Accelerating Homomorphic Comparison Operations for Thresholding Using an Asymmetric Input Range and Input Scaling

Sunwoong Kim
Rochester Institute of Technology
Rochester, New York, USA
sskeme@rit.edu

Wonhee Cho
Seoul National University
Seoul, South Korea
wony0404@snu.ac.kr

ABSTRACT

In a cyber-physical system (CPS), the interconnection of cyber and physical components occurs through a network. This structure, particularly cyber components and networks, makes it susceptible to malicious attacks. One of the solutions to this CPS security issue is to employ end-to-end homomorphic encryption (HE) that allows direct computations on encrypted data. Despite its promise, HE only supports basic operations, such as addition and multiplication, which limits its application areas. Numerical methods have been presented to perform a comparison operation in the HE domain. However, they suffer from a slow processing speed due to an inherently high number of iterations. To accelerate a homomorphic comparison operation, this paper introduces a novel approach that scales inputs using an asymmetric input range in thresholding. Additionally, parallelism in HE-based multilevel thresholding is explored and exploited through the use of a parallel processing application programming interface for further acceleration. Compared to a previous comparison operation method, the proposed method achieves comparable accuracy with fewer iterations, resulting in a 48% reduction in execution time on an edge computing device. Furthermore, employing an additional thread using parallelism increases this reduction to 63%.

CCS CONCEPTS

• **Security and privacy → Cryptography**.

KEYWORDS

cyber-physical system; homomorphic encryption; multithreading; numerical method; security; thresholding

ACM Reference Format:
Sunwoong Kim and Wonhee Cho. 2024. Accelerating Homomorphic Comparison Operations for Thresholding Using an Asymmetric Input Range and Input Scaling. In *Great Lakes Symposium on VLSI 2024 (GLSVLSI '24), June 12–14, 2024, Clearwater, FL, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3649476.3658724

1 INTRODUCTION

A cyber-physical system (CPS) typically consists of cyber components such as cloud servers and edge computing devices for information processing, physical components like sensors and actuators, and communications between cyber and physical components. However, its openness and connectivity make it susceptible to malicious attacks [21]. Therefore, extensive research has been conducted on enhancing privacy and security for CPS. Implementing general encryption and decryption techniques in CPS requires the decryption of encrypted data on cyber components, leaving secret keys and decrypted data vulnerable to potential attackers.

One of the solutions to address this challenge is employing homomorphic encryption (HE). This technique enables direct computation of encrypted data without the need for decryption [7]. Therefore, there is no need to store secret keys on cyber components, and sensitive information remains protected from attackers. Compared to alternative cryptography techniques, HE requires less data transmissions between cyber and physical components. Due to these capabilities, HE has found extensive application across various real-world domains, including machine learning [15], genomics research [20, 22], managing infrastructures [3], password processing [9], speech processing [11], and image processing [19, 24, 26].

HE schemes are categorized into two types: bit-wise and word-wise schemes. Typically, word-wise schemes exhibit better performance in arithmetic operations over large-scale encrypted data than bit-wise schemes [17]. Within word-wise HE schemes, there is a further classification based on the data type of plaintext data. For instance, the BFV scheme is tailored for plaintext integers [12], whereas the CKKS scheme, which is used in this paper, is designed for plaintext real/complex numbers [5].

Despite its promise, HE encounters several issues that hinder its practical application. One significant challenge is the limited types of operations over encrypted data. Although addition, subtraction, and multiplication are commonly supported in many word-wise HE schemes, the requirements of numerous real-world applications extend beyond these basic operations. Specifically, one of the frequently used operations is a comparison operation. To employ this operation in the HE domain, numerical methods have been devised to approximate a comparison operation using additions, subtractions, and multiplications [6, 8].

HE hides plaintext data using noise, and the level of noise increases with each operation over a ciphertext (also called homomorphic operation). When this level exceeds a certain threshold, decryption fails to produce correct results. In particular, homomorphic multiplication, considerably slower than homomorphic addition and subtraction, significantly amplifies noise levels. Therefore, the number of homomorphic multiplications is defined as *depth* and carefully managed. The numerical methods for a comparison operation have iterations, each involving multiple multiplications, thus leading to slow processing speed and depth-related challenges.

GLSVLSI '24, June 12–14, 2024, Clearwater, FL, USA

ACM ISBN 979-8-4007-0605-9/24/06.
https://doi.org/10.1145/3649476.3658724

Several studies have explored efficient approximate comparison operations in the HE domain [23, 26]. However, the research remains insufficient. Specifically, there is a scarcity of studies focusing on lightweight methods tailored to accelerate approximate comparison operations by leveraging application-specific features. Such approaches are particularly advantageous for cyber components constrained by limited hardware resources and network bandwidth.

The contributions of this paper are as follows:

- We modify inputs of a numerical method for a comparison operation used in thresholding. In particular, an asymmetric input range characteristic is used to reduce the required iterations of a numerical method. It leads to a reduction in the execution time and a decrease in depth consumption while achieving comparable accuracy.
- We explore parallelism in multilevel thresholding that involves comparison operations for further acceleration. To exploit this parallelism, a parallel processing application programming interface is utilized.
- We evaluate the HE-based 3-level thresholding with our proposed methods, in terms of depth, accuracy, and execution time, on a workstation and an edge computing device. Our proposed methods improve the feasibility of integrating HE into real-time CPS.

2 BACKGROUND

2.1 Homomorphic Encryption

A word-wise HE scheme consists of the following operations:

- KeyGen(λ): takes a security parameter λ and generates a secret key sk, a public key pk, and an evaluation key evk.
- Enc(μ, pk): encrypts a plaintext message μ into a ciphertext ct using a provided public key.
- Dec(ct, sk): decrypts a ciphertext ct into a plaintext message using a secret key.
- HomAdd($\mathtt{ct}_1$, $\mathtt{ct}_2$, evk): performs addition between ciphertexts $\mathtt{ct}_1$ and $\mathtt{ct}_2$ of messages μ_1 and μ_2 using an evaluation key and produces a ciphertext of a message $\mu_1 + \mu_2$.
- HomMul($\mathtt{ct}_1$, $\mathtt{ct}_2$, evk): performs multiplication between ciphertexts $\mathtt{ct}_1$ and $\mathtt{ct}_2$ of messages μ_1 and μ_2 using an evaluation key and produces a ciphertext of a message $\mu_1 \cdot \mu_2$.

Besides, subtraction of ciphertexts is available through a variant of HomAdd, and one of the operands in homomorphic operations can be a plaintext message (e.g., homomorphic multiplication between a ciphertext and a plaintext constant). These operations are implemented in open-source libraries, such as Microsoft SEAL [25], EPFL Lattigo [1], Zama TFHE [27], and OpenFHE [2].

Many word-wise HE schemes support encoding with packing techniques, which include multiple plaintext messages into a single ciphertext, to improve processing speed. For example, the CKKS scheme allows $N/2$ messages within a ciphertext when the polynomial degree, which is a parameter of this scheme, is set to N. A homomorphic operation on a ciphertext processes multiple messages in a single instruction/multiple data manner.

With each homomorphic operation, the magnitude of noise grows. Bootstrapping is a procedure to refresh this noise, and HE using this technique is called fully HE [14]. However, due to its significant computational overhead, bootstrapping proves impractical for numerous real-time CPS. This paper, in alignment with this consideration, opts against the use of bootstrapping, focusing instead on minimizing total depth.

Algorithm 1 Comp($x, y; n, d$) [8]

Input: normalized real numbers $x, y \in [0, 1]$
Input: the number of iterations $n, d \in \mathbb{N}$
Output: approximate 1 if $x > y$, 0 if $x < y$, and 1/2 otherwise

```
a ← x − y
for (i = 1; i ≤ d; i = i + 1) do
    a ← f_n(a)
end for
return (a + 1)/2
```

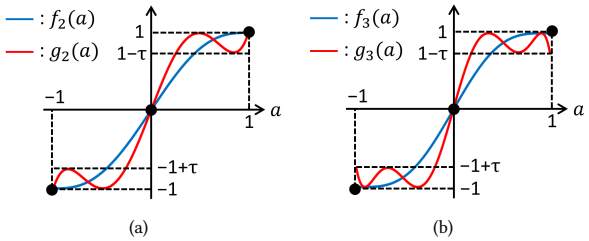

Figure 1: Two types of approximate sign functions used in Comp [8]. (a) when $n = 2$ (b) when $n = 3$.

2.2 Homomorphic Comparison Operation

To perform logical and non-polynomial operations, which are not basic operations supported in word-based HE schemes, several studies have adopted numerical methods. Specifically, Cheon *et al.* presented a numerical method for comparison operation [8]. Algorithm 1 shows the approximate comparison operation referred to as Comp. It involves a nested loop. The outer loop has d iterations, and the inner loop corresponds to an approximate sign function f_n with n iterations ($f_n(a) \approx \frac{a}{|a|}$). The definition of $f_n(a)$, which is an increasing function in [-1, 1], is as follows:

$$f_n(a) = \sum_{j=0}^{n} \frac{1}{4^j} \cdot \binom{2j}{j} \cdot a(1 - a^2)^j. \tag{1}$$

In this previous work, a mixed composition $f \circ \cdots \circ f \circ g$ is introduced as a replacement for $f \circ \cdots \circ f \circ f$ to accelerate convergence. The polynomial function g_n exhibits a sharper slope than f_n, but its function values oscillate within the intervals $[1 - \tau, 1]$ and $[-1, -1 + \tau]$, where τ is a small constant. Therefore, g_n is employed during the first iteration ($i = 1$), with f_n taking over thereafter ($i > 1$). $g_n(a)$'s with τ of 1/4 introduced in [8] are as follows:

- $g_1(a) = -\frac{1359}{2^{10}} \cdot a^3 + \frac{2126}{2^{10}} \cdot a$
- $g_2(a) = \frac{3796}{2^{10}} \cdot a^5 - \frac{6108}{2^{10}} \cdot a^3 + \frac{3334}{2^{10}} \cdot a$
- $g_3(a) = -\frac{12860}{2^{10}} \cdot a^7 + \frac{25614}{2^{10}} \cdot a^5 - \frac{16577}{2^{10}} \cdot a^3 + \frac{4589}{2^{10}} \cdot a$
- $g_4(a) = \frac{46623}{2^{10}} \cdot a^9 - \frac{113492}{2^{10}} \cdot a^7 + \frac{97015}{2^{10}} \cdot a^5 - \frac{34974}{2^{10}} \cdot a^3 + \frac{5850}{2^{10}} \cdot a$

Fig. 1 compares the graphs of $f_n(a)$ and $g_n(a)$ where $a \in [-1, 1]$.

In some real-world applications, such as thresholding for images, one of the inputs to be compared by Comp is a constant. Suppose

that y of Algorithm 1 is a (normalized) constant $t \in (0, 1)$. The range of a in Algorithm 1 then changes from $[-1, 1]$ to $[-t, 1-t]$. When using $f_n(a)$, it limits the range of comparison output values from $[0, 1]$ to $[c_{min}, c_{max}]$ where c_{min} and c_{max} stand for $\mathtt{Comp}(0, t; n, d)$ and $\mathtt{Comp}(1, t; n, d)$, respectively. The values of c_{min} and c_{max} can be computed in advance. The optimal outcomes when employing Comp involve attaining values that closely approach 0 and 1 while minimizing the number of iterations. Transforming the range of comparison output values for $a \in [-t, 1-t]$ back to $[0, 1]$ results in a sharper slope for the approximate sign function, which helps reduce the number of iterations. Based on this, Shyi and Kim [26] proposed a shifting-and-scaling-based fast convergence (SSFC) method, of which equation is presented in (2).

$$\mathtt{Comp\text{-}SSFC}(x, t; n, d) = \frac{\mathtt{Comp}(x, t; n, d) - c_{min}}{c_{max} - c_{min}}. \tag{2}$$

Compared to the original Comp, Comp-SSFC requires one more multiplication with a constant $1/(c_{max} - c_{min})$, increasing the total depth by 1 in the HE domain.

3 FAST CONVERGENCE METHOD

Comp-SSFC enables fast convergence when the number of iterations is small. However, as the number increases, the gain decreases because the value of c_{max} and the value of c_{min} become closer to 1 and 0, respectively. In this case, the added computation and depth consumption become worthless. Furthermore, an error occurs when the value of x is near t (i.e., $a \approx 0$) because of the shift. This error is amplified as iterations progress.

To solve these problems, we propose a lightweight input scaling-based fast convergence method for an approximate comparison operation, which is referred to as Comp-ISFC. In contrast to Comp-SSFC, which conducts post-processing on comparison outputs, Comp-ISFC scales comparison inputs prior to executing the operation, which compresses the approximate sign graph along the a axis. The formula for the proposed method for scaling inputs of $g_n(a)$, which is denoted by $w_n(a)$, is shown in (3).

$$w_n(a) = g_n(\frac{a}{k}). \tag{3}$$

The value of k is determined based on the value of t as follows:

$$k = \begin{cases} 1 - t, & \text{if } 0 \le t < 1/2, \\ t, & \text{otherwise.} \end{cases} \tag{4}$$

For a constant t, $1/k$ is pre-applied to the coefficients of $g_n(a)$. Therefore, Comp-ISFC has almost identical computational complexity as the original Comp.

Fig. 2 compares g_2 and w_2. Specifically, Figs. 2(a) and 2(b) present comparisons for t smaller than 0.5 and t greater than 0.5, respectively. Our w_n shows a steeper slope than g_n near the origin, resulting in a faster convergence towards 1 or −1. However, due to oscillations in its function values, w_n does not always present a more accurate result than g_n. The gray arrows in Fig. 2 represent the "reverse" area. However, w_n is located near 1 (between $1-\tau$ and 1) or −1 (between −1 and $-1+\tau$) in this area. As $w_n(a)$ is an input for f_n in subsequent iterations, it satisfies $f_n(w_n(a)) \approx f_n(1) = 1$ or $f_n(w_n(a)) \approx f_n(-1) = -1$. Therefore, $f \circ \cdots \circ f \circ w$ shows higher accuracy than $f \circ \cdots \circ f \circ g$ when using the same number of iterations.

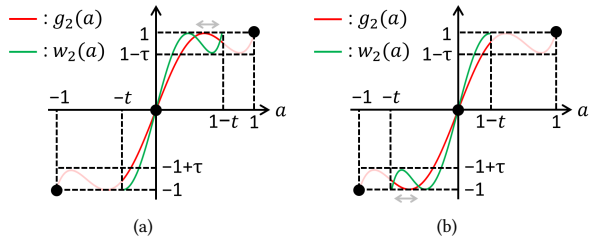

Figure 2: Comparison of the g_n and w_n functions. (a) when t is smaller than 0.5 (b) when t is greater than 0.5.

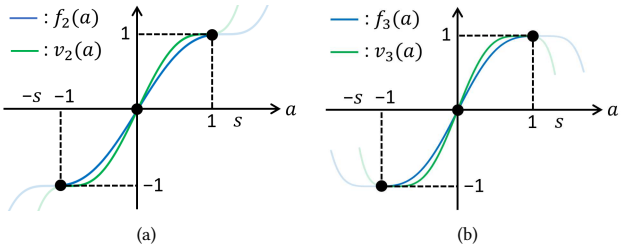

Figure 3: Comparison of the f_n and v_n functions. (a) when n is 2 (b) when n is 3.

To further enhance the convergence speed, Comp-ISFC applies input scaling to f_n as well. The blue graphs in Fig. 3 depict $f_2(a)$ and $f_3(a)$. When a exceeds the range $[-1, 1]$, $f_n(a)$ deviate from 1 or −1. Nonetheless, they are maintained in proximity to 1 or −1 to some extent, particularly when a lies within $[-s, s]$. Therefore, $f_n(a)$ is modified as described by (5), and Comp-ISFC uses the composition $v \circ \cdots \circ v \circ w$ instead of $f \circ \cdots \circ f \circ g$.

$$v_n(a) = f_n(s \cdot a). \tag{5}$$

The green graphs in Fig. 3 represent v_2 and v_3. The value of s is empirically set to 1.2 and is pre-applied to the coefficients of f_n.

4 MULTITHREADING FOR THRESHOLDING

In this paper, Comp-ISFC is applied to multilevel image thresholding in the HE domain. The same method in [24] is used: 1) multiple comparison operations with different thresholds are performed; 2) the comparison results are added together (e.g., the ideal sum is 0+0, 1+0, or 1+1 if two comparison operations are used for 3-level thresholding); 3) the sum is multiplied by a constant (e.g., 127 is used for 3-level thresholding, with each pixel intensity value represented as 0, 127, or 254). In this method, the total depth does not increase even if the number of thresholds increases because comparison operations are not performed in series.

To accelerate this application, we explore various levels of parallelism. The first level of parallelism arises in multiple comparison operations. These operations take the same input pixels but different thresholds, which enables them to be executed concurrently. The second level pertains to multiple ciphertexts generated from the same image. Specifically, when the number of pixels in an image exceeds the maximum limit accommodated by a ciphertext, the image is divided into smaller sub-images. They are then encoded

and encrypted into multiple independent ciphertexts, allowing for parallel processing. The third level of parallelism exists within the loop for the approximate sign functions.

Listing 1 presents the pseudocode of `Comp-ISFC` with OpenMP directives [10]. It exploits the second and third parallelism levels. `HomSub` and `HomMulPlain` stand for homomorphic subtraction between ciphertexts and homomorphic multiplication between a ciphertext and a plaintext constant, respectively. `TermCal` that involves `HomMul` and `HomMulPlain` computes innermost loop results, minimizing depth accumulation. Here are brief explanations about the OpenMP directives used in this pseudocode:

- `omp parallel for`: distributes iterations of a loop among available threads.
- `omp parallel for collapse(2)`: parallelizes a nested loop.
- `omp barrier`: synchronizes threads.
- `omp critical`: executes only one thread at a time.

Leveraging these multiple levels of parallelism enables the concurrent execution of a substantial number of threads. This is particularly advantageous for cloud servers with large hardware resources and a cluster of edge computing devices.

```
#pragma omp parallel for
for (c=0; c<ct_no; c++) { # ct_no: no. ciphertexts
  A[c] = HomSub(X[c], T); # X: input pixel, T: threshold
}
#pragma omp barrier
for (i=1; i≤d; i++)
#pragma omp parallel for collapse(2)
for (c=0; c<ct_no; c++) {
  for (j=0; j≤n; j++) {
    R1[j][c] = TermCal(A[c], C[j], j); # C: coefficient
    #pragma omp critical
    R2[c] = HomAdd(R2[c], R1[j][c]);
  }
  A[c] = R2[c];
}
#pragma omp barrier
#pragma omp parallel for
for (c=0; c<ct_no; c++) {
  R3[c] = HomAdd(A[c], 1);
  R4[c] = HomMulPlain(R3[c], 0.5); # result
}
#pragma omp barrier
```

Listing 1: Pseudocode of `Comp-ISFC` with OpenMP directives.

5 EVALUATION

5.1 Experimental Setup

In this section, `Comp-ISFC` is evaluated in HE-based 3-level thresholding. The overall evaluation process is illustrated in Fig. 4, where HE-based 3-level thresholding with the previous and proposed approximate comparison operations is highlighted in gray. This process is conducted on a workstation with Intel Xeon W-2295 with 18 cores and 128 GB RAM. In addition, evaluation in an edge device scenario is conducted using a Raspberry Pi 5, equipped with a 64-bit quad-core ARM Cortex-A76 processor and 8 GB RAM [13].

For the implementation using CKKS functions, we employ the Microsoft SEAL open-source library (version 3.6) [25]. The HE parameters we consider include the security level λ, polynomial degree N, and coefficient modulus bit count $\log q$. The security level is set to 128 bits, which is one of the most popular in contemporary real-world applications based on HE [7]. We set N to 2^{15} to ensure

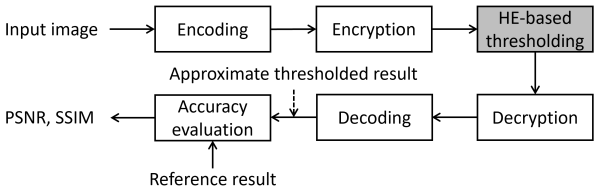

Figure 4: Overall evaluation process.

Figure 5: Input test images. (a) cameraman (b) lake (c) pirate (d) woman.

Table 1: Total Depth in HE-based 3-Level Thresholding

d	2		3		4	
n	2-3	4	2-3	4	2-3	4
Original Comp [8]	9	12	12	15	15	19
Comp-SSFC [26]	10	13	13	16	16	20
Comp-ISFC (this work)	**9**	**12**	**12**	**15**	**15**	**19**

ample depth. Our chosen security level and N result in $\log q$ being 885-bits, enabling a maximum depth of approximately 20 [4].

Considering this maximum depth, d and n of the approximate comparison operations are configured within the range of 2-4. For all previous and proposed approximate comparison operations in this section, g_n or w_n is executed during the first iteration, while f_n or v_n is executed during subsequent $d-1$ iterations.

As input images, four 512×512 gray-scale standard test images (cameraman, lake, pirate, and woman) are used, which are shown in Fig. 5 [16]. Each pixel intensity in these images is represented using 8 bits. With N of 2^{15}, each image is encoded and encrypted into 16 ($= 512 \times 512/2^{15-1}$) ciphertexts. For 3-level thresholding, two threshold values, 85 and 170, are selected. To meet the input range requirements of the approximate comparison operations, these thresholds are normalized to 0.33 and 0.67, respectively. Each pixel intensity obtained by thresholding with approximate comparison operations has a value close to 0, 127, or 254 as in [24].

To evaluate the accuracy of HE-based 3-level thresholding compared to the original non-HE-based approach using precise comparison operation, the popular peak signal-to-noise ratio (PSNR) and structural similarity index measure (SSIM) are used [18]. In particular, SSIM is used to evaluate the perceptual image quality, given that the resulting pixel intensity values by HE-based thresholding closely approximate 0, 127, and 254.

5.2 Depth

The total depths in HE-based 3-level thresholding are shown in Table 1. The depth consumed in the approximate sign functions increases proportionally with $\lceil \log_2(n+1) \rceil$ in our implementations,

Table 2: PSNR Comparison in HE-based 3-Level Thresholding

d	n	Cameraman				Lake				Pirate				Woman			
		[8]	[26]	this work		[8]	[26]	this work		[8]	[26]	this work		[8]	[26]	this work	
				D1	D2			D1	D2			D1	D2			D1	D2
2	2	16.92	16.93	**18.01**	**18.60**	17.41	17.47	**19.06**	**19.99**	17.30	17.33	**18.67**	**19.38**	18.38	18.43	**20.26**	**21.20**
2	3	18.23	18.23	**19.74**	**20.50**	19.36	19.42	**21.23**	**22.14**	18.92	18.94	**20.56**	**21.32**	20.59	20.64	**22.51**	**23.38**
2	4	19.56	19.57	**21.24**	**22.07**	21.04	21.04	**22.94**	**23.88**	20.37	20.38	**22.03**	**22.82**	22.31	22.32	**24.13**	**24.98**
3	2	18.77	18.77	**20.38**	**21.86**	20.05	20.05	**21.99**	**23.88**	19.53	19.53	**21.20**	**22.67**	21.32	21.32	**23.23**	**24.86**
3	3	21.32	21.32	**23.11**	**24.86**	23.06	23.06	**24.95**	**26.62**	22.11	22.11	**23.79**	**25.37**	24.23	24.23	**25.99**	**27.59**
3	4	23.48	23.48	**25.31**	**26.86**	25.32	25.32	**27.13**	**28.66**	24.13	24.13	**25.83**	**27.28**	26.34	26.34	**28.07**	**29.55**
4	2	21.32	21.32	**23.11**	**25.42**	23.06	23.06	**24.95**	**27.46**	22.11	22.11	**23.79**	**26.06**	24.23	24.23	**25.99**	**28.32**
4	3	24.87	24.87	**26.56**	**28.84**	26.70	26.70	**28.35**	**30.54**	25.42	25.42	**27.00**	**29.07**	27.66	27.66	**29.26**	**31.40**
4	4	27.44	27.44	**29.02**	**30.68**	29.21	29.21	**30.74**	**32.34**	27.81	27.81	**29.22**	**30.58**	30.09	30.09	**31.59**	**33.13**

Table 3: SSIM Comparison in HE-based 3-Level Thresholding

d	n	Cameraman				Lake				Pirate				Woman			
		[8]	[26]	this work		[8]	[26]	this work		[8]	[26]	this work		[8]	[26]	this work	
				D1	D2			D1	D2			D1	D2			D1	D2
2	2	0.654	0.668	**0.702**	**0.742**	0.517	0.520	**0.594**	**0.688**	0.404	0.406	**0.513**	**0.619**	0.337	0.338	**0.429**	**0.615**
2	3	0.725	0.726	**0.774**	**0.790**	0.631	0.643	**0.754**	**0.801**	0.560	0.577	**0.693**	**0.741**	0.486	0.518	**0.714**	**0.793**
2	4	0.769	0.770	**0.807**	**0.822**	0.749	0.750	**0.830**	**0.861**	0.684	0.685	**0.777**	**0.813**	0.713	0.714	**0.829**	**0.864**
3	2	0.746	0.746	**0.790**	**0.817**	0.681	0.681	**0.794**	**0.862**	0.621	0.621	**0.733**	**0.811**	0.589	0.589	**0.778**	**0.866**
3	3	0.809	0.809	**0.844**	**0.871**	0.834	0.834	**0.892**	**0.910**	0.780	0.780	**0.849**	**0.876**	0.831	0.831	**0.894**	**0.888**
3	4	0.851	0.851	**0.890**	**0.920**	0.902	0.902	**0.940**	**0.961**	0.860	0.860	**0.907**	**0.935**	0.903	0.903	**0.939**	**0.959**
4	2	0.809	0.809	**0.844**	**0.894**	0.834	0.834	**0.892**	**0.947**	0.780	0.780	**0.849**	**0.914**	0.831	0.831	**0.894**	**0.945**
4	3	0.881	0.881	**0.915**	**0.949**	0.933	0.933	**0.957**	**0.976**	0.897	0.897	**0.931**	**0.958**	0.932	0.932	**0.956**	**0.973**
4	4	0.930	0.930	**0.952**	**0.970**	0.966	0.966	**0.977**	**0.986**	0.943	0.943	**0.961**	**0.973**	0.961	0.961	**0.976**	**0.984**

resulting in variations as n changes from 3 to 4. Additionally, the total depth in a comparison operation grows as d increases. As described in Sections 2.2, `Comp-SSFC` demands an additional depth, when compared to the original `Comp`. In contrast, `Comp-ISFC` does not require this as the multiplications with $1/k$ and s are pre-applied to the coefficients of the approximate sign functions.

5.3 Accuracy

Tables 2 and 3 show the PSNR and SSIM results in HE-based 3-level thresholding. The proposed work includes two distinct designs: D1 uses $f \circ \cdots \circ f \circ w$ for approximate comparison operations, while D2 uses $v \circ \cdots \circ v \circ w$.

In general, the application of `Comp-SSFC` produces slightly better results compared to the application of the original `Comp` for $d = 2$. However, as d increases, the gain becomes negligible because c_{min} and c_{max} approach 0 and 1, respectively.

`Comp-ISFC` presents a notable enhancement in both PSNR and SSIM values across all cases. On average, applying `Comp-ISFC` (D2) yields PSNR values that are 13%, 15%, and 14% higher for d of 2, 3, and 4, respectively, compared to applying the original `Comp`. In terms of SSIM, the application of `Comp-ISFC` shows an increase of 31%, 15%, and 8% for d values of 2, 3, and 4, respectively.

Considering the same d, the results achieved by `Comp-ISFC` are in most cases better than those of the previous works using one larger n. This implies a reduction in total depth while achieving a similar or slightly better accuracy.

When consuming the same depth, $(d, n) = (3, 3)$ shows higher accuracy compared to $(d, n) = (2, 4)$. Similarly, $(d, n) = (4, 3)$ presents higher accuracy than $(d, n) = (3, 4)$. This observation proves the claim in [8] that d has a greater impact on accuracy than n.

5.4 Execution Time

In this subsection, the execution time results (in minutes) in HE-based 3-level thresholding are presented. Fig. 6 shows the results on the workstation, and the number of threads changes from 1 to 8. Due to space constraints, the results for $n = 3$ are not included.

As d increases while maintaining n, the outer loop of the approximate comparison operations undergoes more iterations. Consequently, the results show nearly linear growth. As n increases from 2 to 4 while maintaining d, all experimental cases (with different approximate comparison operations, d values, and thread counts) present similar increase rates, 3.23× on average. Increasing the thread count enhances processing speed. However, the improvement is not exactly linear. Specifically, compared to the case where a single thread is used, the average execution time increases by 1.68×, 3.09×, and 5.60× when the number of threads is 2, 4, and 8, respectively. This is due to factors such as thread synchronization, communication overhead, and shared resources.

Compared to the implementation using the original `Comp`, the implementation using `Comp-ISFC` results in a mere 2% increase in average execution time. When compared to the implementation using `Comp-SSFC`, the proposed work reduces the average execution

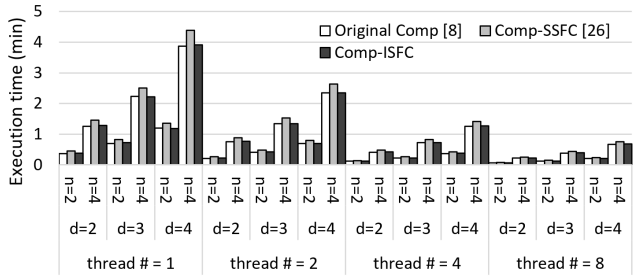

Figure 6: Execution time (minutes) on a workstation.

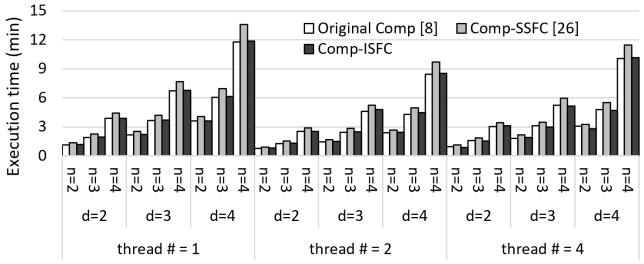

Figure 7: Execution time (minutes) on a Raspberry Pi.

time by 12%. This reduction stems from the fact that `Comp-SSFC` requires an additional multiplication.

Fig. 7 presents the execution time results of HE-based 3-level thresholding performed on the Raspberry Pi. Using a single thread shows a 3.06× slower processing speed than that achieved on the workstation. Using two threads enhances the processing speed by 47%. When scaling up to 4 threads, the processing speed becomes slower. This slowdown may be attributed to contention for limited hardware resources and synchronization overhead.

Compared to the single-thread implementation using the original `Comp` with n of 4, that using `Comp-ISFC` with n of 3, showing comparable accuracy, results in a 48% enhancement in execution time on the Raspberry Pi. When the number of threads increases to 2, this improvement increases to 63%.

6 CONCLUSION

This paper presents a novel method that exploits an asymmetric input range and scales inputs for an approximate comparison operation. It reduces the number of iterations and execution time while showing comparable accuracy. Consequently, this method expands the feasibility of privacy-preserving real-world applications executed on cyber components of CPS. Furthermore, this paper explores parallelism inherent in HE-based multilevel thresholding involving comparison operations and implements it using OpenMP. In the experiments conducted with the Raspberry Pi, it was observed that increasing the number of threads beyond 2 resulted in a contrary effect. Therefore, as part of future work, we plan to explore the utilization of a Raspberry Pi cluster for the application.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 2347253. We thank Dr. Rob Rutenbar, Dr. Jung Hee Cheon, and Dr. Keewoo Lee.

REFERENCES

[1] 2023. Lattigo v5. Online: https://github.com/tuneinsight/lattigo. EPFL-LDS, Tune Insight SA.
[2] Ahmad Al Badawi et al. 2022. Openfhe: Open-source fully homomorphic encryption library. In *Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. 53–63.
[3] David Archer et al. 2017. Applications of homomorphic encryption. In *Crypto Standardization Workshop, Microsoft Research*, Vol. 14. sn.
[4] Hao Chen et al. 2017. Simple encrypted arithmetic library v2. 3.0. *Microsoft Research, December* 13 (2017).
[5] Jung Hee Cheon et al. 2017. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*. Springer, 409–437.
[6] Jung Hee Cheon et al. 2019. Numerical method for comparison on homomorphically encrypted numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 415–445.
[7] Jung Hee Cheon et al. 2021. Introduction to homomorphic encryption and schemes. *Protecting Privacy through Homomorphic Encryption* (2021), 3–28.
[8] Jung Hee Cheon, Dongwoo Kim, and Duhyeong Kim. 2020. Efficient homomorphic comparison methods with optimal complexity. In *Advances in Cryptology–ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part II 26*. Springer, 221–256.
[9] Michael Cho, Keewoo Lee, and Sunwoong Kim. 2022. HELPSE: Homomorphic encryption-based lightweight password strength estimation in a virtual keyboard system. In *Proceedings of the Great Lakes Symposium on VLSI 2022*. 405–410.
[10] Leonardo Dagum and Ramesh Menon. 1998. OpenMP: an industry standard API for shared-memory programming. *IEEE computational science and engineering* 5, 1 (1998), 46–55.
[11] Daniel L Elworth and Sunwoong Kim. 2022. HEKWS: Privacy-Preserving convolutional neural network-based keyword spotting with a ciphertext packing technique. In *2022 IEEE 24th International Workshop on Multimedia Signal Processing (MMSP)*. IEEE, 01–06.
[12] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive* (2012).
[13] Raspberry Pi Foundation. 2023. *Raspberry Pi 5*. https://www.raspberrypi.com/documentation/computers/raspberry-pi-5.html
[14] Craig Gentry. 2009. *A fully homomorphic encryption scheme*. Stanford university.
[15] Ran Gilad-Bachrach et al. 2016. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International conference on machine learning*. PMLR, 201–210.
[16] Rafael C. Gonzalez, Richard E. Woods, and Steven L. Eddins. [n. d.]. *Image Databases*. https://www.imageprocessingplace.com/root_files_V3/image_databases.htm
[17] Seungwan Hong et al. 2021. Efficient sorting of homomorphic encrypted data with k-way sorting network. *IEEE Transactions on Information Forensics and Security* 16 (2021), 4389–4404.
[18] Alain Hore and Djemel Ziou. 2010. Image quality metrics: PSNR vs. SSIM. In *2010 20th international conference on pattern recognition*. IEEE, 2366–2369.
[19] Sharmila Devi Kannivelu and Sunwoong Kim. 2021. A Homomorphic Encryption-based Adaptive Image Filter Using Division Over Encrypted Data. In *2021 IEEE 27th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 67–72.
[20] Andrey Kim et al. 2018. Logistic regression model training based on the approximate homomorphic encryption. *BMC medical genomics* 11, 4 (2018), 23–31.
[21] Junsoo Kim et al. 2016. Encrypting controller using fully homomorphic encryption for security of cyber-physical systems. *IFAC-PapersOnLine* 49, 22 (2016), 175–180.
[22] Miran Kim et al. 2021. Ultrafast homomorphic encryption models enable secure outsourcing of genotype imputation. *Cell Systems* 12, 11 (2021), 1108–1120.e4. https://doi.org/10.1016/j.cels.2021.07.010
[23] Eunsang Lee et al. 2021. Minimax approximation of sign function by composite polynomial for homomorphic comparison. *IEEE Transactions on Dependable and Secure Computing* 19, 6 (2021), 3711–3727.
[24] Paul Nam, Justin Shyi, and Sunwoong Kim. 2022. HEMTH: Small Depth Multilevel Thresholding for a Homomorphically Encrypted Image. In *2022 IEEE 24th International Workshop on Multimedia Signal Processing (MMSP)*. IEEE, 1–6.
[25] SEAL 2020. Microsoft SEAL (release 3.6). https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA..
[26] Justin Shyi and Sunwoong Kim. 2023. HEBGS: Homomorphic Encryption-based Background Subtraction Using a Fast-Converging Numerical Method. In *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–5.
[27] Zama. 2022. TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data. https://github.com/zama-ai/tfhe-rs.