An Efficient FHE Radix-2 Addition Algorithm in BGV Scheme

Zongsheng Hou\textsuperscript{1,2}, Neng Zhang\textsuperscript{1}, Leibo Liu\textsuperscript{1}

\textsuperscript{1}Institution of Microelectronics, Tsinghua University, Beijing China, 100084
\textsuperscript{2}hzs18@tsinghua.org.cn

Abstract. Fully homomorphic encryption (FHE) is a novel encryption method that can perform operations on encrypted data. The performance of applications based on FHE is still low due to the high computational complexity of operations on the ciphertext. In this paper, we proposed an efficient FHE radix-2 addition algorithm in BGV scheme, called Optimized Cheon’s SIMD Adder. It is an optimization of Cheon’s SIMD Adder. The proposed adder uses the previously calculated results as much as possible to calculate the new result, so that it can keep the circuit depth as $O(l\log(\mu))$ and reduce the number of homomorphic multiplications. We also compared the weighted computational complexity (WCC) between this work and Cheon’s. The WCC of this work is only $O(\mu)$, while Cheon’s is $O(\mu \cdot l\log(\mu))$. At the end of the paper, we implemented the proposed addition, which improves the calculation speedup by 1.3 $\sim$ 1.4 times for 8 $\sim$ 512-bit adders compared with Cheon’s SIMD Adder.

1. Introduction

Homomorphic encryption is a new type of encryption system that can perform certain operations on data without decrypting the information. Since the calculation process is all performed in ciphertext, homomorphism can greatly reduce the risk of information leakage [1][2][3][4]. This concept was first proposed by Rivest et al.[5] in 1987, and its basic idea is to use only $E(x)$ and $E(y)$ to achieve the effect of calculating $E(x + y)$ and $E(x \cdot y)$. A scheme that can calculate both homomorphic addition, denoted as $A$, and homomorphic multiplication, denoted as $M$, is called a fully homomorphic encryption (FHE) scheme. Gentry [6] proposed the first viable fully homomorphic algorithm based on the ideal lattice problem in 2009. Since then, research on FHE schemes has been continued.

Following the Gentry’s initial scheme, there are several others have been developed. Van Dijk et al. [7] proposed a fully homomorphic scheme based on integer technology, but because the length of the public key is too long, the efficiency is greatly reduced. In order to solve this problem, Coron et al. and Smart, Vercauteren et al. proposed new FHE schemes, BV11a [8] and BV11b [9], based on learning with errors (LWE) problem and Ring-LWE problem, respectively.

The slow key generation process of the Smart-Vercauteren system (BV11b) [9] was then addressed in the paper by Gentry and Halevi [10]. Starting from the BV scheme of Brakerski and Vaikuntanathan [8], [9], more schemes started based on the LWE/RLWE problem. Since then, second-generation fully homomorphic encryption schemes have appeared. Following the BV11b scheme, Brakerski, Gentry, and Vaikuntanathan proposed a leveled FHE scheme (BGV12) with significantly improved performance in 2012 [11]. In order to use the BGV scheme more conveniently, Halevi et al. [10] developed the Helib library based on it. Similar to Helib, Microsoft proposed the SEAL library [12] based on the somewhat homomorphic encryption (SWHE) schemes. In 2012, Brakerski et al. [13] proposed an FHE scheme...
(Bra12) based on LWE without module switching, which can control noise well at the same time. The LWE-based homomorphic encryption scheme has the problem of ciphertext expansion during the multiplication process. Thus, a method called “relinearization” which can reduce the ciphertext size to its original size is proposed by H. Chen et al. [14].

Nowadays, some other FHE schemes, which are called the third-generation fully homomorphic encryption schemes, are proposed to further improve the performance. Gentry et al. [15] designed the GSW13, an SWHE scheme based on approximate eigenvector in 2013, to solve the ciphertext expansion and relinearization problems. Following the GSW13, Sheriff et al. [16] proposed a two-layer FHE scheme in 2014 which has less noise and higher efficiency than BV [17] algorithm above. In 2015, another advanced scheme, FWHE, which optimizes the bootstrapping time within one second was proposed by Ducas and Micciancio [18]. One year later, the TFHE scheme was constructed by Chillotti et al. [19] which obtains a speed up from less than 1 s to less than 0.1 s. In 2017, the authors of TFHE improved the scheme to a new version [13], whose bootstrapping time is less than 0.013s per gate.

The above-mentioned FHE schemes can be divided into two categories [20]. One is FHE with a bootstrapping process (GSW-like: GSW, FHEW, TFHE, suitable for radix-2 arithmetic operations), and the other is leveled FHE with predefined circuit depth (BGV-like: BGV, BFV, suitable for word-wise operations), which is mainly determined by the depth of homomorphic multiplications. However, when we only use a certain type of fully homomorphic encryption scheme (such as BGV-like or GSW-like), we cannot efficiently implement arithmetic operations and Boolean logic operations at the same time. Therefore, calculation examples such as the ARX encryption algorithm that have both arithmetic operations and Boolean logic operations cannot be efficiently implemented in a fully homomorphic scheme.

In order to solve this problem, there are currently two main solutions: The first is to propose a new conversion scheme between the BGV-like and GSW-like fully homomorphic encryption schemes to reduce the conversion overhead between the two schemes as much as possible. The other is to use the radix-2 encoding method in BGV-like schemes, which improves the efficiency of radix-2 arithmetic operations by changing the calculation rules and steps. Since the radix-2 encoding supports Boolean logic operations, the second solution can also achieve more efficient arithmetic operations and Boolean logic operations at the same time.

The representative scheme of the first solution is the CHIMERA and PEGASUS conversion schemes proposed in 2020. Both of these two schemes can switch freely between different types of fully homomorphic schemes, so as to efficiently implement two different types of operations in the same environment. However, such a conversion requires a redesign of the computing environment, and the calculation is more complicated.

The second solution is simpler than the former. This paper also follows this solution. The difficulty of this solution is to improve the efficiency of radix-2 arithmetic operations. Here are some related works about this difficulty. To implement the search-and-compute query algorithms, J. H. Cheon et al. [21] designed a series of radix-2 circuit primitives, such as the equality circuit, comparison circuit, and integer addition circuit, to achieve the clauses in SQL statements. [21] also proposed optimized circuit primitives combined with single-instruction multiple-data (SIMD). However, the carry chain and complex algorithms cannot make full use of the SIMD, and the performance of integer addition is still low compared with the equality and comparison circuits. To improve the performance of integer addition, Cheon et al. changed the message space from $2^1$ to $2^{14}$, which greatly improved the performance of integer addition. However, the running time of equality and comparison increased by nearly 200 times and 35 times at 10 bits, respectively. This method does not solve the problem of performance mismatch between logic operations and arithmetic operations. Zhang et al. [22] proposed a weighted computational complexity model to build the relationship between circuit depth and computational complexity. By using this model, they moved the dot-multiplication’s position of the carry-lookahead adder (CLA) and improved the operational efficiency. Qin et al. [23] not only implemented the original CLA but also used the SIMD method to accelerate the calculation. Qin also used the dot-multiplication
method, which is similar to Zhang’s. But the computational complexity of Qin’s SIMD CLA increases significantly when the bit width increases.

2. Background

2.1. Notations

The parameters in this paper are denoted as follows. \( p \) represents the plaintext modulus, and \( p = 2 \) in this paper; \( n \) stands for the degree of the polynomial modulus \( (x^n + 1) \); \( q \) denotes the coefficient modulus; \( L \) stands for the depth of the circuit; and \( \mu \) represent the bit width of the ciphertext addition operands.

2.2. BGV Scheme

The BGV scheme is very widely used. The comparative work in this paper uses this scheme. Therefore, this paper also selects the BGV scheme. We will briefly introduce this scheme here. It consists of four primitive functions:

2.2.1. Setup and KeyGen.

A ladder of parameters are obtained from a security parameter \( \lambda \) and a number of levels \( L \). In the ladder, there is a series of module \( p_L > p_{L-1} > \ldots > p_0 \). For each level \( l \), the KeyGen will generate a secret key \( s_l \subseteq \mathbb{R}^{p_l} \), a public key \( A_l \subseteq \mathbb{R}^{p_l \times 2} \), where \( N = \text{polylog}(p_l) \), \( s_l \otimes s_l \) and a switch key.

2.2.2. Encrypt.

A message \( m \) in the plaintext space \( \mathbb{R}_t \) where \( t \geq 2 \) is encrypted to a ciphertext \( c \). The encryption formula is

\[
    c = m + r^T \cdot A_l \subseteq \mathbb{R}^{p_l^2}, \quad \text{where} \quad r \text{ is a column vector with small coefficients sampled from } \mathbb{R}^N, \quad \text{and} \quad m = (m, 0).
\]

2.2.3. Decrypt.

At level \( l \), ciphertext \( c \) is decrypted as \( p = [\langle c, s \rangle]_{p_l} \) under the secret key \( s_l \).

2.2.4. Evaluate.

BGV scheme can support both homomorphic addition and multiplication. Addition is calculated as \( c_1 + c_2 \), and multiplication is calculated as \( \langle c_1 \otimes c_2, s_l \otimes s_l \rangle \). To reduce the noise and move to another level, a “refresh” process consisting of relinearization and modules switching is used after the multiplication. And in Boolean expressions, homomorphic addition and multiplication are represented as XOR and AND respectively.

2.3. Weighted Computational Complexity Model

Due to the native computational complexity (NCC) of the previous evaluation models cannot reflect the differences of the homomorphic multiplications at different levels, Zhang et al. [22] proposed a weighted computational complexity (WCC) model to address the issue. The number, depth, and distribution of homomorphic multiplications are considered together in the model for the first time. This model can more accurately measure the computational efficiency of leveled fully homomorphic encryption. For an evaluation with the \( L \) circuit depth, the calculation of WCC is:

\[
    WCC = \sum_{l=0}^{L-1} W_l \times N_l. \tag{1}
\]

where \( W_l \) and \( N_l \) denote the weight and the number of the homomorphic multiplication at level \( l \). The weight \( W_l \) is simply set to be \( l + 1 \), which is related to the multiplications of polynomials, modulus-switching and key-switching.

2.4. Single-instruction Multiple-data (SIMD)

The FHE scheme uses polynomials of a high degree, which is up to 2048 for a 256-bit adder as an example. The scalar encoded data using only one of the 2048 coefficients, although all homomorphic operations (additions and multiplications) act on all 2048 coefficient polynomials. As a result, the homomorphic operations take a long time due to the high degree, and only one significant coefficient is
calculated. To make full use of parallelism, Smart and Vercauteren [24] first noted that FHE schemes can support SIMD operations on finite fields of characteristic two by choosing appropriate parameters in certain FHE schemes. The SIMD encoding technique is based on the reverse process of the Chinese remainder theorem (CRT). By using SIMD, multiple bits can be packed into a single ciphertext with slots. In this way, parallel operations can be achieved. In this paper, \( W_M \), \( W_A \), and \( W_L \) denote three SIMD operations: homomorphic SIMD multiplication, SIMD addition, and SIMD shift operations, respectively.

2.5. Addition Circuits
The straightforward way to implement a \( \mu \)-bit addition is defined as Equation 2-4:

\[
add(x, y) = (s_0, s_1, \ldots, s_{\mu-1}).
\]  

(2)

The sums of the addition are:

\[
s_i = x_i \oplus y_i \oplus c_{i-1}.
\]  

(3)

The carry-outs of the addition are:

\[
c_i = (x_i \otimes y_i) \oplus ((x_i \oplus y_i) \otimes c_{i-1}).
\]  

(4)

where \( i \in [1, \mu - 1] \) with initial values \( s_0 = x_0 \oplus y_0 \) and \( c_0 = x_0 \otimes y_0 \). There are two main normal addition circuits. The first is the ripple-carry adder (RCA), which is to carry out calculations in turn from LSB (least significant bit) to MSB (most significant bit). According to the BGV scheme, for a \( \mu \)-bit addition, the number of homomorphic multiplications is \( 2\mu - 1 \), and the depth of the circuit is \( \mu \). The second addition is the carry-lookahead addition (CLA), which uses the dot-multiplication to reduce the circuit depth of addition from \( \mu + 1 \) to \( \log(\mu) + 1 \). Taking the Ladner-Fischer CLA as an example, the number of homomorphic multiplications is \( (\mu + \mu \log(\mu)) \), and the depth of the circuit is only \( \log(\mu) + 1 \).

3. Optimized Cheon’s SIMD Adder
3.1. WCC of the Cheon’s SIMD Adder
The main idea of Cheon’s SIMD Adder is to express each sum and carry of the adder in closed form and minimize the number of homomorphic operations using SIMD operations. Cheon et al. rewritten the \( s_i \)s as follows:

\[
s_i = x_i \oplus y_i \oplus \sum_{j=0}^{i-1} t_{i,j},
\]  

(5)

where \( t_{ij} = (x_i \otimes y_i) \otimes \prod_{k=j+1}^{i-1} (x_k \oplus y_k) \) for \( j < i - 1 \) and \( t_{i,i-1} = x_{i-1} \otimes y_{i-1} \). As mentioned in [21], there are \( (\mu - 2) \) homomorphic multiplications when \( i = \mu - 1 \) and \( j = 0 \). Thus, the circuit depth of Cheon’s SIMD Adder is \( \log(\mu - 2) + 1 \), which also considers one circuit depth for \( x_i \otimes y_i \).

According to the WCC model, the WCC of an 8-bit Cheon’s SIMD Adder can be calculated as follows. There are \( (3\mu - 5) = 19 \) homomorphic multiplications in a circuit with depth 4. For level 0, \( (\mu - 2) = 6 \) homomorphic multiplications are used to calculate the \( (x_i \otimes y_i) \otimes \prod \ldots \), which introduces a WCC \( O(\mu) \). For level 1-3, \( (2\mu - 4) = 12 \) homomorphic multiplications are used to calculate the \( \prod_{k=j+1}^{i-1} (x_k \oplus y_k) \), which introduces another WCC \( O(\mu \cdot \log(\mu)) \). At the same time, \( x_i \otimes y_i \) also needs one homomorphic multiplication at level 1, which has a WCC \( O(1) \). Thus, the total WCC of the Cheon’s SIMD Adder is \( O(\mu \cdot \log(\mu)) \). And we also give the specific WCCs of Cheon’s SIMD Adder whose bit width is 8/16/32 bits in Section 3.4.

3.2. Optimized Cheon’s SIMD Adder
In this section, we improve Cheon’s SIMD adder and reduce the number of multiplications from \((3\mu - 5)\) to \((3\mu/2 - 3)\). The specific improvement method is shown in Figure 1-2, which is an 8-bit SIMD adder.

In step 1 of the Optimized Cheon’s SIMD Adder, we first use \(x_i \oplus y_i\) and \(x_i \otimes y_i\) to compute the homomorphic SIMD addition and multiplication results, namely \(C_i\) and \(D_i\). Figure 1 is an example of step 1.

![Figure 1](image1.png)

**Figure 1.** Step 1 of the Optimized Cheon’s SIMD Adder.

In step 2 of the Optimized Cheon’s SIMD Adder, we use the \(C_L\) and \(C_M\) operations to calculate the \(\sum_{j=0}^{i-1} t_{i,j}\). We define two kinds of intermediate variables \(IC(m)\) (a \(\mu\) bit SIMD data, \(m \in [1, \mu/2 - 1]\)) and \(ID(n)\) (a \(\mu\) bit SIMD data, \(n \in [1, \mu - 1]\)). By homomorphic addition to all \(ID(n)s\), we can calculate the value of each bit \(\sum_{j=0}^{i-1} t_{i,j}\) parallelly. For the \(i\)-th position, \(\sum_{j=0}^{i-1} t_{i,j} = \sum_{j=0}^{i-1} (D_j \otimes \prod_{k=j+1}^{i-1} C_k) = \sum_{j=0}^{i-1} D_j \otimes C_{j+1} \otimes C_{j+2} \otimes \cdots \otimes C_{i-1}\). Figure 2 takes 8 bits as an example.

![Figure 2](image2.png)

**Figure 2.** Step 2 of the Optimized Cheon’s SIMD Adder.

In the above figure, each row represents the \(i\)-th bit, and the 8-bit data in each rectangle constitutes a series of intermediate variables, namely \(IC(m)\) and \(ID(n)\). By summing up \(ID(1) \oplus ID(\mu - 1)\) (namely, \(\sum_{n=0}^{\mu-1} ID(n) = ID(0) \oplus ID(1) \oplus \cdots \oplus ID(\mu - 1)\)), all bits of \(\sum_{j=0}^{i-1} t_{i,j}\) can be calculated parallelly. We give the generalized formulas for \(IC(m)\) and \(ID(n)\) here.

\[
IC(m) = \begin{cases} 
\{C_7, 0\}^1 & m = 1 \\
IC(m/2) \otimes IC(m/2)^{m/2} & m = \text{Power of 2} \\
IC(p) \otimes IC(q)^p & m \neq \text{Power of 2}
\end{cases}
\]

where \(m \in [1, \mu/2 - 1], \{C_7, 0\}^1 = \{C_6, C_5, \ldots, C_1, 0, 0\}, q\) is the largest integer less than a power of 2, \(p + q = m\).
\[ ID(n) = \begin{cases} 
D^1(n) & n = 1 \\
IC(n/2) \otimes ID(n/2)^{n/2} & n = \text{Power of 2} \\
IC(s) \otimes ID(t)^s & n \neq \text{Power of 2}
\end{cases} \]

where \( n \in [1, \mu - 1] \), \( t \) is the largest integer less than a power of 2, \( s + t = n \).

For a \( \mu \) bit addition, \((\mu - 2) ID(n)s\) and \((\mu/2 - 2) IC(m)s\) are needed to calculate \( t_{ij} \). Therefore, in the second step, \((3\mu/2 - 4)\) homomorphic multiplications are required, and Cheon’s SIMD Adder requires \((3\mu - 6)\) homomorphic multiplications.

In step 3, we perform homomorphic addition operations on \( \sum_{n=0}^{\mu-1} ID(n) \) and \( C \) and obtain the final results of the Optimized Cheon’s SIMD Adder.

Different from Cheon’s work, the method proposed in this paper uses the previously calculated results as much as possible to calculate the new result, so that it can keep the circuit depth as \( O(\log(\mu)) \) and reduce the number of homomorphic multiplications as much as possible.

### 3.3. WCC of Optimized Cheon’s SIMD Adder

Next, we will calculate the WCC of Optimized Cheon’s SIMD Adder. At level 0, \((\mu/2 - 1)\) homomorphic multiplications are used to calculate the \( ID(\mu/2 + 1) \sim ID(\mu - 1) \). At level \( i = 1 \), there are \((\mu/2^{i+1})\) homomorphic multiplications for \( ID(n) \) and \((\mu/2^{i+1} - 1)\) homomorphic multiplications for \( IC(m) \). For level \( i \in [2, \log(\mu) - 1] \), there are \((\mu/2^{i+1})\) homomorphic multiplications for \( ID(n) \) and \((\mu/2^{i+1})\) homomorphic multiplications for \( IC(m) \). At the same time, \( x_i \otimes y_i \) also needs one homomorphic multiplication at level \( \log(\mu) \). Thus, the total WCC of Optimized Cheon’s SIMD Adder can be calculated as follows:

\[ WCC = (\mu/2 - 1) + \sum_{i=1}^{\log(\mu)-1} ((i + 1) \times (\mu/2^i)) + 2 \times (\mu/2 - 1) + O(\log(\mu)) = O(\mu)(6) \]

The proposed adder reduces the WCC from \( O(\mu \cdot \log(\mu)) \) to \( O(\mu) \).

### 3.4. Comparison between different additions in FHE

Table 1 shows the comparison between different additions in FHE mentioned above. Note that, according to [25], a 0.5 circuit depth for shift or rotation operations is added in SIMD cases. Thus, the circuit depth of the SIMD cases is slightly deeper. But because of the advantages in the number of operations, the WCCs of SIMD cases are still smaller than non-SIMD cases. Compared with this work, Qin’s SIMD Adder and Cheon’s SIMD Adder both have the same circuit depth, but the larger number of operations and WCC.

**Table 1. Comparison between different additions in FHE.**

| Adders                  | SIMD    | Circuit depth | WCC⁶ [23]          | Number of operations |
|-------------------------|---------|---------------|--------------------|---------------------|
| Cheon’s Naive Adder [21]| NO      | \( \mu + 1 \) | \( x \)            | \((\mu^3 - 3\mu^2 + 8\mu)/6\)M + \( x \)A⁵        |
| Ripple-Carry Adder      | NO      | \( \mu + 1 \) | \( O(\mu^2) \)     | \((2\mu - 1)M + (4\mu - 3)A\)                     |
| Ladner-Fischer CLA [26] | NO      | \( \log(\mu) + 1 \) | \( O(\mu \cdot \log(\mu)^2) \) | \((\mu \log(\mu) + \mu)M + (0.5\mu \log(\mu) + 3\mu)A\) |
| Zhang’s CLA [22]       | NO      | \( \log(\mu) + 1 \) | \( O(\mu \cdot \log(\mu)) \) | \((\mu \log(\mu) + \mu)M + (0.5\mu \log(\mu) + 3\mu)A\) |
| Qin’s SIMD Adder [23]  | YES     | \( \log(\mu) + 1.5^c \) | \( O(\mu \cdot \log(\mu)^2) \) | \((\mu + 1)\log(\mu)\)C_M + \((\mu + 4 \log(\mu)/2)C_A\) |
| Cheon’s SIMD Adder [21] | YES     | \( \log(\mu) + 1.5^c \) | \( O(\mu \cdot \log(\mu)) \) | \((3\mu - 5)C_M + \mu C_A\) |
| This work              | YES     | \( \log(\mu) + 1.5^c \) | \( O(\mu) \)        | \((3\mu/2 - 3)C_M + (\mu + 1)C_A\) |

⁶ WCC stands for the weighted computational complexity proposed in [22].

\( x \) means the data are not given.

⁵ Cheon’s and Qin’s paper, the circuit depth is related only to homomorphic multiplication. However, according to [25], a 0.5 circuit depth for shift or rotation operations is added here.
In order to show the improvement effect of this work on Cheon’s SIMD Adder more clearly, we take the additions of 8/16/32 bits as three examples to analyze WCCs in detail. It can be seen from Table 2 that the number of homomorphic SIMD multiplications and WCC of the method proposed in this paper is much smaller than that of Cheon’s. When the bit width is larger, the advantage of this paper is more obvious.

| LEVEL | Bit width | 0 | 1 | 2 | 3 | 4 | 5 | Total Number | WCC [22] |
|-------|-----------|---|---|---|---|---|---|--------------|----------|
|       | 8         | 6\textsuperscript{a} | 7 | 4 | 2 | - | - | 19 | 40 |
|       | 16        | 14 | 15 | 8 | 4 | 2 | - | 43 | 229 |
|       | 32        | 30 | 31 | 19 | 7 | 3 | 1 | 91 | 1071 |

\textsuperscript{a}Number of \(c_M\) at level 0.

### Table 2. Comparison of Number of \(c_M\) between Cheon’s and Optimized Cheon’s Adder.

**4. Experimental results**

All experiments in this paper are based on the following experiment details. This study uses a computer equipped with an Intel Core i7-6700K 4.0 GHz processor with 32 GB of main memory running a Linux 2.6.32 operating system. The experiments use the BGV scheme with Shoup’s NTL library and Shoup-Halevi’s HE library. The security level, Hamming weight of the secret key, and the number of columns in the key switching matrix keep the default values of 80, 64, and 2, respectively. The context of the library, such as the polynomial degree, special modulus, and ciphertext modulus, are automatically calculated by the library functions \(\text{FindM()}\) and \(\text{buildModChain()}\). For SIMD cases, the minimum number of slots is set to be the bit width of the operand. This paper uses the system function \(\text{gettimeofday()}\) and /usr/bin/time to measure the running time and occupied memory and reports the arithmetic mean of multi-execution, i.e., 2~10 times according to the running time. Note that the running times required for data encryption and decryption are excluded from the results.

**Table 3. Running time (s) of the different additions in FHE.**

| Adders\textsuperscript{d} | 4   | 8   | 16  | 32  | 64  | 128 | 256  | 512  |
|---------------------------|-----|-----|-----|-----|-----|-----|------|------|
| Ripple-Carry Adder       | 0.124 | 0.758 | 5.706 | 24.084 | 173.117 | -\textsuperscript{a} | - | - |
| Ladner-Fischer CLA [26]  | 0.188 | 0.588 | 1.732 | 3.680 | 20.243 | 46.845 | 135.932 | 324.915 |
| Zhang’s OptCLA [22]      | 0.176 | 0.582 | 1.667 | 3.527 | 18.416 | 41.474 | 113.666 | 268.768 |
| Qin’s SIMD Adder [23]    | 0.401 | 1.039 | 2.484 | 3.132 | 13.919 | x\textsuperscript{c} | x | x |
| Cheon’s SIMD Adder [21]  | 0.020 | 0.276 | 0.882 | 3.904 | 10.072 | 23.772 | 59.815 | 230.289 |
| This work                | 0.023 | 0.210 | 0.645 | 2.732 | 6.974 | 16.560 | 42.351 | 163.711 |
| Speedup                  | 0.87  | 1.31 | 1.37 | 1.15 | 1.44 | 1.44 | 1.41 | 1.41 |

\textsuperscript{a}The complexity of the RCA in 128/256/512 bits is too large to be computed. Thus, we didn’t give these experimental results.

\[3.132\] means the fastest running time of the related works.

\[x\] means the data are not given.

\textsuperscript{d}Line 2-4 is the running time of Adders without SIMD technique. Line 5-6 is the running time of Adders with SIMD technique.
Table 3 shows the running time of the different additions in FHE. We enumerate the performance of several related works here, including non-SIMD implementations and SIMD implementations. The last line of the table is the speedup of this work compared to other circuits. The speedup is calculated as \( \frac{A}{B} \), where \( B \) is the running time of the proposed circuit, and \( A \) is the running time of the circuit that is the fastest among the other circuits. We can see that the SIMD implementations have higher performance than the non-SIMD implementation. And as the bit width increases, the advantages of SIMD become more and more obvious. In the SIMD cases, when the bit width is greater than 4, the performance of this work is the highest. It can be seen from the experimental results that using the Optimized Cheon’s SIMD Adder generally improves the calculation speedup by 1.3 ~ 1.4 times for 8 ~ 512-bit adders compared with Cheon’s SIMD Adder.

5. Conclusion
To achieve an efficient FHE radix-2 addition algorithm in BGV scheme, this paper proposed the Optimized Cheon’s SIMD Adder. It greatly reduces the number of homomorphic SIMD multiplications without increasing the circuit depth, thereby reducing the WCC from \( O(\mu \cdot \log(\mu)) \) to \( O(\mu) \). The Optimized Cheon’s SIMD Adder generally improves the calculation speedup by 1.3 ~ 1.4 times for 8 ~ 512-bit adders compared with Cheon’s SIMD Adder. There are three aspects of future work. First, the SIMD technique is particularly suitable for the BGV-like scheme. We will continue to explore the application of SIMD in the radix-2 field. Second, this paper uses the BGV scheme to demonstrate the idea of Optimized Cheon’s SIMD Adder. Future work concerns exploiting this work in other BGV-like schemes. At the same time, realizing the conversion of bit-wise and word-wise operations in the same scheme may tap the greater potential of BGV-like schemes. Third, even though this work improves the performance of arithmetic operations, the performance is still not at a practical level for real-world applications. GPUs and FPGAs could be used to further accelerate the operations.

6. References
[1] Zhao, Y., Yang, L. T., & Sun, J. (2018). A secure high-order cfs algorithm on clouds for industrial internet-of-things. IEEE Transactions on Industrial Informatics, 1-1.
[2] Gai, K., & Qiu, M. (2017). Blend arithmetic operations on tensor-based fully homomorphic encryption over real numbers. IEEE Transactions on Industrial Informatics, 1-1.
[3] Xing, K., Hu, C., Yu, J., Cheng, X., & Zhang, F. (2017). Mutual privacy preserving k-means clustering in social participatory sensing. IEEE Transactions on Industrial Informatics, 1-1.
[4] Hariss, K., Noura, H., & Samhat, A. E. (2017). Fully enhanced homomorphic encryption algorithm of more approach for real world applications. Journal of Information Security & Applications, S2214212616303052.
[5] Rivest, R. L., Adleman, L. M., & Dertouzos, M. L. (1978). On Data Banks and Privacy Homomorphisms. Foundations of Secure Computation.
[6] Gentry, & Craig. (2009). Fully homomorphic encryption using ideal lattices. Stoc, 169-178.
[7] Dijk, M. V., Gentry, C., Halevi, S., & Vaikuntanathan, V. (2010). Fully Homomorphic Encryption over the Integers. International Conference on Theory & Applications of Cryptographic Techniques. Springer, Berlin, Heidelberg.
[8] Coron, J. S., Mandal, A., D Na ac ace, & Tibouchi, M. (2011). Fully Homomorphic Encryption over the Integers with Shorter Public Keys. Conference on Advances in Cryptology. Springer, Berlin, Heidelberg.
[9] Smart, N. P., & Vercauteren, F. (2010). Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes. Springer Berlin Heidelberg.
[10] Halevi, S., & Shoup, V. (2014). Algorithms in helib. Lecture Notes in Computer Science, 8616, 554-571.
[11] Zvika, Brakerski, Craig, Gentry, Vinod, & Vaikuntanathan. (2014). (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory
(TOCT) - Special issue on innovations in theoretical computer science 2012 - Part II, 6(3), 1-36.

[12] Microsoft Research, Microsoft SEAL (v1.0), 2015, https://github.com/Microsoft/SEAL.

[13] Chillotti, I., Gama, N., Georgieva, M., & M Izabachène. (2017). Faster Packed Homomorphic Operations and Efficient Circuit Bootstrapping for TFHE. International Conference on the Theory and Application of Cryptology and Information Security. Springer, Cham.

[14] Hao, C., Laine, K., & Player, R. (2017). Simple Encrypted Arithmetic Library - SEAL v2.1. International Conference on Financial Cryptography and Data Security. Springer, Cham.

[15] Gentry, C., Sahai, A., & Waters, B. (2013). Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. Annual Cryptology Conference. Springer, Berlin, Heidelberg.

[16] Alperin-Sheriff, J., & Peikert, C. (2014). Faster Bootstrapping with Polynomial Error. International Cryptology Conference. Springer, Berlin, Heidelberg.

[17] ZVika BRAKERSKI†, & VINOD VAIKUNTHANATH‡. (2011). EFFICIENT FULLY HOMOMORPHIC ENCRYPTION FROM. IEEE Symposium on Foundations of Computer Science. IEEE.

[18] Ducas, L., & Micciancio, D. (2015). FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second. Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer Berlin Heidelberg.

[19] Chillotti, I., Gama, N., Georgieva, M., & M Izabachène. (2016). Faster fully homomorphic encryption: bootstrapping in less than 0.1 seconds. International Conference on the Theory and Application of Cryptology and Information Security.

[20] Brakerski, Z. (2012). Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. Springer Berlin Heidelberg.

[21] Cheon, J. H., Kim, M., & Kim, M. (2017). Optimized search-and-compute circuits and their application to query evaluation on encrypted data. IEEE Transactions on Information Forensics & Security, 11(1), 188-199.

[22] N. Zhang et al., (2020) Efficient Comparison and Addition for FHE with Weighted Computational Complexity Model. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, doi: 10.1109/TCAD.2020.3031266.

[23] Q. Qin, N. Zhang, L. Liu, S. Yin and S. Wei, (2019) Addition Circuit optimization Using Carry-Lookahead and SIMD for Homomorphic Encryption. IEEE International Conference on Electron Devices and Solid-State Circuits (EDSSC), Xi’an, China, pp. 1-3, doi:10.1109/EDSSC.2019.8753931.

[24] Vercauteren, F. (2010). Fully homomorphic encryption with relatively small key and ciphertext sizes. lecture notes in computer science.

[25] Mella, S., & Susella, R. (2013). On the Homomorphic Computation of Symmetric Cryptographic Primitives. M. Stam (Ed.), Cryptography and Coding (pp. 28 – 44). Springer Berlin Heidelberg.

[26] Ladner, R. E., & Fischer, M. J. (1980). Parallel prefix computation. Journal of the ACM, 27(4), 831-838.

Acknowledgments
This work was supported by the National Key Research and Development Program of China under Grant 2018YFB2202101. We also thank the editors and reviewers for their valuable comments.