellent parallel computing capabilities when processing large-scale data, so they can be used to accelerate FHE schemes. However, since GPU is designed for graphics processing, its support for numerical computing is limited, and sometimes it is difficult to adapt to the special requirements of FHE schemes. FPGA is a programmable logic device with high flexibility and parallel performance, and can be used to implement high-performance FHE schemes. Since FPGA is programmable, it can be customized and optimized according to specific needs and application scenarios, thereby achieving higher performance and efficiency. ASIC is a chip specially designed and manufactured to implement highly customized and optimized FHE schemes. Compared with FPGA, ASIC has higher performance and lower power consumption, but also has higher design costs and longer development cycles.

In summary, the application scenario and specific requirements must be considered for selecting an accelerator in order to achieve the optimal performance and efficiency.

Table 2 compares the hardware platforms from 6 aspects. *Universality* refers to whether the hardware platform is widely used, and it is persuasive to say that CPU is the most widely used hardware platform. Whether or not a hardware platform provides enough customization can be evaluated through customization flexibility, which also reflects the acceleration ability it can offer. ASIC is considered to be able to provide the largest customization flexibility because it allows you to design hardware circuits. As far as practicality is concerned, there are two aspects to consider: ease of use and acceleration ability. In short, if a hardware platform provides good acceleration but with low usability, it would be regarded as having low practicality, such as ASIC. The metric of research popularity, the same as in Table 3, is determined according to the number of related papers.

Table 2 Comparison of hardware platforms

Metric	Hardware platform			
	CPU	GPU	FPGA	ASIC
Universality	+ + + + +	+ + + +	+ + +	+ +
Customization flexibility	+	+ + +	+ + + +	+ + + + +
Acceleration ability	+	+ + +	+ + + +	+ + + + +
Price	+ + + +	+ + +	+ +	+ + + + +
Practicality	+	+ + + +	+ + + +	+ + +
Research popularity	+	+ + + +	+ + + +	+ + +

Note that We use the number of '+' to describe the strength of hardware with respect to a feature, with five '+' representing the strongest and one '+' representing the weakest. The same is true for the meaning of '+' in the other tables presented in this paper

Algorithm acceleration schemes

This section introduces the acceleration schemes of FHE using algorithm optimization, which is mainly divided into three categories, including NTT, Bootstrapping, and Encoding. Table 3 compares their characteristics of them, including acceleration approaches, importance, acceleration effect, and research popularity. It can be seen that NTT and bootstrapping are studied by more researchers. Table 4 gives all the acceleration schemes discussed in Sects. "Algorithm acceleration schemes" and "Hardware acceleration schemes, and also shows whether they are algorithm-based.

NTT optimization

As a very important primitive operation in FHE, NTT has very important research value for FHE acceleration. Therefore, a large amount of research works focus on the design of acceleration schemes for NTT, including algorithm optimization and hardware optimization. The core idea of NTT algorithm optimization is to use the existing algorithm to simplify the NTT operation and replace it with a more suitable form of hardware parallel computing.

Rashmi et al. (2020) accelerate FHE by using algorithmic optimization. The most important part of this paper in terms of algorithmic optimization is to use low-cost operations to replace high-cost operations. Barrett

Table 3 Comparison of acceleration object

Accelerated object	Acceleration approaches	Importance	Research popularity	Accelerated effect
NTT	Operational substitution exploration	+ + + + +	+ + + + +	+ + + +
	Algorithm parallelism exploration			
Bootstrapping	Data dependency exploration	+ + + + +	+ + +	+ + +
	Algorithm simplification exploration			
Encoding	Efficient coding based on the data access mode of the subsequent algorithm	+	+	+

reduction (Barrett 1986) method is adopted to transform the high-consumption modular operations. In addition, the NTT operation in polynomial multiplication is used to perform the indices computation operation by shifting and XOR operation, so as to accelerate the NTT process. Similar to Rashmi (2020), Türkoglu et al. (2022) also use Barrett Reduction to accelerate the implementation of the algorithm.

Compared to Rashmi (2020), not only do Shivdikar et al. (2022) use Barrett reduction, but they also improve it. Based on several variants of Barrett reduction, an efficient Barrett reduction for 64-bit integers is proposed to accelerate the modular division of NTT operations used in polynomial multiplication operations. Mert et al. (2020) accelerate the encryption and decryption process of BFV algorithm. Unlike (Shivdikar et al. 2022; Rashmi 2020), an efficient polynomial multiplier is proposed which can also be used for homomorphic operations other than encryption and decryption. They mainly use Montgomery algorithm to reduce the modular division operation in polynomial multiplication, so as to achieve the algorithm acceleration. Goey et al. (2021) not only use Barrett reduction but also use SSMA algorithm (Schönhage and Strassen 1971) to further accelerate NTT operation. SSMA is a fast multiplication algorithm for large integers with a low computational complexity of $O(n\log(n)\log(n\log(n)))$. Therefore, in terms of acceleration results, cuHE (Dai and Sunar 2015) is surpassed. Roy et al. (2019) accelerate the BFV. But they only use the existing latest algorithm level optimization method.

The algorithm acceleration for NTT mainly focuses on utilizing Barrett reduction. Some studies (Türkoglu et al. 2022; Rashmi 2020) just use it, while others (Shivdikar et al. 2022) improve it. Moreover, some studies (Goey et al. 2021; Mert and Öztürk 2020) also use other fast multiplication algorithms to accelerate NTT.

Bootstrapping optimization

Bootstrapping, which has huge time complexity and space complexity, is another important performance bottleneck in FHE. Therefore, the core idea of bootstrapping optimization is to solve the memory bandwidth problem in its implementation process and improve throughput.

Chen et al. (2019) focus on bootstrapping (Chillotti et al. 2017) acceleration of CKKS. They use a dynamic programming approach (Halevi and Shoup 2014) to decide the optimal level collapsing strategy for a generic multi-leveled linear transform in order to fully explore the trade-off between levels consumption and the number of operations. And the result shows a large increase in the bootstrapping throughput. In addition, they replace the Taylor approximation with the Chebyshev interpolant to approximate the scaled sine function, which not only consumes fewer levels but also is more accurate than the original method.

For the full-residue number system (full-RNS) variant of CKKS, Han et al. (2020) combine the RNS-decomposition method (Bajard et al. 2016) and the temporary modulus technique (Gentry and Halevi 2012) to reduce about half complexity for HMult even with a larger security parameter. For the evaluation of *sine* function and *cosine* function, they consider a ratio between the size of a message and the size of a ciphertext modulus. As a result, the number of non-scalar multiplications is almost reduced by half compared to the previous work (Chen and Chillotti 2019).

Bossuat et al. (2021) accelerate the bootstrapping for the full-RNS variant of the CKKS. They propose a new format for rotation keys and a modified key-switching procedure in order to improve the baby-step giant-step algorithm (Halevi and Shoup 2021), which is used by previous works (Chen and Chillotti 2019; Han and Ki 2020; Bossuat and Troncoso-Pastoriza 2022). The modified key-switching procedure extends the hoisting (Halevi 2018) technique to a second layer and reduces the cost of the linear transformations compared to the previous hoisting approach. Moreover, they also discuss the parametrization of the CKKS and its bootstrapping circuit and propose a procedure to choose and fine-tune the parameters for a given use-case.

In (Castro et al. 2021), the optimization method of bootstrapping is proposed aiming at improving the

Table 4 Comparison of different acceleration schemes

References	Year	Algorithm-based optimization			Hardware-based optimization				
		NTT	Bootstrapping	Encoding	CPU	GPU	FPGA	ASIC	Other
Chen and Chillotti (2019)	2019		√		√				
Han and Ki (2020)	2020		√		√				
Bossuat et al. (2021)	2021		√		√				
Boemer et al. (2021)	2021				√				
Ishimaki (2021)	2021				√				
Inoue and Suzuki (2022)	2022				√				
Jin et al. (2019)	2019			√		√			
Ahmad Al Badawi (2019)	2019					√			
Lupascu (2019)	2019					√			
Lei et al. (2019)	2019					√			
Xia et al. (2019)	2019					√			
Kim and Jung (2020)	2020					√			
Morshed (2020)	2020					√			
Ahmad Al Badawi (2020)	2020					√			
Ahmad Al Badawi (2021)	2021					√			
Pedro et al. (2021)	2021					√			
Goey et al. (2021)	2021	√				√			
Jung et al. (2021)	2021					√			
Jung and Kim (2021)	2021					√			
Castro et al. (2021)	2021		√			√			
Özerk and Elgezen (2022)	2022					√			
Türkoglu et al. (2022)	2022	√				√			
Shivdikar et al. (2022)	2022	√				√			
Shen et al. (2022)	2022					√			
Sujoy Sinha Roy (2019)	2019	√					√		
Mert and Öztürk (2020)	2020	√					√		
Riazi et al. (2020)	2020	√					√		
Kim et al. (2020)	2020						√		
Rashmi (2020)	2020	√					√		
Turan (2020)	2020						√		
Serhan et al. (2021)	2021						√		
Fadhli et al. (2021)	2021						√		
Ye et al. (2021)	2021		√				√		
Xin and Zhao (2021)	2021						√		
Syafalni et al. (2022)	2022						√		
Yang et al. (2022a)	2022						√		
Han et al. (2022)	2022						√		
Agrawal et al. (2022)	2022						√		
Ye and Kannan (2022)	2022						√		
Yang et al. (2022b)	2022						√		
Tan et al. (2021)	2021							√	
Reagen et al. (2021)	2021							√	
Samardzic et al. (2021)	2021							√	
Kim and Kim (2022)	2022							√	
Kim et al. (2022)	2022		√					√	
Samardzic et al. (2022)	2022							√	
Geelen et al. (2022)	2022							√	
Jiang et al. (2022)	2022							√	

Table 4 (continued)

References	Year	Algorithm-based optimization			Hardware-based optimization				
		NTT	Bootstrapping	Encoding	CPU	GPU	FPGA	ASIC	Other
Ahmet Can Mert (2022)	2022							√	
Reis et al. (2020)	2020								√
Gupta (2021)	2021								√
Gupta and Cammarota (2022)	2022								√
Chielle et al. (2022)	2022								√

Note that the article with a '√' for both algorithm-based optimization and hardware-based optimization, represents that it leverages both algorithm optimization and hardware optimization simultaneously

throughput. The authors present algorithmic optimizations including combining ModDown and rescale in Mult, hoisting the ModDown in PtMatVecMult, and compressing the key with a pseudo-random number generator (PRNG). By combining ModDown and rescale in Mult, they realize a faster encrypted inner product. The ModDown in PtMatVecMult leads that the same ciphertext can be computed more efficiently than simply applying the rotate function. Compressing the key with a PRNG can avoid shipping the large random polynomials to dynamic random access memory, instead sending only the short PRNG key. The three optimization methods can reduce the number of operations and the consumption of memory, thus realizing the acceleration of bootstrapping.

Ye et al. (2021) accelerate the application of FHE in CNN. They mainly use a new convolution method and explore the parallelism of the algorithm to realize the algorithm acceleration. By using Im2col convolution and Frequency domain convolution, they guarantee that any pair of elements to be summed or multiplied are in the same position in the vector. Therefore, only Pt-Ct Mult and Additions without the expensive rotations are needed for homomorphic convolution calculations.

Kim et al. (2022) propose the Minimum key-switching (Min-KS) based on the minimal key-switching proposed by Halevi et al. (2018). Compared with minimal key-switching (Halevi 2018), Min-KS further reduces the use of the evaluation key in bootstrapping, which means that the number of data accessed in the chip memory at a time is decreased. In addition, the Min-KS algorithm is generalized so that it can be better applied to common homomorphic operations.

Since bootstrapping involves many primitive operations, many studies have been done to accelerate it by speeding up some of them. The key-switching procedure is the focus of some research works (Bossuat et al. 2021; Castro et al. 2021; Ye et al. 2021; Kim et al. 2022), which reduce the number of operations to achieve the purpose of acceleration.

Encoding optimization

The core idea of encoding optimization is to make the encoded data have more parallel computing potential, so as to accelerate the implementation of the algorithm. However, there are a few research works on algorithm acceleration by using encoding optimization.

Jin et al. (2019) design an encoding strategy for FHE so that the encoded plaintext data can be better executed in parallel. Compared with the previous work, they design the corresponding data encoding strategy for images with higher dimensions. In view of the feature that different weight components are calculated simultaneously in convolution operation, they encode the components that can be parallel into a vector, so as to facilitate the parallel homomorphic operation later. Besides, the encoding strategy improves memory efficiency and reduces the message size transferred.

Algorithm acceleration summary

In general, the acceleration effect of FHE schemes based on algorithm acceleration is limited. Many research works on the acceleration of FHE focus on the acceleration of NTT and bootstrapping. And the Barrett reduction is most commonly used to accelerate NTT. There are two main ways of algorithm optimization. One is to replace operations, which needs high computing resource consumption, with operations consuming high computing resource. The other is to improve the parallelism of the algorithm so that the hardware acceleration schemes can be designed on this basis. In general, the algorithm acceleration schemes are unable to achieve a breakthrough acceleration effect and most works just apply existing algorithms. Therefore, algorithm-based acceleration schemes require a theoretical breakthrough, especially for bootstrapping, in order to make FHE performance closer to the actual application requirements. However, the advantage of an acceleration scheme based on algorithm optimization is that it is more adaptive and can adapt to different hardware platforms.

Hardware acceleration schemes

This section introduces FHE acceleration schemes using hardware platforms, including CPU, GPU, FPGA, ASIC, and others. CPU has the least parallel computing resources, resulting in the worst acceleration effect. GPU is cheaper and easier to use. FPGA is a popular choice for FHE acceleration due to its parallel computing resources and customization. ASIC achieves the best acceleration effect although it's difficult to put into production.

CPU-based

There are only a few CPU-based acceleration schemes, and the acceleration effect that can be achieved is limited.

Based on the Intel® Advanced Vector Extensions 512 (Intel® AVX512) instruction set, Boemer et al. (2021) implement acceleration on polynomial modular multiplication and NTT. They provide it as a Homomorphic Encryption Acceleration Library which can be used in combination with the SEAL library. The acceleration scheme mainly uses two optimization methods, including loop optimization and data parallel computing optimization, which are both based on the instruction set. Loops are unrolled either manually or using a pre-processor directive, with a manually-tuned unrolling factor. Within manually unrolled loops, instructions are reordered where possible for best pipelining. Based on the property of the instruction set, eight 64-bit integers can be processed simultaneously, the acceleration scheme completes the acceleration of NTT by expressing NTT and INTT in the form of elements and processing them together. For the vector and vector multiplication and polynomial and polynomial multiplication expressed in the form of elements, the input data is also aligned based on the purpose for simultaneous processing of eight 64-bit integers to achieve performance optimization.

In Ishimaki et al. (2021), the Trace-Type Function Evaluation of CKKS is accelerated. The homomorphic trace-type function evaluation is performed by repeating homomorphic rotation followed by addition (rotations-and-sums). The homomorphic trace-type function is a commonly used and time-consuming subroutine that enables homomorphically summing up the components of the vector or homomorphically extracting the coefficients of the polynomial. They propose a more efficient trace-type function evaluation using loop-unrolling to reduce the number of expensive operations by leveraging a property of automorphisms and using a multi-core environment, thus reducing the computational cost compared with the sequential method (Halevi and Shoup 2014; Chen et al. 2021) at the expense of slightly increasing the required storage. In addition, they successively unroll consecutive subloops in the trace-type function evaluation and parameterize the number of iterations after the unrolling, which further realizes the acceleration of the Homomorphic Trace-Type Function Evaluation.

Inoue et al. (2022) also accelerate the trace-type function by using Intel® AVX512 instruction set. Through using loop unrolling, they implement the optimization of the trace-type function. Their acceleration scheme can be regarded as another implementation scheme of Intel® AVX512 instruction set in accelerating FHE, without much innovation.

Most studies (Boemer et al. 2021; Inoue and Suzuki 2022)) are based on the Intel® AVX512 instruction set to accelerate FHE, since the mainstream CPU instruction set is not suitable. This also shows the limitations of the CPU for FHE acceleration. Table 5 shows the comparison of acceleration schemes based on CPU. In it, three aspects should be considered in the availability evaluation of schemes. They are the acceleration ability offered, whether the hardware platforms are universal, and whether the schemes are only accelerated architectures or form acceleration libraries that could be directly called. Acceleration results are about the acceleration effect of schemes in terms of latency. We directly cite the acceleration results given in works to show. This is why different schemes have the different significant digits about acceleration result. There are some works (Ahmad Al Badawi 2019; Goey et al. 2021; Castro et al. 2021; Fadhli et al. 2021; Xin and Zhao 2021) that don't give the exact acceleration result in terms of acceleration times. So we compute it by using the experimental result shown in them. And the result is rounded to two decimal places. The Tables 6, 7, 8 and 9 are the same as the Table 5. The '×' in Table 5 means the times. The same is true for the meaning of '×' in the other tables presented in this paper.

Table 5 Comparison of acceleration schemes based on CPU

References	Year	Availability	Data storage optimization	Homomorphic parameters tuning	Accelerated object	Acceleration result (times)
Boemer et al. (2021)	2021	+	√	×	Bootstrapping	4.83–6.26
Ishimaki (2021)	2021	+	√	√	Rotation / HAdd	1.32–2.12
Inoue and Suzuki (2022)	2022	+	√	×	Rotation / HAdd	1.05–2.30

Note that the '×' in the Table indicates that the work doesn't satisfy the attribute represented in this column. The same is true for the meaningof '×' in the other tables presented in this paper

Table 6 Comparison of acceleration schemes based on GPU

References	Year	Availability	Data storage optimization	Homomorphic parameters tuning	Accelerated object	Acceleration result (times)
Ahmad Al Badawi (2019)	2019	+	×	×	HPS RNS variant of the BFV	*
Lupascu (2019)	2019	+ + +	×	×	Ciphertext multiplication	5.14×
Lei et al. (2019)	2019	+ + +	×	×	KeyGen / Bootstrapping	1.672–13.268×
Xia et al. (2019)	2019	+ +	×	×	DGHV	1.67–1.84×
Kim and Jung (2020)	2020	+ + +	√	×	NTT	6.52×
Morshed (2020)	2020	+ +	×	×	TFHE	14.5–20×
Ahmad Al Badawi (2020)	2020	+	×	×	CKKS	10–100×
Ahmad Al Badawi (2021)	2020	+ +	×	×	HPS RNS variant of the BFV	10–1000×
Pedro et al. (2021)	2021	+ + +	×	×	BFV	2.6×
Goey et al. 2021)	2021	+ +	√	×	CMNT	1.41×
Jung et al. 2021)	2021	+ +	√	×	CKKS	4.05×
Jung and Kim (2021)	2021	+ + + +	√	×	Bootstrapping	7.02×
Castro et al. (2021)	2021	+	×	×	NTT / Polynomial Multiplication	1092.29×
Özerk and Elgezen (2022)	2022	+ + +	√	×	NTT / Polynomial Multiplication	90.13–141.95×
Türkoglu et al. (2022)	2022	+ + +	×	×	HMult / Relinearization / Rotation / HAdd	13.39–47.01×
Shivdikar et al. (2022)	2022	+ + + +	√	×	Polynomial Multiplication	123.13×
Shen et al. (2022)	2022	+ + + +	×	×	BGV / BFV / CKKS	234.5–378.4×

*(Ahmad Al Badawi 2019) compares the performance of two optimized variants of BFV, namely BEHZ and HPS. And it doesn't give the acceleration result

Table 7 Comparison of acceleration schemes based on FPGA

References	Year	Availability	Data storage optimization	Homomorphic parameters tuning	Accelerated object	Acceleration result
Sujoy Sinha Roy (2019)	2019	+ + +	×	×	BFV	13×
Mert and Öztürk (2020)	2019	+ + +	×	×	BFV	7–12×
Riazi et al. (2020)	2020	+ +	×	√	CKKS	164–268×
Kim et al. (2020)	2020	+ + + +	√	×	NTT	118×
Rashmi (2020)	2020	+ + +	×	×	Polynomial Multiplication	2950–4200×
Turan (2020)	2020	+ + +	√	×	BFV	5×
Serhan et al. (2021)	2021	+ +	×	×	Bootstrapping	*
Fadhli et al. (2021)	2021	+ +	×	×	BFV	3.85×
Ye et al. (2021)	2021	+ + +	√	×	BFV	3.4–6.7×
Xin and Zhao (2021)	2021	+ + + +	×	×	CKKS	26.04×
Syafalni et al. (2022)	2022	+ +	×	×	Polynomial Multiplication	38.5×
Yang et al. (2022a)	2022	+ +	√	×	BFV	5.6×
Han et al. (2022)	2022	+ + +	×	×	Key-switching	1.6×
Agrawal et al. (2022)	2022	+ + +	√	×	Bootstrapping	533×
Ye and Kannan (2022)	2022	+ +	√	×	Bootstrapping	16.5×
Yang et al. (2022b)	2022	+ +	√	×	NTT	#

*(Serhan et al. 2021) uses less hardware resource to achieve linear performance scaling with up to 16 vector lanes about matrix–vector operations in bootstrapping of TFHE. And it doesn't achieve significant acceleration

#(Yang et al. 2022b) also uses less hardware resource to implement FHE and doesn't give the acceleration result in terms of latency

GPU-based

GPU has been widely used in FHE acceleration in recent years. The acceleration scheme based on GPU is mainly realized by utilizing parallel computing resources it provides. At the same time, the data storage strategy will also be optimized in order to reduce the time taken to access data. However, due to the limited customization flexibility provided by GPU, the innovation of GPU-based

Table 8 Comparison of acceleration schemes based on ASIC

References	Year	Availability	Data storage optimization	Homomorphic parameters tuning	Accelerated object	Acceleration result (times)
Tan et al. (2021)	2021	+ +	√	×	Bootstrapping	2.43 ×
Reagen et al. (2021)	2021	+ +	√	√	BFV	79 ×
Samardzic et al. (2021)	2021	+ + + +	√	×	BGV	5400–14,000 ×
Kim and Kim (2022)	2022	+ + +	√	√	Bootstrapping	1306–5556 ×
Kim et al. (2022)	2022	+ + +	√	×	Bootstrapping	11,590–18,214 ×
Samardzic et al. (2022)	2022	+ + + +	√	×	CKKS	4600 ×
Geelen et al. (2022)	2022	+ + +	√	×	BGV	4000 ×
Jiang et al. (2022)	2022	+ + +	√	×	TFHE	24–160 ×
Ahmet Can Mert (2022)	2022	+ + +	√	×	CKKS	68–78 ×

Table 9 Comparison of acceleration schemes based on other hardware

References	Year	Availability	Data storage optimization	Homomorphic parameters tuning	Accelerated object	Acceleration result (times)
Reis et al. (2020)	2020	+ +	√	×	BFV	14.3 ×
Gupta (2021)	2021	+ +	√	×	Bootstrapping	88,397 ×
Gupta and Cammarota (2022)	2022	+ + +	√	×	GSW	20,000 ×
Chielle et al. (2022)	2022	+ +	√	√	Bootstrapping	10–100 ×

acceleration schemes is also limited. On the other hand, since GPU is a more general hardware platform, its acceleration scheme can be presented in the form of the acceleration library and put into use.

Badawi et al. (2019) implement and evaluate the performance of two optimized variants, which are called Bajard Eynard-Hasan-Zucca (BEHZ) and Halevi-Polyakov-Shoup (HPS), based on CPU and GPU. The lazy reduction and several precomputations are added to optimize the implementation of HPS. When implementing HPS on GPU platform, the discrete galois transform (DGT) is used for efficient polynomial multiplication using negacyclic convolution. It cuts the transform length into half and requires less amount of memory for precomputed twiddle factors. The DGT algorithm was originally proposed by Crandall (1999) for fast negacyclic convolution. Besides, data transfer between CPU and GPU is avoided. The final result shows that HPS performs better than BEHZ, and for 128-bit security settings, HPS is already practical for cloud environments supporting GPU computations.

Lupascu et al. (2019) use several GPUs to accelerate the FHE and first adapt them for HElib. The acceleration scheme proposed in this paper is to make full use of parallel computing resources of multiple GPUs by distributing computing tasks reasonably. For task allocation on multiple GPUs, the unity of task loads is balanced to maximize the work efficiency of multiple GPUs. Before tasks are distributed, for the ciphertext addition, subtraction, and multiplication operations in homomorphic operations, the parallelization of the operation sequence is explored, and the operation sequence that can be executed in parallel is decomposed. The innovation of the acceleration scheme proposed in this paper is limited, and it is just to accelerate the algorithm implementation at the cost of consuming multiple GPUs.

Similar to Lupascu (2019), Lei et al. (2019) also use GPUs to accelerate the adder in FHEW-V2. The cuFTT (https://developer.nvidia.com/cufft) library is used to accelerate the parallelization of FTTs operations in bootstrapping. For a complex multiplication operation, they use the parallelism of GPU to accelerate it. In addition, for the data that need to be shared in the calculation, they put it into the shared memory to improve the data access speed. Different from Lupascu (2019), their acceleration scheme also uses a multicore CPU to accelerate the algorithm. For example, since bootstrapping is independent of each key, this paper uses a multicore CPU to accelerate the key generation process by generating different keys at the same time.

Based on the joint architecture of CPU and GPU, Xia et al. (2019) accelerate the DGHV (Dijk and Gentry 2010). Aiming at serialization operations of DGHV, they explore the method of parallel implementation and use parallel computing resources of GPU to carry out the implementation of corresponding algorithms. For example, for the long message sequence, the acceleration scheme adopts the method of dividing the message into blocks and then simultaneously encrypting each message block. For HAdd, the acceleration scheme also uses the computing resources of GPU to implement it in parallel. In general, the acceleration scheme (Xia et al. 2019) is relatively simple, and the exploration of algorithm parallelism is not very sufficient. So the final acceleration effect is limited.

Different from studies mentioned above, based on GPU, Kim et al. (2020) aim at the severe memory bandwidth bottleneck faced by NTT operation under a larger homomorphic parameters set. An implementation scheme of runtime data generation is proposed, which uses factorization recursion to calculate the rotation factor, thus balancing the calculation speed and space consumption.

Based on CPU and GPU, Morshed et al. (2020) design and implement an acceleration scheme for TFHE homomorphic encryption. On the one hand, they analyze the parallelism for HAdd, HMult, and other operations, and realize the algorithm acceleration by assigning tasks, which can be computed in parallel, to different threads provided by CPU. On the other hand, they improve the parallelization of the algorithm by optimizing Bit Coalescing, Compound Gate, Karatsuba Multiplication (Karatsuba 1962), and so on, so as to make full use of the parallel computing resources of GPU.

Badaw et al. (2020) realize the acceleration of CKKS based on GPU, while the important optimization they used here is related to matrix packing. Packing is useful in FHE to reduce both the number of homomorphic operations due to single instruction multiple data (SIMD) evaluation and the number of ciphertexts. In general, the innovation of the acceleration scheme proposed in this paper is limited, and it can not achieve an obvious acceleration effect.

Based on multiple GPUs, Badawi et al. (2021) also design a data allocation strategy for the computation process of the FHE algorithm, which ensures the load balancing of multiple GPUs. According to the size of the polynomial matrix, the strategy allocates data with limited rows and columns, which achieves acceleration similar to the task allocation strategy proposed by Lupascu et al. (2019). However, Badawi et al. (2021) also design an efficient CPU-GPU communication protocol, thus realizing better algorithm acceleration.

In Alves et al. (2021), the acceleration scheme of BFV algorithm is designed based on GPU. Due to the inspiration of Bailey's version of the Fast Fourier Transform (David 1989), they proposed a novel hierarchical formulation of the DGT that offers about two times lower latency than the best version available in the previous work. The DGT is also faster than NTT due to its lower memory bandwidth requirement. Moreover, they also choose the compatible parameters between the DGT and the RNS representation. In addition, a more efficient and polynomial-oriented state machine is designed to reduce the need for moving data in and out of the DGT domain and between the main memory and the GPU global memory. In a word, in order to better play the performance of the GPU, this paper innovatively uses some mathematical tools and implements them on the GPU through CUDA.

Goey et al. (2021) accelerate CMNT (Coron and Mandal 2011) based on GPU. In terms of hardware optimization, they propose to buffer the twiddle factors in GPU registers and then access these data across different threads during the NTT computation through warp shuffle instruction. Combined with algorithm optimization mentioned in Sect. "NTT optimization", the acceleration scheme finally realizes the further surpassing of cuHE.

Jung et al. (2021) explore the parallel implementation and data access of CKKS. Based on this, the corresponding acceleration schemes are implemented on CPU and GPU and the result shows the effectiveness of the acceleration method. For CPU, the acceleration scheme is based on scatter instructions in AVX-512 (Boemer et al. 2021), which accelerates matrix multiplication. For GPU, they assign each independently computable output element to a thread, so as to realize the acceleration of the algorithm through parallel calculation of non-interference data. This also makes the acceleration scheme more sufficiently explore the parallelism of algorithm implementation compared with cuHE. In addition to the design of parallel strategy, they also calculate and store data in advance to speed up data access.

Jung et al. (2021) implement the optimization of CKKS bootstrapping. By fully exploring the parallelism in FHE, they discover that the major performance bottleneck is their high main-memory bandwidth requirement, which is exacerbated by leveraging existing optimizations targeted to reduce the required computation. Further, they find that inner product in key-switching is a major bottleneck when attempting to accelerate HMult. On this basis, kernel fusion and reordering primary functions, which are memory-centric optimizations, are used to accelerate bootstrapping. And the result shows that the acceleration scheme has achieved good acceleration effect.

Through the analysis of bootstrapping, Castro et al. (2021) find that the main performance bottleneck of bootstrapping mainly comes from the memory bandwidth bottleneck. That is, a large number of intermediate process data needs to be generated and stored during the execution of bootstrapping. On this basis, the optimization method of bootstrapping is proposed, which mainly uses physical address mapping optimization. The physical address mapping of the data in the main memory has a substantial impact on the time it takes to transfer data. The reduction of data transfer time speeds up the implementation of bootstrapping.

Based on GPU, Özerk et al. (2022) design and implement a corresponding acceleration scheme for key generation, encryption, and decryption of FHE using NTT operation. In this paper, the parallel operation of NTT is designed based on the parallel computing resources provided by GPU. At the same time, they also optimize the GPU memory usage and kernel function call, so as to realize the algorithm acceleration. The difference is that they focus on different acceleration objects mentioned above.

Türkoğlu et al. (2022) design and implement a corresponding acceleration scheme for HMult, relinearization, rotation, and HAdd. They use the CUDA to further accelerate the Barrett reduction process. And they provide the corresponding GPU FHE acceleration library.

Based on GPU, Shivdikar et al. (2022) design an acceleration method for polynomial multiplication, which is a very important performance bottleneck in homomorphic encryption. The method optimizes the memory access of GPU to improve the data access speed and throughput during the algorithm operation process, so as to achieve further acceleration.

The acceleration scheme proposed by Shen et al. (2022) is GPU-based and accelerated for word-size FHE, including BGV, BFV, and CKKS. They mainly combine some previous acceleration optimization methods and develop the acceleration library on this basis. This acceleration library can be used for both ordinary GPU and embedded GPU. Meanwhile, it can better support the use of Internet of Things devices. And aiming at the security vulnerabilities of FHE acceleration stock in preventing side-channel attacks, they implement time consistency for multiply-accumulation, conditional subtraction, and Barrett reduction. At the same time, it also carried on the simplification of the implementation instructions. Besides, they also implement two versions for NTT operations, which are performance first and memory first. In view of the limited performance, the operation is further accelerated by simplifying the instructions. For polynomial multiplication operations and ciphertext multiplication operations, a general RNS multiplication kernel is designed to speed up the calculation. In a word, compared with the FHE accelerated GPU library (Türkoglu et al. 2022), the acceleration library has three advantages, which are better adaption, higher security, and more features.

Due to the need for CPU cooperation, most studies (Ahmad Al Badawi 2019, 2021; Xia et al. 2019; Morshed 2020; Jung et al. 2021) based on GPU acceleration require support from CPU. However, schemes designed by these studies focus on using GPU to accelerate FHE. Thus, we have classified these schemes as GPU-based FHE acceleration schemes. Research on GPU-based FHE acceleration started earlier, and currently, GPU-based acceleration schemes are gradually transitioning from proposing acceleration architectures to forming acceleration libraries (Türkoglu et al. 2022; Shen et al. 2022), which greatly improves availability. Additionally, some works (Lupascu 2019; Lei et al. 2019) utilize multiple GPUs instead of a single GPU for accelerating FHE. Table 6 shows the comparison of acceleration schemes based on GPU. We can see that all FHE acceleration schemes based on GPU do not utilize homomorphic parameters tuning. This may be because parameters tuning itself is a complex process, and early acceleration of FHE could achieve good results by just using GPUs. And researchers have not delved deeply into the acceleration of GPU-based schemes. Hence, incorporating homomorphic parameters tuning is also a direction for research into accelerating FHE using GPU.

FPGA-based

Compared with GPU, FPGA can provide more flexible parallel computing resources. Therefore, the acceleration schemes based on FPGA will further explore the parallel computing implementation of FHE. In addition, the data access patterns and data placement strategies are also the focus of researchers. Reasonable data placement can not only speed up data access but also improve the parallelism of algorithm execution. In short, the acceleration scheme based on FPGA can achieve a better acceleration effect compared with GPU on the whole. However, the corresponding acceleration scheme is also more difficult to design.

Based on the joint architecture of ARM-FPGA, Roy et al. (2019) accelerate the BFV. They mainly use the hardware parallel computing resources to design the hardware acceleration architecture and finally realize it, which is more efficient than the software implementation. The acceleration scheme is based on the NTT parallel computing method (Sujoy Sinha Roy 2014) and at the same time they make improvements to it, which is that on-chip memory is used to store a constant rotation factor to save cycles for further acceleration.

Mert et al. (2020) accelerate the encryption and decryption process of BFV by using FPGA. In this paper, a more efficient parallel hardware architecture is proposed for NTT operation, which is divided into two parts. Then the input of each part is calculated in parallel, and finally the calculation results are combined to complete the NTT operation, thus realizing the acceleration of NTT operation. In addition, the most important part of this acceleration scheme is to implement a faster polynomial multiplier using FPGA. And compared with the acceleration scheme (Seiler 2018), the acceleration scheme (Mert and Öztürk 2020) can adapt to larger homomorphic parameters.

Riazi et al. (2020) (HEAX) are based on FPGA to optimize the performance of CKKS, which explores multi-level parallelism for the implementation of algorithms and gives the corresponding optimization method and implementation framework. For HMult, HEAX first modifies the SEAL library so that more parameters can be adapted. In addition, it realizes optimization of polynomial multiplication by storing multiple correlation coefficients of a polynomial in memory space that can be accessed in parallel, thus speeding up the reading and writing speed of the data access. At the same time, the data access mode in NTT process is analyzed. In view of the key-switching operation, the pipeline architecture is designed by analyzing the data dependency, which can execute many key-switching operations simultaneously. As can be seen from the above introduction, the biggest advantage of HEAX is that it sufficiently explores the parallelism of algorithm implementation and carries out optimization implementation on this basis. Therefore, a good acceleration effect has been achieved by HEAX.

Kim et al. (2020), based on FPGA, mainly accelerate NTT operation. Compared with acceleration schemes (Sujoy Sinha Roy 2019; Riazi et al.2020), they focus more on bootstrapping and RNS domain and present a novel hardware architecture of NTT. In the design of the acceleration scheme, they take into account the algorithm to realize the time consumption and space consumption, which maximizes the use of resources. Aiming at butterfly operation in INTT, the acceleration scheme designs the corresponding computing unit to realize it. At the same time, according to the characteristics of butterfly operation and hardware I/O characteristics, it adopts the method of organizing butterfly operation in the way of serialization connection and carries out parallel design of butterfly operation unit in a small range. In addition, data storage is reorganized to speed up the speed of data access during butterfly operation execution. In order to reduce the consumption of memory and balance the consumption of memory and time, some data used in the process of NTT is stored in advance. And the rest of the data is generated during the execution of NTT operation, so as to ensure the fast implementation speed of the algorithm and less memory consumption. However, run-time-generated data and pre-generated data have their own advantages and disadvantages, which are about the balance between time consumption and space consumption, and should be used according to the actual situation.

For the basic operation of LWE-based FHE, such as RNS, CRT, NTT-based polynomial multiplication, modulo inverse, modulo reduction, and all the other polynomial and scalar operations are carried out to design and implement the hardware acceleration library by Rashmi et al. (2020). By mining the parallelism of some basic operations, they adopt the parallelization method for multi-round calculation in RNS, so as to realize the acceleration of the algorithm.

Turan et al. (2020) (HEAWS) accelerate the BFV based on FPGA, who design multiple parallel coprocessors in the FPGA in order to execute several operations simultaneously. At the same time, they also design an off-chip data transfer strategy to meet the needs of multiple coprocessors. Because the FPGA used by them is cloud-based FPGA, which brings the extra communication overhead brought, the state-of-the-art 512-bit XDMA feature of high bandwidth communication is used to reduce the overhead of HW/SW data transfer. However, one of the innovative points of HEAWS is that it uses cloud-based FPGA for the first time, which improves its availability.

Gener et al. (2021) accelerate the bootstrapping algorithm of TFHE based on FPGA. They design an accelerator, which adds TFHE-specific custom instruction extensions to an FPGA-based programmable vector engine. This makes linear performance scale as the number of vector lanes increases from 4 to 16. However, this paper only presents a preliminary architecture to accelerate TFHE bootstrapping, which still uses an $o(n^2)$ algorithm of the polynomial multiplier. And as the paper says, this is one direction in which the accelerator designed could be improved.

Fadhli et al. (2021) mainly realize the systolic arrays by using FPGA, so as to achieve acceleration of the algorithm. Systolic arrays feature the ability to perform multiple calculations on a single input without waiting. At the same time, it can continue to input the required data while calculating.

Ye et al. (2021) focus on the acceleration of the plaintext multiplication ciphertext operation. In this paper, based on multiple processing elements (PEs), parallelization is designed and realized for Hadamard product and accumulations operations of vectors, which is that the vector is divided into subvectors, and then parallelization operations between subvectors were realized. The communication delay between FPGA and external memory is avoided by means of a double data cache. Besides, based on the designed performance model, the optimal homomorphic parameters are selected for each layer of the neural network, so as to achieve further acceleration of the neural network using the FHE. Compared to prior work, the acceleration scheme first proposes the low latency inference accelerator for convolutional layers of ResNet-50 based on FPGA.

Based on FPGA, a multi-level parallelism degree is explored for FHE by Xin et al. (2021). And parallel scheme is designed and implemented to realize algorithm acceleration. This paper mainly implements parallelization acceleration for three levels of the algorithm. NTT parallel implementation is the basic parallel acceleration method. About this acceleration scheme, by designing several butterfly computing units and executing them in parallel, the parallel acceleration of NTT algorithm is realized. RNS parallel implementation is the medium parallel acceleration method. Based on the NTT core composed of multiple butterfly units, it uses three NTT cores to realize acceleration of polynomial multiplication of RNS form. The parallel implementation of two ciphertext operations is the advanced parallel acceleration method. They equip NTT cores for each polynomial in both two ciphertexts and form the high-level parallelism in ciphertexts. Compared with the acceleration scheme (Riazi et al. 2020), under the condition of using the same number of digital signal processor (DSP) blocks, the NTT cores in the acceleration scheme (Xin and Zhao 2021) consume fewer resources.

Aiming at the polynomial multiplication operation used in BFV, Syafalni et al. (2022) propose a method based on a convolution approach to complete the corresponding polynomial multiplication operation. At the same time, this paper also designs a two-dimensional systolic array on the level of hardware implementation, so as to complete the polynomial multiplication with high parallelism. In order to avoid data transmission delay caused by noise generated by software, they first design a corresponding hardware noise generation module.

Based on FPGA, Yang et al. (2022a) design and implement the corresponding acceleration scheme for SCNN which uses FHE. They analyze the memory access requirement in the process of FHE, so as to obtain the memory access pattern. Based on the memory access pattern, they continue to redesign the data flow to avoid memory access conflicts and realize highly parallel execution of the algorithm, so as to complete the acceleration of the algorithm.

Aiming at key-switching operation, which is the performance bottleneck in CKKS, Han et al. (2022) explore the data dependence in this operation to minimize the interdependence between data. Then, based on this, they maximize the parallel calculation of data and achieve the acceleration of the algorithm.

Agrawal et al. (2022) (FAB), aiming at the bootstrapping process of large homomorphic parameters, design a new hardware architecture and balance the memory consumption and computing consumption. FAB architecture efficiently utilizes the available U/BRAM blocks on the FPGA as on-chip memory. Mapping the polynomial data bit-width to that of the U/BRAM blocks data width enables storage of up to 43 MB of on-chip data.

Aiming at the bootstrapping of TFHE, the acceleration scheme is designed and implemented by Ye et al. (2022). On one hand, to enable efficient multi-level parallelism, they customize the data layout of TFHE ciphertext for FPGA on-chip SRAM to optimize data access and reduce memory access conflict. The data layout also improves data reuse to effectively utilize the external memory bandwidth. On the other hand, the acceleration scheme is parameterized and can be configured to achieve high throughput and low latency for TFHE bootstrapping when different users have specific security requirements. This improves its availability of it in another way compared with (Turan 2020).

Yang et al. (2022b) accelerate NTT based on FPGA. To reduce latency, the acceleration scheme merges the preprocessing and postprocessing into the NTT and INTT, respectively. Besides, a reconfigurable modular multiplier, which is based on DSP, is proposed to speed up the modular multiplication. In order to avoid designing an independent memory access pattern for INTT, a unified read/write structure of NTT/INTT is presented. Furthermore, they propose a novel memory access pattern named "cyclic-sharing" to reduce 25% memory capacity. In summary, the most significant feature of the acceleration scheme is reconfigurability.

In summary, various researchers have proposed FPGA-based hardware acceleration schemes for different FHE algorithms. These schemes focus on optimizing specific operations, such as NTT, polynomial multiplication, and bootstrapping. The key to achieving acceleration is to explore the parallelism of the algorithm and design hardware architectures that can utilize it. Some schemes (Sujoy Sinha Roy 2019; Agrawal et al. 2022; Ye and Kannan 2022) also use on-chip memory to store constant factors or pre-generated data to reduce computation time and memory consumption. Cloud-based FPGA is also used to improve the availability of the acceleration scheme (Turan 2020). However, there is still room for improvement in some schemes, such as combining homomorphic parameters tuning to further accelerate FHE. Table 7 shows the comparison of acceleration schemes based on FPGA. It can be seen that the availability of FHE schemes based on FPGA is relatively high.

ASIC-based

ASIC offers the greatest customization flexibility of any hardware platform. However, this is why it is the most difficult to design accelerated solutions based on ASIC because it requires a lot of expertise. Most of the acceleration schemes based on ASIC aim to optimize the data placement strategy, as well as design the corresponding computational circuit so that it can achieve the best effect in the acceleration of FHE. One weakness based on ASIC is that the production process of ASIC is complex and requires a large number of resources. Therefore, the corresponding ASIC-based acceleration solution is difficult to be practical.

In (Tan et al. 2021), aiming at the NTT operation used in bootstrapping, the parallelism of it is fully explored. On the basis of the design and integration of multiple PEs, the corresponding memory management is carried out in the algorithm implementation. They design multiple PEs by exploring the degree of parallelism of the algorithm. Each PE can be adjusted dynamically to perform a different task, including NTT, INTT, and pointwise multiplication. Thus, they realize highly parallel algorithm execution based on multiple PEs. Because, in theory, more PEs should lead to more parallel computing. In addition, suitable memory management strategies are also explored for each execution stage of polynomial multiplication by taking into account the number of PEs. Through memory management, they realize the efficient utilization of memory resources, so as to further realize the algorithm acceleration.

Reagen et al. (2021) implement the acceleration of FHE applications, named as Cheetah, which is different from the general FHE algorithm optimization. Instead, Cheetah combines it with the DNN model to optimize the FHE performance. For example, the optimal parameter analysis model is established to select the optimal parameter first. Then the optimal parameters are selected for FHE used by each layer network in the DNN model in order to reduce algorithm complexity. For the dot product operation involved in BFV homomorphic encryption algorithm, the partial alignment operation is carried out. At the same time, the homomorphic operation sequence is optimized for performance-sensitive homomorphic operation, in order to minimize the use of noise budget. A lower noise budget allows smaller homomorphic parameters to be selected. Cheetah also proposes a hardware acceleration architecture, which greatly accelerates the computation speed of the FHE combined DNN model by utilizing the parallel computing resources of hardware devices, so that it can meet the needs of practical applications. Compared with prior work, Cheetah has been further improved in practicability, which makes the application of FHE combined with DNN more in line with the practical demand.

Samardzic et al. (2021) (F1) introduce universal programmable hardware accelerators in order to accelerate the BGV based on ASIC. Based on ASIC's characteristics, the accelerator speeds up BGV. For various homomorphic operations performed using wide vectors, such as modular addition, modular multiplication, NTTs (forward and inverse in the same unit), and automorphisms, the tailored functional units (FUs) are designed to accelerate them. Several FUs are also grouped in computational clusters for further acceleration. Besides, the programmable framework proposed in F1 fully explores the memory management of FPGA. And then they propose a multi-level memory management system to accelerate the FHE. F1 also uses decoupled data orchestration to hide main memory latency. At the same time, it implements a single-stage bit-sliced crossbar network (Passas et al. 2012) that provides full bisection bandwidth. Moreover F1 adopts a static, exposed microarchitecture: all components have fixed latencies, which are exposed to the compiler. Static scheduling simplifies logic throughout the chip. Compared with the acceleration scheme (Lupascu 2019), F1 is more available because of its programmability.

Kim et al. (2022) (BTS) are optimized for the bootstrapping of CKKS. In view of the problem that different selections of homomorphic parameters will affect the performance of FHE in many ways, they study the time required by different combinations of homomorphic parameters for each slot in the bootstrapping process on the premise of ensuring security, so as to select the optimal homomorphic parameters. As a result, BTS achieves a balance between safety and performance. Besides, a lot of time-consuming functions are analyzed, including NTT, INTT, and BConv. On this basis, two kinds of data parallelism modes are explored, which are residue-polynomial level parallelism (rPLP) and coefficient-level parallelism (CLP). And finally, the CLP method is used to accelerate the parallel algorithm. Based on the CLP, the corresponding parallel execution microarchitecture is designed. At the same time, based on the different types of memory and the data access frequency related to the algorithm, the data storage strategy is optimized to achieve further acceleration of bootstrapping of CKKS. Compared with F1, BTS improves the throughput of bootstrapping.

Kim et al. (2022) (ARK) analyze the memory bottleneck for CKKS bootstrapping hardware acceleration process and design the corresponding solution, which solves the problem of on-chip memory limitation for hardware platforms such as CPU, GPU, and FPGA. On one hand, they design the on-the-fly limb extension, which can pre-calculate the data required for PMult and PAdd operation, so as to reduce the number of times they access data to the on-chip memory. On the other hand, they also design specialized FUs for operations for BConv, thus realizing the acceleration of BConv. In addition, a data distribution strategy is set based on access patterns for BConv to speed up data access. The ARK greatly reduces the number of accesses to the memory under the chip, thus greatly speeding up the implementation of the algorithm. However, the corresponding hardware resource consumption also has increased.

Based on ASIC, Samardzic et al. (2022) (CraterLake) study the acceleration scheme for the application of FHE combined with DNN. CraterLake mainly focuses on solving the huge computing overhead of key-switching and uses boosted key-switching. The key innovation in boosted key-switching is to expand the input polynomial to use wider coefficients. It reduces a large auxiliary operand for key-switching hint (KSH). CraterLake also designs the CRB unit and the KSHGen unit. The CRB unit exploits the high internal reuse to allow much higher throughput than independent multipliers and adders communicating through the register file. KSHGen implements an optimization approach primarily through hardware, which has been previously implemented in software (Halevi and Shoup 2020). In addition, they have optimized scalar modular multipliers and pipeline each multiplier to its energy-optimal point. Compared with F1, CraterLake can support unlimited depth of multiplication. At the same time, compared with HEAX, (Mert and Öztürk 2020), HEAWS, etc., CraterLake can execute the entire application based on FHE.

Geelen et al. (2022) (BASALISC) explore ASIC to accelerate bootstrapping. BASALISC is a three-abstraction-layer RISC architecture. In the process of designing PEs for NTT, BASALISC avoids memory access conflicts in order to accelerate the realization of NTT. Meanwhile, for the twiddle factor, BASALISC designs a twiddle factor factory to reduce the number of twiddles needed to be stored. In addition, the multiply-accumulate unit is designed to accelerate the key-switching process of BGV. The result shows that BASALISC achieves a good balance between performance and resource consumption, and a good acceleration effect. Compared with F1, BASALISC provides better security. And compared with HEAX, BASALISC can adapt homomorphic parameters more in line with realistic requirements.

Jiang et al. (2022) (MATCHA) accelerate TFHE based on ASIC. They first identify the possibility to use approximate integer FFTs and IFFTs to accelerate TFHE without decryption errors. The depth-first iterative conjugate-pair FFT algorithm (Bécoulet and Verguet 2021) also is adopted to decrease the computing overhead of a single FFT or IFFT kernel. Moreover, they propose a pipeline flow for MATCHA to support aggressive bootstrapping key unrolling (Bourse et al. 2018; Zhou et al. 2018), which can reduce the number of HAdd, thus achieving further acceleration. In this paper, the acceleration design for TFHE is relatively novel and can achieve better acceleration effect.

Mert et al. (2022) (Medha) accelerate the homomorphic evaluation for CKKS based on ASIC. Medha is able to flexibly support several homomorphic encryption parameter sets by using a technique, which is called divide-and-conquer. In addition, the corresponding FUs are designed after sufficiently exploring the parallelism of the algorithm implementation. Further, during the design process, the communication efficiency of each unit is taken into account. And a memory-conservative approach is designed to get rid of any off-chip memory access during homomorphic evaluations.

Due to the significant acceleration effect of ASIC-based FHE acceleration solutions, some researchers (Reagen et al. 2021; Samardzic et al. 2022) have been studying how to efficiently apply them to DNN to meet practical needs, and have achieved good results. Table 8 shows the comparison of acceleration schemes based on ASIC. It can be seen that all schemes for FHE acceleration based on ASIC will optimize data storage policy. which has to do with the accelerated customization flexibility ASIC can provide.

Other acceleration schemes

Compared to the hardware mentioned above, a few works have been focused on leveraging other hardware for FHE acceleration, such as processing in memory (PiM). PiM, which is also called computing-in-memory or storage and computing fusions, is different from all the above hardware. Its biggest feature is that it can be calculated directly in memory. Therefore, the time consumption of data transmission is greatly reduced. Based on this, the algorithm can be accelerated. As a new hardware platform, the design of FHE acceleration scheme based on PiM is more at the theoretical level, and it is also difficult to put into production and application. Table 9 shows the comparison of acceleration schemes based on other hardware.

Reis et al. (2020) are the first to support the execution of all essential evaluation operations of the BFV scheme within a PiM framework and propose a mapping of polynomial primitives to the underlying PiM hardware, with support to polynomial ring operations. They also use complementary metal sxide semiconductor-based custom memory peripherals to support different operations. As a result, the acceleration scheme achieve a good acceleration result.

Based on the PiM, Gupta et al. (2021) accelerate the key homomorphic operations, including bootstrapping, HAdd, homomorphic subtraction, HMult, polynomial addition, polynomial multiplication, and NTT. The biggest contribution of PiM is avoiding frequent movement of data and speeding up data access. Then the time consumption of data-intensive computing tasks is reduced. Later, they in Gupta and Cammarota (2022) accelerate GSW and design the pipeline structure based on PiM, so as to accelerate the implementation of the algorithm.

In (Chielle et al. 2022), the analog–digital implementation and circuit implementation of FHE is converted, so that more homomorphic operations can be supported. At the same time, the algorithm implementation is accelerated.

Hardware acceleration summary

Compared with acceleration based on algorithm optimization, scholars pay more attention to hardware-based FHE acceleration. Specifically, a lot of researchers focus on it based on FPGA. This is because, on the one hand, FPGA can provide parallelism similar to GPU, while at the same time, they can provide customization similar to ASIC. Therefore, FPGA has the greatest practicability, which can realize the ideal acceleration effect and facilitate more extensive applications. Compared with FPGA, GPU-based acceleration schemes are easier to use and still have good acceleration effects. Moreover, the acceleration of FHE based on GPU is gradually mature, forming some GPU acceleration libraries that can be directly called. There are two main types of hardware-based acceleration. One is to make full use of the parallel computing resources of the hardware platform, by fully exploring the parallelism of the algorithm, especially for NTT, and then design the pipeline structure. This kind of optimization method can fully combine the parallel computing resources of hardware with the parallel execution of the algorithm, so as to accelerate FHE. The other is that, based on hardware memory characteristics and algorithmic data access patterns, different data generation and storage strategies are designed, such as data precomputation, data runtime generation, data preplacement, and other operations. This kind of optimization method can speed up the data access speed and thus accelerate the implementation of the algorithm. The advantage of a hardware-based acceleration solution is that it can achieve a better acceleration effect, while it requires hardware support and is difficult to develop, especially for hardware platforms such as FPGA and ASIC.

Challenges and future research directions

In this paper, we review FHE acceleration from two aspects: algorithm-based acceleration and hardware-based acceleration. Owing to the considerable potential for privacy protection provided by FHE while the bottleneck lies in its performance, a rapid development is witnessed in accelerating FHE. The chosen of hardware used for acceleration is from the common CPU to the complex ASIC, which is more suitable for FHE acceleration, with more and more algorithmic optimizations applied. Despite the progress made in accelerating FHE, there remain several open problems, and the performance of FHE is yet to mature to meet the demands of real-time services. To inspire the following research of it, here we list several future directions worthy of further investigation.

Homomorphic parameters selection The selection of homomorphic parameters also needs further research while a few studies on it. The homomorphic parameters selection not only decides the security of FHE, but also a reasonable selection of it can further improve the performance of FHE. It is quite a complex process, while each scheme has specifically selected parameters, all of which are interlinked. And these parameters are usually selected based on current possible lattice-based attacks and their existing limits. An example of a weakness in this approach to parameter selection was exploited in an attack by Lee (Moon Sung Lee 2011), where the parameter selection in the scheme (Gentry and Halevi 2011) was not conservative enough to prevent a lattice-based attack exploiting the sparse subset sum problem. In addition, according to different application requirements, there are usually different optimal homomorphic parameters. For example, when FHE is applied to DNN, the corresponding optimal parameter can be selected for each layer of the network. Therefore, more efforts into parameter selection are needed, which must ensure the most suitable parameters are chosen to guarantee both efficiency and security.

Memory management Memory storage is another major bottleneck in the implementation of practical FHE. It involves the use of large parameter sizes and very large ciphertext sizes, which can consume significant amounts of memory. As a result, memory management becomes crucial in FHE implementations. Different FHE schemes, different execution stages of the same FHE scheme, will have different data access patterns. At the same time, different hardware has corresponding memory structures. Existing work explores the data access pattern and data dependency, and designs memory management strategies, including data pregeneration, data preplacement, and data reuse, which also take the characteristics of hardware into account. For example, some methods optimize the utilization of highly accessed data by placing it in the fastest accessible memory in the hardware. However, the current research on memory management strategy is still not enough. Taking GPU as an example, the programming flexibility it provides limits the design of optimal memory management strategies. In addition, most of the research on memory management focuses on NTT and bootstrapping. Therefore, the research on the design of better memory management strategies based on hardware that can provide more flexible memory management needs to be further developed. How to combine hardware characteristics with specific algorithms to minimize storage overhead is important. Especially for special devices, good memory management is extremely important. At the same time, it is necessary to further explore the data dependency and data access pattern of other FHE primitive operations.

More suitable hardware At present, the GPU, FPGA, and ASIC are the major hardware used for FHE acceleration. Among them, the research on GPU has been relatively mature and there are some GPU-based FHE acceleration libraries, including cuHE, cuFHE (CUDA-accelerated Fully Homomorphic Encryption Library 2018). In practice, however, the GPU is not suitable for FHE acceleration because of its bandwidth bottleneck. This is because when GPU performs a computing task, data must be transferred from CPU to GPU and then sent back to CPU after the task execution. The large amount of data that FHE generates results in a large memory bandwidth requirement. FPGA and ASIC also have the problem of memory limitation. In addition, due to the specificity of AISC itself, which means that manufacturing ASIC is costly and difficult for researchers to manufacture ASIC, the acceleration scheme based on it only stays in emulation. And it is difficult to directly evaluate and apply, although acceleration schemes based on it have the best acceleration results. Therefore, it is necessary for further research to use new hardware to accelerate FHE. For example, Gupta et al. (Gupta and Cammarota 2022) try to use PiM to accelerate FHE. At one end of the spectrum, it is still of great research value to design better FHE acceleration schemes based on existing hardware.

Application convenience and flexibility There are many acceleration schemes for FHE, but a few of them are easy to use. The construction theory of FHE is much more complex than the basic symmetric and asymmetric encryption algorithms, such as AES, DES, and RSA, which undoubtedly makes it more difficult for non-professionals to use it. Moreover, since almost all existing acceleration schemes rely on hardware, extra hardware knowledge hinders the use of FHE acceleration schemes even for cryptography professionals. Therefore, the convenience of the accelerated program needs to be further improved. At the same time, different application scenarios have different application requirements for FHE. Therefore, a general acceleration framework also requires further investigation, with the ability to meet different application requirements.

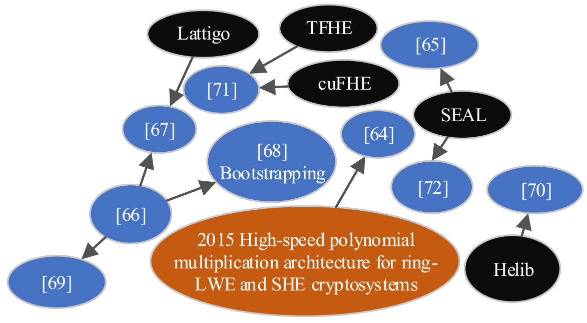
Lattigo
TFHE
[65]
[71]
cuFHE
SEAL
[67]
[68] Bootstrapping
[64]
[72]
[70]
[66]
2015 High-speed polynomial multiplication architecture for ring-LWE and SHE cryptosystems
Helib
[69]

Fig. 3 Reference relationship for comparison of acceleration effects of ASIC-based FHE acceleration schemes

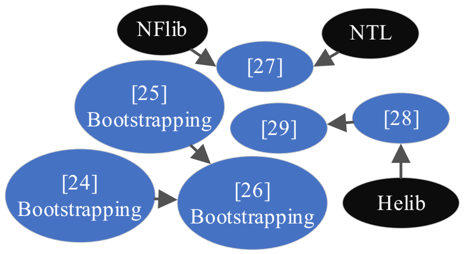

Fig. 4 Reference relationship for comparison of acceleration effects of CPU-based FHE acceleration schemes

Algorithm optimization At present, most research works on using algorithm optimization to accelerate FHE focus on just applying existing algorithms. Unfortunately, this does not address the inherent performance bottleneck of the FHE itself. Especially for the bootstrapping process, due to its huge computing resources and storage overhead, FHE is often applied in the practical application process in the way of leveled FHE. Therefore, if we want to significantly improve the performance of FHE, we need to make a theoretical breakthrough in FHE itself, and this is a very worthy direction for further research, although it is extremely difficult. Still, further research is needed to explore existing algorithms and techniques that are more suitable for FHE acceleration. For example, batching techniques proposed for FHE schemes could greatly improve the performance of any implementation and should also be investigated further.

In summary, while theoretical innovations in FHE hold promise for greatly improving performance, it may take a significant amount of time to realize these improvements. In the short term, the hardware-based acceleration of FHE is bringing us closer to practical FHE. In addition, how to effectively combine various optimization

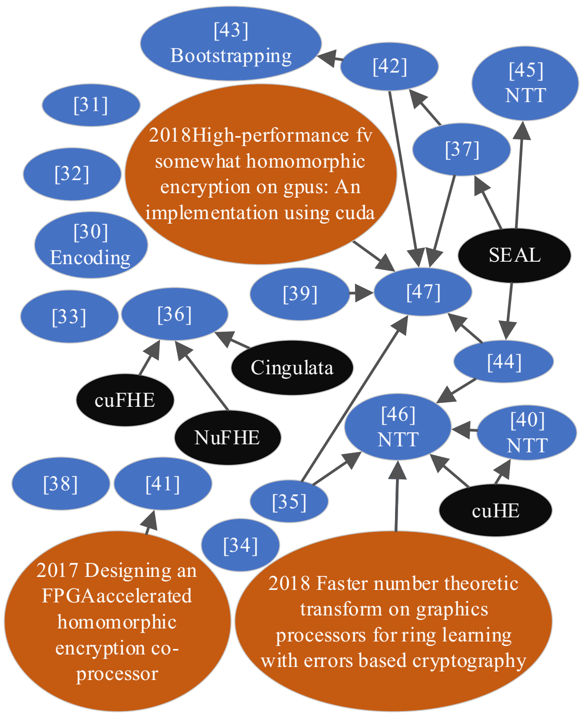
[43] Bootstrapping
[42]
[45] NTT
[31]
2018High-performance fv somewhat homomorphic encryption on gpus: An implementation using cuda
[37]
[32]
[30] Encoding
SEAL
[39]
[47]
[33]
[36]
Cingulata
[44]
cuFHE
NuFHE
[46] NTT
[40] NTT
[38]
[41]
[35]
cuHE
[34]
2017 Designing an FPGAaccelerated homomorphic encryption co-processor
2018 Faster number theoretic transform on graphics processors for ring learning with errors based cryptography

Fig. 5 Reference relationship for comparison of acceleration effects of GPU-based FHE acceleration schemes

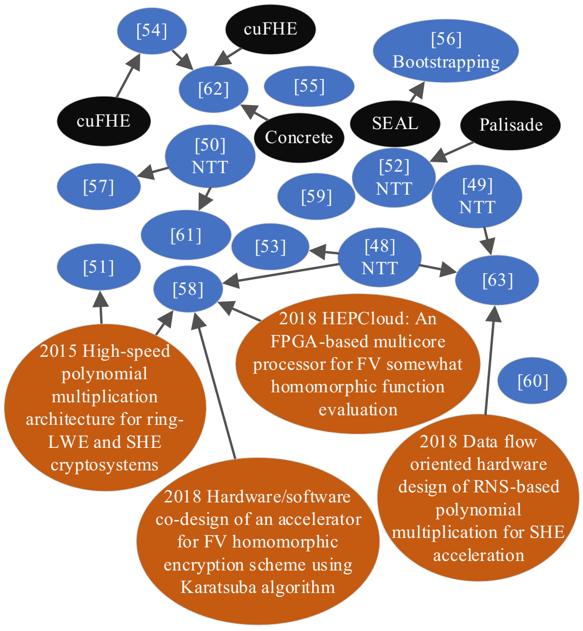
[54]
cuFHE
[56] Bootstrapping
[62]
[55]
cuFHE
Concrete
SEAL
Palisade
[50] NTT
[57]
[59]
[52] NTT
[49] NTT
[61]
[53]
[48] NTT
[51]
[58]
[63]
2015 High-speed polynomial multiplication architecture for ring-LWE and SHE cryptosystems
2018 HEPCloud: An FPGA-based multicore processor for FV somewhat homomorphic function evaluation
[60]
2018 Data flow oriented hardware design of RNS-based polynomial multiplication for SHE acceleration
2018 Hardware/software co-design of an accelerator for FV homomorphic encryption scheme using Karatsuba algorithm

Fig. 6 Reference relationship for comparison of acceleration effects of FPGA-based FHE acceleration schemes

methods to achieve fine-grained FHE acceleration is also an issue that researchers need to consider. Figures 3, 4, 5 and 6 respectively shows the reference relationship for comparison of acceleration effects of FHE acceleration schemes based on different hardware. The articles in blue represent themselves discussed in this paper, which are related to FHE acceleration and published between 2019 and 2022. The articles in orange are related to FHE acceleration but were published earlier than 2019. The black node represents the FHE implementation library. In the process of drawing them, we found that there is no good baseline when comparing the acceleration effect of FHE acceleration schemes. Therefore, how to create a reasonable baseline is also a valuable research direction, which can promote the development of FHE accelerated resea rch.

Conclusion

By conducting a thorough analysis and comparison of various methods for accelerating fully homomorphic encryption (FHE), we comprehensively explored the future research directions of homomorphic encryption acceleration from multiple perspectives. The primary objective of this article is to provide researchers in the field of homomorphic encryption acceleration with a clear, comprehensive, and in-depth perspective, in order to facilitate a better understanding and application of homomorphic encryption technology, and to offer valuable guidance and support for the development of homomorphic encryption in practical applications and theoretical research. It is our belief that the novel ideas and solutions presented in this article will have a positive impact and drive the research and application of homomorphic encryption acceleration, and promote the advancement and dissemination of FHE.

Abbreviations

HE	Homomorphic encryption
PHE	Partially homomorphic encryption
FHE	Fully homomorphic encryption
SWHE	Somewhat homomorphic encryption
HAdd	Homomorphic addition
HMult	Homomorphic multiplication
CUDA	Compute unified device architecture
FU	Functional unit
PE	Processing element
CPU	Central processing unit
GPU	Graphics processing unit
FPGA	Field programmable gate array
ASIC	Application specific integrated circuit
PiM	Processing in memory
RNS	Residue number system
DSP	Digital signal processor
NTT	Number theoretic transforms
INTT	Inverse number theoretic transforms
CRT	Chinese remainder theorem
DGT	Discrete galois transform
SIMD	Single instruction multiple data
PRNG	Pseudo-random number generator

Author contributions
Drafting the manuscript: YG, JW and HZ. Revising the manuscript critically for important intellectual content: XC, JM and VBM. All authors read and approved the final manuscript.

Funding
The work was supported in part by National Natural Science Foundation of China under Grant No.62272028 and 6230203.

Availability of data and materials
Not applicable.

Declarations

Ethics approval and consent to participate
Not applicable.

Competing interests
No potential conflict of interest was reported by the authors.

Received: 17 May 2023 Accepted: 30 August 2023
Published online: 01 March 2024

References
https://developer.nvidia.com/cufft
Acar A, Hidayet Aksu A, Uluagac S, Conti M (2018) A survey on homomorphic encryption schemes: theory and implementation. ACM Comput Surv 51(4):79:1-79:35
Agrawal R, de Castro L, Yang G, Juvekar C, Yazicigil RT, Chandrakasan AP, Vaikuntanathan V, Joshi A (2022) FAB: an FPGA-based accelerator for bootstrappable fully homomorphic encryption. CoRR abs/2207.11872
Alaya B, Laouamer L, Msilini N (2020) Homomorphic encryption systems statement: trends and challenges. Comput Sci Rev 36:100235
Alves PGMR, Ortiz JN, Aranha DF (2021) Faster homomorphic encryption over GPGPUs via hierarchical DGT. Financial Cryptography (2), pp 520–540
Badawi AA, Hoang L, Mun CF, Laine K, Aung KMM (2020) PrivFT: private and fast text classification with homomorphic encryption. IEEE Access 8:226544–226556
Badawi AA, Polyakov Y, Aung KMM, Veeravalli B, Rohloff K (2019) Implementation and performance evaluation of RNS variants of the BFV homomorphic encryption scheme. IEEE Trans Emerg Topics Comput 9(2):941–956
Badawi AA, Veeravalli B, Lin J, Xiao N, Kazuaki M, Mi AKM (2021) Multi-GPU design and performance evaluation of homomorphic encryption on GPU clusters. IEEE Trans Parallel Distrib Syst 32(2):379–391
Bajard JC, Eynard J, Anwar Hasan M, Zucca V (2016) A full RNS variant of FV like somewhat homomorphic encryption schemes. SAC, pp 423–442
Barrett P (1986) Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor. CRYPTO, pp 311–323
Boemer F, Kim S, Seifu G, de Souza FDM, Gopal V (2021) Intel HEXL: accelerating homomorphic encryption with intel AVX512-IFMA52. WAHC@CCS, pp 57–62
Boneh D, Goh EJ, Nissim K (2005) Evaluating 2-DNF formulas on ciphertexts. TCC, pp 325–341
Bos JW, Lauter KE, Naehrig M (2014) Private predictive analysis on encrypted medical data. J Biomed Informatics 50:234–243
Bossuat JP, Troncoso-Pastoriza JR, Hubaux JP (2022) Bootstrapping for approximate homomorphic encryption with negligible failure-probability by using sparse-secret encapsulation. ACNS, pp 521–541
Bossuat JP, Mouchet C, Troncoso-Pastoriza JR, Hubaux JP (2021) Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. EUROCRYPT (1), pp 587–61
Bourse F, Minelli M, Minihold M, Paillier P (2018) Fast homomorphic evaluation of deep discretized neural networks. CRYPTO (3), pp 483–512
Brakerski Z, Gentry C, Vaikuntanathan V (2014) (Leveled) Fully homomorphic encryption without bootstrapping. ACM Trans Comput Theory 6(3):131–1336
Bécoulet A, Verguet A (2021) A depth-first iterative algorithm for the conjugate pair fast fourier transform. IEEE Trans Signal Process 69:1537–1547
CUDA-accelerated fully homomorphic encryption library. https://github.com/vernamlab/cuFHE (2018)
Chen H, Chillotti I, Song Y (2019) Improved bootstrapping for approximate homomorphic encryption. EUROCRYPT (2), pp 34–54
Chen H, Dai W, Kim M, Song Y (2021) Efficient homomorphic conversion between (Ring) LWE ciphertexts. ACNS (1), pp 460–479
Cheon JH, Kim A, Kim M, Song Y (2017) Homomorphic encryption for arithmetic of approximate numbers. In: Takagi T, Peyrin T (eds) ASIACRYPT 1. Springer, Cham, pp 409–437
Chi-Chih Yao A (1982) Protocols for secure computations (Extended Abstract). FOCS, pp 160–164
Chielle E, Mazonka O, Gamil H, Maniatakos M (2022) Accelerating fully homomorphic encryption by bridging modular and bit-level arithmetic. ICCAD 100(1–100):9
Chillotti I, Gama N, Georgieva M, Izabachène M (2017) Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE. ASIACRYPT (1), pp 377–408
Coron JS, Mandal A, Naccache D, Tibouchi M (2011) Fully homomorphic encryption over the integers with shorter public keys. In: Rogaway P (ed) CRYPTO 2011. Springer, Berlin, Heidelberg, pp 487–504
Crandall RE (1999) Integer convolution via split-radix fast Galois transform. Center for Advanced Computation Reed College
Dai W, Sunar B (2015) cuHE: a homomorphic encryption accelerator library. BalkanCryptSec, pp 169–186
David HB (1989) FFTs in external of hierarchical memory. SC, pp 234–24
Deviani R, Nazhifah SA, Aziz AS (2022) Fully homomorphic encryption for cloud based e-government data. Cyberspace J Pendidik Teknol Inf 6:105–118
de Castro L, Agrawal R, Yazicigil RT, Chandrakasan AP, Vaikuntanathan V, Juvekar C, Joshi A (2021) Does fully homomorphic encryption need compute acceleration? CoRR abs/2112.06396
Fadhli H, Syafalni I, Sutisna N, Mulyawan R, Iqbal Arsyad M, Adiono T (2021) Accelerating homomorphic encryption using systolic arrays with polynomial optimization. In: 2021 International Symposium on Electronics and Smart Devices (ISESD). IEEE, pp 1–6
Fan J, Vercauteren F (2012) Somewhat practical fully homomorphic encryption. IACR Cryptol Eprint Arch 2012:144
El Gamal T (1985) A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans Inf Theory 31(4):469–472
Geelen R, Van Beirendonck M, Pereira HVL, Huffman B, McAuley T, Selfridge B, Wagner D, Dimou G, Verbauwhede I, Vercauteren F, Archer DW (2022) BASALISC: programmable asynchronous hardware accelerator for BGV fully homomorphic encryption. Cryptology ePrint Archive
Gentry C, Sahai A, Waters B (2013) Homomorphic encryption from learning with errors: conceptually-simpler, asymptotically-faster, attribute-based. In: Canetti R, Garay JA (eds) Advances in cryptology–CRYPTO 2013. Springer, Berlin, pp 75–92
Gentry C (2009) A fully homomorphic encryption scheme[M]. Stanford university
Gentry C, Halevi S (2011) Implementing gentry's fully-homomorphic encryption scheme. EUROCRYPT, pp 129–148
Gentry C, Halevi S, Smart NP (2012) Homomorphic evaluation of the AES circuit. CRYPTO, pp 850–867
Goey J-Z, Lee W-K, Goi B-M, Yap W-S (2021) Accelerating number theoretic transform in GPU platform for fully homomorphic encryption. J Supercomput 77(2):1455–1474
Gupta S, Cammarota R, Rosing TŠ (2022) MemFHE: end-to-end computing with fully homomorphic encryption in memory. ACM Trans Embed Comput Syst. https://doi.org/10.1145/3569955
Gupta S, Rosing TS (2021) Invited: accelerating fully homomorphic encryption with processing in memory. DAC, pp 1335–1338
Halevi S, Shoup V (2020) Design and implementation of HElib: a homomorphic encryption library. IACR Cryptol Eprint Arch 2020:1481
Halevi S, Shoup V (2021) Bootstrapping for HElib. J Cryptol 34(1):7

Halevi S, Shoup V (2018) Faster homomorphic linear transformations in HElib. CRYPTO (1), pp 93–120
Halevi S, Shoup V (2014) Algorithms in HElib. CRYPTO (1), pp 554-571
Han K, Ki D (2020) Better bootstrapping for approximate homomorphic encryption. CT-RSA, pp 364–390
Han M, Zhu Y, Lou Q, Zhou Z, Guo S, Ju L (2022) coxHE: a software-hardware co-design framework for FPGA acceleration of homomorphic computation. DATE, pp 1353–1358
Hanafizadeh P, Ravasan AZ (2020) A systematic literature review on IT outsourcing decision and future research directions. J Glob Inf Manag 28(2):160–201
Inoue K, Suzuki T, Yamana H (2022) Acceleration of homomorphic unrolled trace-type function using AVX512 instructions. WAHC@CCS, pp 47–52
Ishimaki Y, Yamana H (2021) Faster homomorphic trace-type function evaluation. IEEE Access 9:53061–53077
Jiang L, Lou Q, Joshi N (2022) MATCHA: a fast and energy-efficient accelerator for fully homomorphic encryption over the torus. DAC, pp 235–240
Jiayi H, Jiahui D, Wenqing W, Jiawei Q (2020) Multi-party secure computing financial shared platform based on lightweight privacy protection under FHE. In: 2020 international conference on artificial intelligence and computer engineering. IEEE, pp 245–249
Jin C, Badawi AA, Unnikrishnan B, Lin J, Mun CF, Brown JM, Campbell JP, Chiang M, Kalpathy-Cramer J, Chandrasekhar VR, Krishnaswamy P, Aung KMM (2019) CareNets: efficient homomorphic CNN for high resolution images. NeurIPS Workshop on Privacy in Machine Learning (PriML)
Jung W, Kim S, Ahn JH, Cheon JH, Lee Y (2021) Over 100x faster bootstrapping in fully homomorphic encryption through memory-centric optimization with GPUs. IACR Trans Cryptogr Hardw Embed Syst 4:114–148
Jung W, Lee E, Kim S, Kim J, Kim N, Lee K, Min C, Cheon JH, Ahn JH (2021) Accelerating fully homomorphic encryption through architecture-centric analysis and optimization. IEEE Access 9:98772–98789
Karatsuba AA, Ofman YP (1962) Multiplication of many-digital numbers by automatic computers. Doklady Akademii Nauk Rus Acad Sci 145(2):293–294
Kim S, Jung W, Park J, Ahn JH (2020) Accelerating number theoretic transformations for bootstrappable homomorphic encryption on GPUs. IISWC, pp 264–275
Kim S, Kim J, Kim MJ, Jung W, Kim J, Rhu M, Ahn JH (2022) BTS: an accelerator for bootstrappable fully homomorphic encryption. ISCA, pp 711–725
Kim S, Lee K, Cho W, Nam Y, Cheon JH, Rutenbar RA (2020) Hardware architecture of a number theoretic transform for a bootstrappable RNS-based homomorphic encryption scheme. FCCM, pp 56–64
Kim J, Lee G, Kim S, Sohn G, Rhu M, Kim J, Ahn JH (2022) ARK: fully homomorphic encryption accelerator with runtime data generation and inter-operation key reuse. MICRO, pp 1237–1254
Lee MS (2011) On the sparse subset sum problem from Gentry-Halevi's implementation of fully homomorphic encryption. IACR Cryptol ePrint Arch 2011:567
Lei X, Guo R, Zhang F, Wang L, Xu R, Qu G (2019) Accelerating homomorphic full adder based on FHEW using multicore CPU and GPUs. HPCC/SmartCity/DSS, pp 2508–2513
Li J, Ye H, Li T, Wang W, Wenjing Lou Y, Hou T, Liu J, Rongxing L (2022) Efficient and Secure Outsourcing of Differentially Private Data Publishing With Multiple Evaluators. IEEE Trans Dependable Secur Comput 19(1):67–76
Lupascu C (2019) Mihai Togan. Acceleration Techniques for Fully-Homomorphic Encryption Schemes. CSCS, Victor Valeriu Patriciu, pp 118–122
Marcolla C, Sucasas V, Manzano M, Bassoli R, Fitzek FHP, Aaraj N (2022) Survey on fully homomorphic encryption, theory, and applications. Proc IEEE 110(10):1572–1609
Marten van D, Ari J (2010) On the impossibility of cryptography alone for privacy-preserving cloud computing. hotsec
Mert AC, Kwon AS, Shin Y, Yoo D, Lee Y, Roy SS (2023) Medha: microcoded hardware accelerator for computing on encrypted data. IACR Trans Cryptogr Hardw Embed Syst 1:463–500
Mert AC, Öztürk E, Savas E (2020) Design and implementation of encryption/decryption architectures for BFV homomorphic encryption scheme. IEEE Trans Very Large Scale Integr Syst 28(2):353–362
Moore C, O'Neill M, O'Sullivan E, Doröz Y, Sunar B (2014) Practical homomorphic encryption: a survey. ISCAS, pp 2792–2795
Morshed T, Aziz MMA, Mohammed N (2020) CPU and GPU accelerated fully homomorphic encryption. HOST, pp 142–153
Paillier P (1999) Public-key cryptosystems based on composite degree residuosity classes. EUROCRYPT, pp 223–238
Özerk Ö, Elgezen C, Mert AC, Öztürk E, Savaş E (2022) Efficient number theoretic transform implementation on GPU for homomorphic encryption. J Supercomput 78(2):2840–2872. https://doi.org/10.1007/s11227-021-03980-5
Passas G, Katevenis M, Pnevmatikatos DN (2012) Crossbar NoCs Are Scalable Beyond 100 Nodes. IEEE Trans Comput Aided Des Integr Circuits Syst 31(4):573–585
Rashmi SA, Bu L, Kinsy MA (2020) Fast Arithmetic Hardware Library For RLWE-Based Homomorphic Encryption. FCCM, p 206
Reagen B, Choi W, Ko Y, Lee VT, Lee HS, Wei GY, Brooks D (2021) Cheetah: optimizing and accelerating homomorphic encryption for private inference. HPCA, pp 26–39
Reis D, Takeshita J, Jung T, Niemier MT, Hu XS (2020) Computing-in-memory for performance and energy-efficient homomorphic encryption. IEEE Trans Very Large Scale Integr Syst 28(11):2300–2313
Riazi SM, Laine K, Pelton B, Dai W (2020) HEAX: an architecture for computing on encrypted data. ASPLOS, pp 1295–1309
Rivest RL, Adleman L, Dertouzos ML (1978) On data banks and privacy homomorphisms. Found Secure Comput 4(11):169–180
Roy SS, Vercauteren F, Mentens N, Chen DD, Verbauwhede I (2014) Compact ring-LWE cryptoprocessor. CHES, pp 371–391
Roy SS, Turan F, Järvinen K, Vercauteren F, Verbauwhede I (2019) FPGA-based high-performance parallel architecture for homomorphic computing on encrypted data. HPCA, pp 387–398
Samardzic N, Feldmann A, Krastev A, Devadas S, Dreslinski RG, Peikert C, Sánchez D (2021) F1: a fast and programmable accelerator for fully homomorphic encryption. MICRO, pp 238–252
Samardzic N, Feldmann A, Krastev A, Manohar N, Genise N, Devadas S, Eldefrawy K, Peikert C, Sánchez D (2022) CraterLake: a hardware accelerator for efficient unbounded computation on encrypted data. ISCA, pp 173–187
Sander T, Young AL, Yung M (1999) Non-interactive cryptocomputing for NC1. FOCS, pp 554–567
Schönhage A, Strassen V (1971) Schnelle Multiplikation großer Zahlen. Computing 7(3–4):281–292
Seiler G (2018) Faster AVX2 optimized NTT multiplication for Ring-LWE lattice cryptography. IACR Cryptol Eprint Arch 2018:39
Serhan G, Parker N, Daniel T, Silas R, Lemieux, Guy, Philip B (2021) An fpga-based programmable vector engine for fast fully homomorphic encryption over the torus. SPSL: Secure and Private Systems for Machine Learning (ISCA Workshop)
Shen S, Hao Yang Y, Liu ZL, Zhao Y (2022) CARM: CUDA-accelerated RNS multiplication in word-wise homomorphic encryption schemes for internet of things. IEEE Trans Comput. https://doi.org/10.1109/TC.2022.3227874
Shivdikar K, Jonatan G, Mora E, Livesay N, Agrawal R, Joshi A, Abellán JL, Kim J, Kaeli DR (2022) Accelerating polynomial multiplication for homomorphic encryption on GPUs. SEED, pp 61–72
Syafalni I, Jonatan G, Sutisna N, Mulyawan R, Adiono T (2022) Efficient homomorphic encryption accelerator with integrated PRNG using low-cost FPGA. IEEE Access 10:7753–7771
Tan W, Case BM, Gengran Hu, Gao S, Lao Y (2021) An ultra-highly parallel polynomial multiplier for the bootstrapping algorithm in a fully homomorphic encryption scheme. J Signal Process Syst 93(6):643–656
Torres WAA, Bhattacharjee N, Srinivasan B (2014) Effectiveness of fully homomorphic encryption to preserve the privacy of biometric data. iiWAS, pp 152–158
Turan F, Roy SS, Verbauwhede I (2020) HEAWS: an accelerator for homomorphic encryption on the amazon AWS FPGA. IEEE Trans Comput 69(8):1185–1196
Türkoglu EP, Özcan AS, Ayduman C, Mert AC, Öztürk E, Savas E (2022) An accelerated GPU library for homomorphic encryption operations of BFV scheme. ISCAS, pp 1155–1159
van Dijk M, Gentry C, Halevi S, Vaikuntanathan V (2010) Fully homomorphic encryption over the integers. EUROCRYPT, pp 24–43
Wood A, Najarian K, Kahrobaei D (2021) Homomorphic encryption for machine learning in medicine and bioinformatics. ACM Comput Surv 53(4):70:1-70:35

Xia J, Ma Z, Dai X (2019) Parallel computing mode in homomorphic encryption using GPUs acceleration in cloud. J Comput 14(7):451–469
Xin G, Zhao Y, Han J (2021) A multi-layer parallel hardware architecture for homomorphic computation in machine learning. ISCAS, pp 1–5
Yang S, Bai-Long Y, Chen Y, Zepeng Y, Yi-Wei L (2022b) A highly unified reconfigurable multicore architecture to speed up NTT/INTT for homomorphic polynomial multiplication. IEEE Trans Very Large Scale Integr Syst 30(8):993–1006
Yang Y, Kuppannagari SR, Kannan R, Prasanna VK (2022a) FPGA accelerator for homomorphic encrypted sparse convolutional neural network inference. FCCM, pp 1–9
Ye T, Kannan R, Prasanna VK (2022) FPGA acceleration of fully homomorphic encryption over the torus. HPEC, pp 1–7
Ye T, Kuppannagari SR, Kannan R, Prasanna VK (2021) Performance modeling and FPGA acceleration of homomorphic encrypted convolution. FPL, pp 115–121
Zhang J, Cheng X, Yang L, Hu J, Liu X, Chen K (2022) Fully homomorphic encryption accelerators. arXiv preprint arXiv:2212.01713
Zhou T, Yang X, Liu L, Zhang W, Li N (2018) Faster bootstrapping with multiple addends. IEEE Access 6:49868–49876

Publisher's Note

Yanwei Gong is currently pursuing the Ph.D. Degree in Cyberspace Security at Beijing Key Laboratory of Security and Privacy in Intelligent Transportation, Beijing Jiaotong University, China. His interests include identity authentication protocol related to MEC and fully homomorphic encryption acceleration.

Xiaolin Chang is a professor at School of Computer and Information Technology, Beijing Jiaotong University. Her current research interests include Cloud-Edge computing, network security, secure and dependable machine learning. She is a member of IEEE.

Jelena Mišić is Professor of Computer Science at Ryerson University in Toronto, Ontario, Canada. She has published over 120 papers in archival journals and close to 200 papers at international conferences in the areas of wireless networks, in particular wireless personal area network and wireless sensor network protocols, performance evaluation, and security. She serves on editorial boards of IEEE Transactions on Vehicular Technology, Computer Networks, Ad hoc Networks, Security and Communication Networks, Ad Hoc & Sensor Wireless Networks, Int. Journal of Sensor Networks, and Int. Journal of Telemedicine and Applications. She is a Fellow of IEEE and Member of ACM.

Vojislav B. Mišić is Professor of Computer Science at Ryerson University in Toronto, Ontario, Canada. He received his PhD in Computer Science from University of Belgrade, Serbia, in 1993. His research interests include performance evaluation of wireless networks and systems and software engineering. He has authored or co-authored six books, 20 book chapters, and over 280 papers in archival journals and at prestigious international conferences. He serves on the editorial boards of IEEE transactions on Cloud Computing, Ad hoc Networks, Peer-to-Peer Networks and Applications, and International Journal of Parallel, Emergent and Distributed Systems. He is a Senior Member of IEEE and member of ACM.

Jianhua Wang received the B.S. and M.S. degrees in Software engineering from the Taiyuan University of Technology in 2017 and 2020. He now pursues his Ph.D. Degree at Beijing Jiaotong University, majoring in Cyberspace Security. His research interests include adversarial machine learning and federated learning.

Haoran Zhu is currently a Ph.D. student in Cyberspace Security at Beijing Key Laboratory of Security and Privacy in Intelligent Transportation, Beijing Jiaotong University, China. His interests include blockchain security and data transmission security.