Soft Computing (2020) 24:12413–12427
https://doi.org/10.1007/s00500-020-04683-4

METHODOLOGIES AND APPLICATION

A novel parallel image encryption algorithm based on hybrid chaotic maps with OpenCL implementation

Lin You[1] · Ersong Yang[2] · Guangyi Wang[3]

Published online: 14 January 2020

Abstract
Since chaotic maps have the excellent properties of unpredictability, ergodicity and sensitivity to their parameters and initial values, they are quite suitable for generating chaotic sequences for securing communication systems and are also especially useful for securing images, and a lot of chaotic map-based image encryption algorithms have been proposed. But some existing image encryption algorithms were proved that their security, encryption efficiency or computational speeds are not quite satisfactory for practical applications. Some of them using only one type of chaotic system may suffer from low key space, and some others using two or more types of chaotic system may suffer from high computational overheads. In this paper, based on the classic 1D logistic map, a 2D one-coupling logistic dynamics system and OpenCL, a novel parallel image encryption algorithm HCMO is proposed. Our algorithm consists of a confusion phase and a diffusion phase using four sub-key matrices based on the hybrid logistic dynamics systems, the linear transformation and the enlarging operation. In the confusion phase, the image's pixel positions are first scrambled by performing row-wise and column-wise permutation operations using two sub-key matrices; then, in its diffusion phase, both the bit XOR operation and the bit cyclic shifting are applied onto the scrambled intermediate image matrix using the other two sub-key matrices. In order to reduce the whole encrypting execution time, we speed up our HCMO on an OpenCL's heterogeneous and parallel characteristics. Compared to the implementation of Vihari's algorithm and some other chaotic map-based algorithms referred in this paper with the OpenCL-based implementation on the CPU and on the GPU, respectively, our algorithm's simulation demonstrates remarkable improvement in the operational speedup, and the experimental result analyses have also shown that HCMO has a higher-level security than some other referred algorithms.

Keywords Hybrid chaotic maps · Image encryption · Parallel computing · OpenCL · GPU

1 Introduction

In this digital era, more and more digital images have been transmitting over networks, and most of them have to be transmitted over public networks. In order to transmit secret digital images to the receivers, the digital image encryption technology should be employed. In the past decades, many image encryption algorithms have been proposed, such as the algorithms based on Arnold transform (Chen and Ping 2006) or on Josephus traversing (Xiang and Xiong 2005). But most of them were based on the chaos maps (Lian et al. 2005; Wang et al. 2009; Wong et al. 2008; Patidar et al. 2009; Ye 2010; Zhu et al. 2011; Zhang et al. 2013; Wu et al. 2012; Hu and Han 2009; Pareek et al. 2013; Zhao et al. 2015; Askar et al. 2015; Vihari and Manoj 2012; Rodrguez-Vzquez et al 2012; Pareek et al. 2006; Fridrich 1998; Eklund et al. 2013; Fu et al. 2013; Bhogal et al. 2018; Gupta et al. 2018; Farajallah et al. 2016; Hanis and Amutha 2019; Mondal et al. 2018). However, almost all of these algorithms are implemented on the CPU, which encryption speed can hardly meet the requirement of the real-time communication. So how to carry out the image encryption as fast as possible has also become a hot issue.

Communicated by V. Loia.

✉ Lin You
mryoulin@gmail.com

1 School of Cybersecurity, Hangzhou Dianzi University, Hangzhou 310018, China

2 School of Communication Engineering, Hangzhou Dianzi University, Hangzhou 310018, China

3 School of Electronic Information, Hangzhou Dianzi University, Hangzhou 310018, China

It is well known that the chaotic system has some good characteristics, such as ergodicity and pseudorandomness. Since these features meet some requirements such as diffusing and mixing in the sense of cryptography, it has played an important role in image encryption. In 1998, Fridrich first proposed the classical image encryption structure which consisted of some permutations and diffusions (Fridrich 1998), called Fridrich structure for convenience. This structure has been demonstrated to be very effective in providing both confusion and diffusion properties in image encryption, and it has become an important work in the information security.

In this paper, based on the framework of Fridrich structure, by using hybrid chaotic maps and OpenCL (Munshi et al. 2011; Qiu 2011), a novel image encryption algorithm is proposed and it is called HCMO for shortly. Here, the hybrid chaotic maps combine a one-dimensional logistic map and a two-dimensional logistic map.

The rest of the paper is organized as follows. In Sect. 2, the backgrounds of the chaotic maps and the OpenCL, and also some previous work are described in Sect. 2. In Sect. 3, a novel image encryption algorithm is proposed by the combined use of a 1D logistic map and a 2D logistic chaotic map. The OpenCL-based implementation of our algorithm is discussed in Sect. 4. The simulation results and the performance analysis are reported in Sect. 5. Finally, some concluding remarks are drawn in Sect. 6.

2 Background and some previous work

2.1 One-dimensional logistic map

The one-dimensional logistic map is a polynomial recurrencial mapping of degree 2, and its mathematical expression can be described as

$$x_{n+1} = f(\eta, x_n) \tag{1}$$

where x_0 is the initial value and $0 < x_n < 1\,(n = 0, 1, 2, \ldots)$ and $f(\eta, x)$ is a polynomial function of degree 2 with η a variable control parameter. The classical logistical map was first created by the mathematician P. F. Verhulst in 1845. It is brought up using the two-degree polynomial $f(\eta, x) = 4\eta x(1 - x)$. Hence, the classic logistic map is expressed as

$$x_{n+1} = 4\eta x_n(1 - x_n) \tag{2}$$

where x_0 is the initial value, $0 < x_n < 1$ $(n = 0, 1, 2, \ldots)$. If η satisfies $0.89249 \le \eta \le 1$, then the system will enter a state of chaos, which means that the sequence iteratively generated by Eq. (1) will keep in a state of pseudorandom distribution. When $\eta = 1$, the system will keep the hyperchaotic state. However, only using the one-dimensional logistic map in an image encryption algorithm cannot ensure its security since the probability density function of this logistic map is non-uniform and it may make their chaotic orbits be estimated.

2.2 Two-dimensional logistic map

A two-dimensional logistic map or a 2D logistic map is an extension of some one-dimensional logistic map or 1D logistic map. A 2D logistic map not only has more parameters than one-dimensional logistic map, but also has some important advantages over the one-dimensional logistic map, such as higher information entropy scores (which means that its trajectory is more random-like), more complex and more dynamic (that is, larger Lyapunov exponent). For more details about the chaotic map complexity comparison, one can refer to Table 1 given in Wu et al.'s work (2012). All these characteristics will make that a 2D logistic map shows a better performance for image encryption than a 1D logistic map. Hence, a lot of work have been done to use 2D logistic maps or even high-dimensional logistic maps for image encryption. Henon map, a typical 2D chaotic map, was proposed by Michel Henon in 1976. It is a prototypical two-dimensional invertible iterated map represented by the state equations with a chaotic attractor. 2D Arnold cat map is another most commonly used 2D chaotic map for image encryption, which was proposed by Vladimir Igorevich Arnold in 1967.

A mostly used equation of 2D logistic dynamics system can be expressed as the following equation set:

$$\begin{cases} x_{n+1} = 4\eta\lambda x_n(1 - x_n) + g_1(x_n, y_n) \\ y_{n+1} = 4\eta\delta y_n(1 - y_n) + g_2(x_n, y_n) \end{cases} \tag{3}$$

where $0 < x_n, y_n < 1$ for $i = 0, 1, 2, \ldots$, x_0 and y_0 are the system initial values, while η, λ and δ are the control parameters that control the system dynamic behavior. In addition, each $g_i(-)\ (i = 1, 2)$ are one- or two-variable polynomials which are called coupling terms. If $g_1(x_n, y_n) = g_2(x_n, y_n) = \gamma x_n y_n$, then $g_i(-)$ are called symmetric two-coupling terms. If $g_1(x_n, y_n) = \gamma y_n$ and $g_2(x_n, y_n) = \tau x_n$, then $g_1(-)$ and $g_2(-)$ are called one-coupling terms with two control parameters γ and τ, respectively. In this work, we assure that $g_1(-)$ and $g_2(-)$ are one-coupling terms with different control parameters, that is, we will employ the following 2D logistic chaotic dynamics system as the following equation set expressed:

$$\begin{cases} x_{n+1} = 4\eta\lambda x_n(1 - x_n) + \gamma y_n \\ y_{n+1} = 4\eta\delta y_n(1 - y_n) + \tau x_n \end{cases} \tag{4}$$

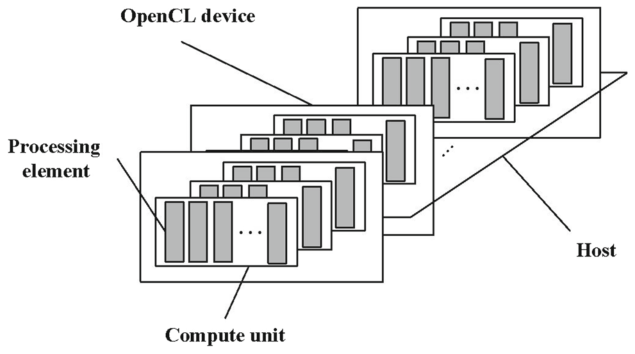

Fig. 1 OpenCL platform model

where $0.2 \leq \lambda,\ \delta \leq 0.9$, $0.1 \leq \gamma,\ \tau \leq 0.9$ and $0.89249 \leq \eta \leq 1$.

2.3 OpenCL overview

The OpenCL (Munshi et al. 2011) is not only a kind of programming language but also a heterogeneous parallel computing framework consisted of programming language standards, application programming interfaces, function libraries and other processors. The OpenCL allows executing calculation programs on many-core processors. In order to deliver high-level portability, the OpenCL not only supports different kinds of CPUs, GPUs and other hardware, but also provides an abstract underlying hardware model. The OpenCL 1.1 specification (Qiu 2011) is made up of four main parts: platform model, execution model, memory model and programming model.

2.3.1 Platform model

This model is shown in Fig. 1, and it is composed of a single host and one or more OpenCL devices. The latter is responsible for implementing the OpenCL kernel program. The host is connected to one or more OpenCL devices where the instruction streams of the kernels execute.

2.3.2 Execution model

This execution model is an abstract representation of how the instruction stream executes on the heterogeneous platform. It defines how an OpenCL application is mapped onto the processing elements, the memory regions and the host. An OpenCL program consists of two distinct parts: a host program and a collection of one or more kernels. The host program runs on the host. The OpenCL execution model defines how the kernels execute on the OpenCL devices.

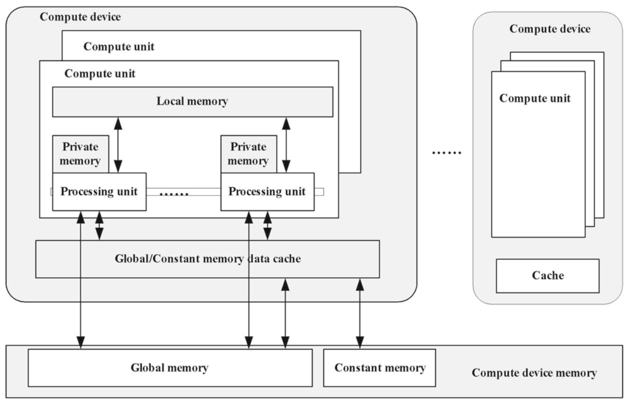

Fig. 2 OpenCL memory model

2.3.3 Memory model

An OpenCL memory model defines the collection of some memory regions within the OpenCL and how the kernels execute and how they interact with the host, etc. The model defines four distinct memory regions: the constant memory, the local memory, the global memory and the private memory. Figure 2 shows the memory regions and how they relate to the platform and work with the execution models.

2.3.4 Programming model

The programming model is a high-level abstract model that a programmer uses to design some algorithms to implement an application. The OpenCL programming model is defined by two different models, that is, the task parallel model and the data parallel model. The data parallel model is the primary model driving the design of OpenCL. In the data parallel model, an index space is associated with the OpenCL execution model and it defines the work items and how the data are mapped onto the work items. All the work items in one compute unit execute the same instructions. A N-dimensional grid of the index space is called a NDRange, whose length is defined by the integer N which denotes the size of the dimensional space (where $N = 1, 2$ or 3).

For the task parallel programming model, different kernels are passed through the command queue to be executed on different compute units or processing elements. An index space is also defined for task parallel processing, in which both the number of work groups and the work items are 1, since the parallel command stream needs to contain a single work item to handle and schedule all the tasks.

2.4 GPU

The term graphics processing unit (GPU) was popularized by Nvidia in 1999, who marketed the GeForce 256 as "the world's first GPU." It is a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display. GPUs are used in embedded systems, mobile phones, personal computers, workstations and game consoles. Modern GPUs are very powerful and efficient at manipulating computer graphics and image processing, and their highly parallel structure makes them much more effective than general-purpose CPUs for algorithms where the processing of large blocks of visual data is done in parallel.

Architecturally, the CPU is composed of just few cores with lots of cache memory that can handle a few software threads at a time. In contrast, a GPU is composed of hundreds of cores that can handle thousands of threads simultaneously. The ability of a GPU with 100 more cores to process thousands of threads can accelerate some software by 100 more times over a CPU alone. What's more, the GPU can achieve this acceleration while being more power and cost efficient than a CPU. In this paper, we will use the Nvidia GPU called GeForce GTX 580 for our proposed algorithm' s simulation. The GTX 580 boasts 16 streaming multiprocessors (SMs) and 512 CUDA cores with the frequency 1.544 GHz. In Vihari and Manoj (2012), Nvidia Tesla C2050 GPU was used for speedup a chaotic image encryption algorithm. This kind of GPU had 14 SMs and 448 CUDA cores with the frequency 1.15 GHz. Obviously, GeForce GTX 580 is more powerful and more efficient than a Nvidia Tesla C2050 GPU.

2.5 Some previous work

Encryption algorithms for digital image data can be classed as two kinds of methods, that is, the non-chaos-based method and the chaos-based method. The traditional text encryption algorithms, such as DES and AES, including the public key encryption algorithms, are non-chaos-based methods. In general, the size of an image is always much larger than that of a text; therefore, it will take much longer time for a traditional encryption algorithm to encrypt/decrypt the image data than to encrypt/decrypt the text data. Moreover, for a traditional encryption algorithm, the size of the decrypted text is required to be exactly equal to that of the original text, but this requirement is not necessary for an image data because of the characteristics of human perception, and some small distortions in the decrypted image are usually acceptable. Compared to the traditional encryption algorithm, the chaos-based encryption algorithms provide some advantages, such as the high security level, high speed especially in stream ciphers, high flexibility and high degree of modularity, low computational overheads and power and easier to be implemented. These features make them more suitable for large-scale data encryption, such as images and videos.

Nowadays, a lot of chaos-based encryption algorithms have been proposed, in which the confusion processes were completed based on some kinds of chaotic standard maps. Most of them are based on the confusion–diffusion architecture (Fridrich 1998), which was also used in the conventional cryptographic algorithms (DES, 3DES, AES, etc.).

In Lian et al. (2005), based on the chaotic standard map, Lian et al. proposed a block cipher which could be used for encrypting multimedia data. Their block cipher consisted of three parts: a confusion process, a diffusion function and a key generator, where confusion process was based on the chaotic standard map. In Wang et al. (2009), a block encryption scheme based on dynamic substitution S-boxes was proposed. In this scheme, the dynamic S-boxes was generated based on a chaotic tent map, and the plain texts were divided into the blocks and each of which was encrypted with a different S-box. In addition, a cipher feedback was used to change the state value of the chaotic tent map, which made the S-boxes relate to the plain text and enhanced the confusion and diffusion properties of the encryption.

An effective chaos-based image encryption, composed of multiple rounds of substitutions and diffusions, was proposed in Wong et al. (2008). As the confusion and diffusion effects were solely contributed by the substitution and the diffusion stages, respectively, the required overall rounds of operations in achieving a certain level of security are found more than necessary. In this letter, the authors suggested to introduce a certain diffusion effect in the substitution stage by simple sequential add-and-shift operations. Patidar et al. (2009) proposed a loss-less symmetric image cipher based on the widely used substitution–diffusion architecture which utilized some chaotic standard and logistic maps. This encryption scheme was specifically designed for color images, and its secret key was composed of the initial conditions and system parameters of the chaotic standard map and the iteration number. This encryption scheme had four rounds. In the fourth round, a robust substitution/confusion was accomplished by generating an intermediate chaotic key stream image with the help of chaotic standard and logistic maps.

In Ye (2010), a scrambling encryption algorithm for images was proposed based on a chaotic map. This algorithm used a single chaotic map only once to implement the gray image scrambling encryption, and it drastically transformed the statistical characteristic of original image information and so increased the difficulty for an unauthorized individual to break the encryption. Zhu et al. (2011) proposed a chaos-based image encryption by employing the Arnold cat map for bit-level permutation and the logistic map for diffusion. Since the bit-level permutation has the effects of both confusion and diffusion, it made their chaos-based image encryption have the much higher security level and more computationally efficient than some other chaos-based image encryptions using pixel-level permutation.

Zhang et al. (2013) proposed an image encryption scheme (Algorithm 1 in Zhang et al. 2013) in which a 1D logistic map was first used with two keys to, respectively, generate two sequences which were then used to perform confusion–diffusion operations based on the reverse 2D cat map, and it would make that the confusion effect on the encrypted image cannot be removed by a homogeneous plain image and lead that this image encryption cannot be compromised by conventional known/chosen-plaintext attacks.

In 1998, Fridrich first described how to adapt invertible two-dimensional chaotic maps to create new symmetric block encryption schemes by producing permutations and diffusions on the pixel data of the images. In addition, he explained the construction of such cipher scheme with the two-dimensional Baker map. Wu et al. (2012) proposed an image encryption method based on a two-dimensional logistic map which is different from the Baker map or the Cat map.

In Hu and Han (2009), a medical image encryption system was proposed based on a novel pixel-based scrambling scheme. This system used a simple pixel-level XOR operation for image scrambling in an innovative way with the structural parameters serving as a part of the cryptographic key which was a number sequence generated from the multi-scroll chaotic attractors. Using key-dependent diffusion and substitution techniques, Pareek et al. also proposed an encryption algorithm for gray images (Pareek et al. 2013). These gray image encryption algorithms can be employed for the protection of medical images, such as magnetic resonance images. In Zhao et al. (2015), using a new improper fractional-order chaotic system, the authors proposed a symmetric image encryption algorithm in which the initial conditions, parameters and fractional orders of chaos were influenced by gray value of all pixels and used as the secret key. While in Askar et al. (2015), a new image encryption algorithm was proposed using a kind of chaotic map called chaotic economic map. These chaotic map-based image encryption algorithms almost showed that the encrypted images had both good information entropy and high security, but most of their computational speeds showed less satisfactory.

Though chaos-based image encryption algorithms perform better compared to conventional encryption algorithms, but most of these algorithms still require considerable amount of time which would increase the overall computational overheads. Compared with the traditional central processing units (CPUs), GPUs not only can dramatically accelerate parallel computing, but also have low energy consumption and low cost. Since 2011, Vihari et al. and Rodrguez-Vzquez et al. began to apply GPU-equipped computers to speedup the simulation implementation of chaos-based image encryption algorithms, respectively. Vihari and Manoj (2012) showed that a CUDA-based implementation works for chaotic image encryption algorithm using logistic map with NVIDIA's Tesla C2050 GPU device had significant amount of improvement in terms of operational speedup compared to original implementation on CPUs, and they also affirmed that there are several other chaotic image encryption algorithms which are more complex and computationally intensive in nature that can be accelerated using CUDA version.

Rodrguez-Vzquez et al (2012) assessed the image encryption algorithm (given by Pareek et al. 2006) based on both an external secret key of 80-bit and the chaotic logistic map $x_{n+1} = 3.9999x_n(1 - x_n)$ using GPU, and they showed that the efficiency of Pareek's image encryption algorithm performed on the GPU was some worse than that performed on the OpenMP[1] variant. It implied that Pareek's algorithm was not suitable for GPU implementation, and it should be made some profound changes if some GPU device is to be used to speed up Pareek's algorithm. In 2014, Lee et al. (2014) experimentally demonstrated that the double random phase encoding algorithm (DRPE) (given in Refregier and Javidi 1995) and the traditional AES algorithm executed on a GPU-based stream-processing model. The authors experimentally demonstrated that for the encryption of an image with a pixel size of 1000×1000, the DRPE and AES techniques executed on a GPU with a parallel computing scheme can dramatically reduce computing time compared to using CPU sequential processing.

Nowadays, more and more digital medical images have been needed to be securely stored or transmitted over public networks or IoTs, and so they should be encrypted before being stored or transmitted. Because some small distortions in the decrypted images are usually acceptable, it makes that chaotic map-based encryption be much more suitable for securing a large mount of medical images and show superior to the traditional symmetric encryptions. In addition, as the work in Lee et al. (2014) had shown, encryption implementation on a GPU can reduce much more computation time than on a CPU. A lot of chaotic map-based medical image encryption methods have been proposed (Eklund et al. 2013; Fu et al. 2013; Bhogal et al. 2018; Gupta et al. 2018), and all these chaotic map-based medical image encryptions had been implemented using GPU device or can be optimized and improved using GPU-based parallel processing model.

In this paper, we will propose a novel image encryption algorithm combining a 1D logistic map and a 2D logistic chaotic map based on the OpenCL. In our algorithm, the 2D logistic chaotic map was first used with two keys to, respectively, generate four initial key streams which were applied to, respectively, produce four metrics based on the 1D logistic map for confusion operations and diffusion operations.

[1] OpenMP is a multi-threaded program design for shared memory parallel system. OpenMP provides a high-level abstract description of parallel algorithms, especially for parallel programming on multi-core CPU machines.

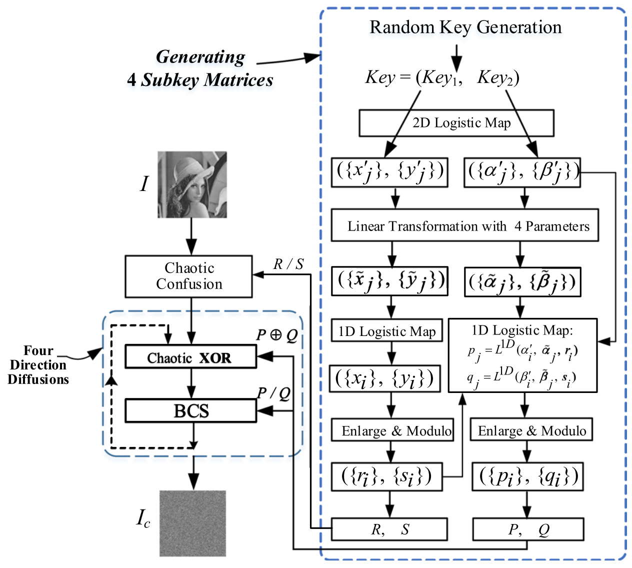

Fig. 3 Encryption process of our algorithm

3 A novel image encryption algorithm HCMO

In this section, based on hybrid chaotic maps and OpenCL, we will propose a novel parallel image encryption algorithm, which is called HCMO for short. Our algorithm mainly includes three kinds of operations: a pixel position shuffling (called PPS for short), a bit cyclic shift (called BCS for short) and a bit XOR operation. The first operation is used to confuse pixel locations; then, BCS and bit XOR operations are used for the diffusion of pixel values. The encryption process of our proposed algorithm is shown in Fig. 3.

The three steps of our algorithm applied on a given image are described in Sects. 3.1–3.3, and the parameters appeared in Fig. 3 are also illustrated in these subsections.

3.1 Generation of the chaotic sub-key matrices

3.1.1 Initializing the image and secret key

We store the original gray image in a matrix I having m rows and n columns. The secret key used for encrypting, denoted as key, consisting of two parts, that is, $key = (key_1, key_2)$ with $key_1 = (x_0, y_0, \lambda_1, \delta_1, \gamma_1, \tau_1)$ and $key_2 = (\alpha_0, \beta_0, \lambda_2, \delta_2, \gamma_2, \tau_2)$, are randomly selected but satisfy the conditions: x_0, y_0, α_0 and β_0 selected from the open interval $(0, 1)$, both λ_i and δ_i from the interval $[0.2, 0.9]$ and both γ_i and τ_i from the interval $[0.1, 0.9]$. Here, x_0, y_0, α_0 and β_0 represent the initial values of Eq. (4), while λ_i, δ_i, γ_i and τ_i represent the control parameters of Eq. (4) with $i = 1, 2$.

3.1.2 Generating the four chaotic sub-key matrices

Here, we will describe how to generate four sub-key matrices with dimension $m \times n$ denoted as P, Q, R and S, respectively. The P and Q will be used in the BCS and bit XOR phases, while R and S will be used to scramble the pixels row-by-row and column-by-column. The four matrices can be generated as follows.

Step 1 Use the secret key key_1 to generate the two initial key streams $\{x'_j\}$ and $\{y'_j\}$ $(j = 0, 1, 2, \ldots, n-1)$ based on Eq. (4). The key_1 includes the initial conditions and the control parameters of Eq. (4) with a given η. In order to avoid the transient effects, the chaotic map should be iterated $t+n$ times ($t \geq 3000$) and the results got in the previous t times are thrown away. As a result, we can obtain the two key streams $\{x'_j\}$ and $\{y'_j\}$.

Step 2 Execute the linear transformation of $\{x'_j\}$ and $\{y'_j\}$ to get the two chaotic sequences $\{\tilde{x}_j\}$ and $\{\tilde{y}_j\}$ by using the following linear equation set Eq. (5), respectively.

$$\begin{cases} \tilde{x}_j = x'_j \times a + b \\ \tilde{y}_j = y'_j \times c + d \end{cases} \tag{5}$$

with $0 < a, b, c, d < 1$, $a + b = 1$ and $c + d = 1$. In order to make Eq. (1) become a state of chaos when we use $\tilde{x}_j$ or $\tilde{y}_j$ as its control parameters, we choose a, b, c and d such that Eq. (1) for $\eta = \tilde{x}_j$ or for $\eta = \tilde{y}_j$ will become chaotic.

Step 3 Take $\{\tilde{x}_j\}$ and $\{x'_j\}$ as the control parameters and the initial values of Eq. (1), respectively, and iteratively apply Eq. (1) m times for $t = 1, 2, \ldots, n-1$ and then get the following sequence $\{x_i\}_{i=0}^{mn-1}$:

$$\begin{cases} x_0 = f(\tilde{x}_0, x'_0), \; \ldots, \; x_{tm} = f(\tilde{x}_t, x'_t); \\ x_1 = f(\tilde{x}_0, x_0), \; \ldots, \; x_{tm+1} = f(\tilde{x}_t, x_{tm}); \\ \quad \cdots \qquad \cdots \qquad \cdots \\ x_{m-1} = f(\tilde{x}_0, x_{m-2}), \; \ldots, \\ \qquad\qquad x_{tm+(m-1)} = f(\tilde{x}_t, x_{tm+(m-2)}). \end{cases} \tag{6}$$

If we choose the classic logistic map Eq. (2), then, we have the sequence as following:

$$\begin{cases} x_0 = 4\tilde{x}_0 x'_0(1 - x'_0), \; \ldots, \; x_{tm} = 4\tilde{x}_t x'_0(1 - x'_0); \\ x_1 = 4\tilde{x}_0 x_0(1 - x_0), \; \ldots, \; x_{tm+1} = 4\tilde{x}_t x_{tm}(1 - x_{tm}); \\ \quad \cdots \qquad \cdots \qquad \cdots \\ x_{m-1} = 4\tilde{x}_0 x_{m-2}(1 - x_{m-2}), \; \ldots, \\ \qquad x_{tm+(m-1)} = 4\tilde{x}_t x_{tm+(m-2)}(1 - x_{tm+(m-2)}). \end{cases} \tag{7}$$

Similarly, we can obtain another sequence $\{y_i\}_{i=0}^{mn-1}$ by using $\{\tilde{y}_j\}$ and $\{y'_j\}$ as the control parameters and the initial values of Eqs. (1) or (2), respectively.

Step 4 Set $\{r_i\} = \{x_i\} \times 10^{14} \bmod m$ and $\{s_i\} = \{y_i\} \times 10^{14} \bmod n, i = 0, 1, 2, \ldots, mn-1$. Express $\{r_i\}$ and $\{s_i\}$ in matrix by row and column as the following matrices R and S, respectively.

$$R = \begin{pmatrix} r_0 & r_1 & \cdots & r_{n-1} \\ r_n & r_{n+1} & \cdots & r_{2n-1} \\ \vdots & \vdots & \vdots & \vdots \\ r_{(m-1)n} & r_{(m-1)n+1} & \cdots & r_{mn-1} \end{pmatrix}$$

and

$$S = \begin{pmatrix} s_0 & s_m & \cdots & s_{(n-1)m} \\ s_1 & s_{m+1} & \cdots & s_{(n-1)m+1} \\ \vdots & \vdots & \vdots & \vdots \\ s_{m-1} & s_{2m-1} & \cdots & s_{nm-1} \end{pmatrix}.$$

Step 5 Similar to Step 1, we use the key key_2 to generate the two initial key streams $\{\alpha'_j\}$ and $\{\beta'_j\}$ $(j = 0, 1, 2, \ldots, n-1)$.

Step 6 Execute the linear transformations of $\{\alpha'_j\}$ and $\{\beta'_j\}$ using Eq. (5) and obtain two new sequences $\{\tilde{\alpha}_j\}$ and $\{\tilde{\beta}_j\}$. Take $\{\alpha'_j\}$, $\{\tilde{\alpha}_j\}$ and $\{r_i\}$ as the initial values, the control parameters and the iteration times, respectively, and apply Algorithm 1, then, we can generate a new chaotic sequence $\{p_i\}$ denoted as the following matrix P, which has a length of $m \times N$.

$$P = \begin{pmatrix} p_0 & p_1 & \cdots & p_{n-1} \\ p_n & p_{n+1} & \cdots & p_{2n-1} \\ \vdots & \vdots & \vdots & \vdots \\ p_{(m-1)n} & p_{(m-1)n+1} & \cdots & p_{mn-1} \end{pmatrix}.$$

Similarly, we take $\{\beta'_j\}$, $\{\tilde{\beta}_j\}$ and $\{s_i\}$ as the initial values, the control parameters and the iteration times, respectively, and then generate another chaotic sequence $\{q_i\}$ as the following matrix Q.

$$Q = \begin{pmatrix} q_0 & q_m & \cdots & q_{(n-1)m} \\ q_1 & q_{m+1} & \cdots & q_{(n-1)m+1} \\ \vdots & \vdots & \vdots & \vdots \\ q_{M-1} & q_{2m-1} & \cdots & q_{mn-1} \end{pmatrix}.$$

Algorithm 1. Generation of the Sequences $\{p_i\}_{i=0}^{nm-1}$ and $\{q_i\}_{i=0}^{nm-1}$.
Input: $\{\alpha'_j, \beta'_j\}$, $\{\tilde{\alpha}_j, \tilde{\beta}_j\}$ and $\{r_i, s_i\}$, $i = 0, 1, \ldots, mn-1$, $j = 0, 1, \ldots, n-1$;
Output: $\{p_i\}_{i=0}^{nm-1}$ and $\{q_i\}_{i=0}^{nm-1}$.

```
1. $i \leftarrow 0$;
2. for $j \leftarrow 0$ to $n-1$;
3.   $p_j \leftarrow L^{1D}(\alpha'_i, \tilde{\alpha}_j, r_i) \times 10^{14} \bmod 256$;
       // $L^{1D}$ denotes one-dimensional logistic map such as Eq. (2);
4.   $q_j \leftarrow L^{1D}(\beta'_i, \tilde{\beta}_j, s_i) \times 10^{14} \bmod 256$;
5.   for $l \leftarrow 0$ to $m-1$;
6.     $i \leftarrow i+1$;
7.     $p_{jm+l} \leftarrow L^{1D}(p_{jm+(l-1)}, \tilde{\alpha}_j, r_i) \times 10^{14} \bmod 256$;
8.     $q_{jm+l} \leftarrow L^{1D}(q_{jm+(l-1)}, \tilde{\beta}_j, s_i) \times 10^{14} \bmod 256$;
9.   end.
10. end.
```

3.2 Image confusion by chaotic sub-key matrices

Let $I(i, j)$ denote an element of the image matrix I located at ith row and jth column in the matrix. Then, we scramble I by using the following pixel position shuffling (PPS) rule Eq. (8).

$$\begin{cases} I(i, j) \longrightarrow I(R(i, j), j); \\ I(i, j) \longrightarrow I(i, S(i, j)). \end{cases} \tag{8}$$

where $i = 0, 1, \ldots, m-1$, $j = 0, 1, \ldots, n-1$. The sign "$\longrightarrow$" means each element at the position (i, j) in I is replaced by the element at the position $(R(i, j), j)$ or $(i, S(i, j))$ in I. R and S are, respectively, the row-wise scrambling and column-wise scrambling matrices given in Sect. 3.1.

Because each element in R is non-correlative to any one in S, the matrix I will be sufficiently scrambled by using R and S. From Eq. (8), we know that the scrambling rule is reversible. Therefore, if we perform the scrambling process inversely on the scrambled image, then we will recover the plain image.

3.3 Image diffusion by four direction matrix transformation

The diffusion operation includes two kinds of operations: BCS and bit XOR. In order to make the diffusion, we need to execute the following operations (9)–(12) in turn on the scrambled matrix $\bar{I}$ obtained by using the PPS rule (8). The three different symbols "$<<<$", "$>>>$" and "$\oplus$" denote the left or right BCS of a binary string, and the bit XOR, respectively, where P and Q are given as in Sect. 3.1.

$$\bar{I}(i, j) \leftarrow \bar{I}(i, j) \oplus P(i-1, j) \oplus Q(i-1, j) \oplus \bar{I}(i-1, j) <<< (P(i, j) \oplus \bar{I}(i-1, j)) \bmod 8, \; i = 1, 2, \ldots, m-1. \tag{9}$$

$$\bar{I}(i, j) \leftarrow \bar{I}(i, j) \oplus P(i, j-1) \oplus Q(i, j-1) \oplus \bar{I}(i, j-1) <<< (Q(i, j) \oplus \bar{I}(i, j-1)) \bmod 8, \; j = 1, 2, \ldots, n-1. \tag{10}$$

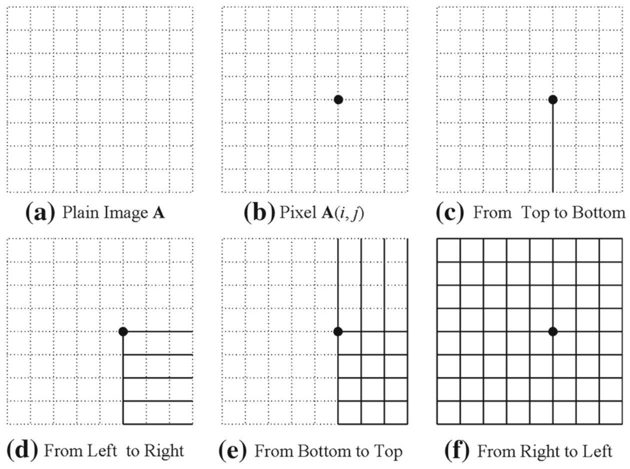

Fig. 4 Diffusion processes of the pixel $\bar{I}(i, j)$

$$\bar{I}(i-1, j) \leftarrow \bar{I}(i-1, j) \oplus P(i, j) \oplus Q(i, j) \oplus \bar{I}(i, j) >>> (Q(i-1, j) \oplus \bar{I}(i, j)) \bmod 8,\ i = m-1, \ldots, 2, 1. \tag{11}$$

$$\bar{I}(i, j-1) \leftarrow \bar{I}(i, j-1) \oplus P(i, j) \oplus Q(i, j) \oplus \bar{I}(i, j) >>> (P(i, j-1) \oplus \bar{I}(i, j)) \bmod 8,\ j = n-1, \ldots, 2, 1. \tag{12}$$

Equations (9)–(12) show the diffusion of the matrix $\bar{I}$ from four different directions, respectively, i.e., from the top to the bottom, from the left to the right, from the bottom to the top and from the right to the left. One of the benefits is that we can spread any pixel value of $\bar{I}$ to the entire cipher image as much as possible, which can improve the plain sensitivity of our algorithm. The specific diffusion processes are shown in Fig. 4. $\bar{I}(i, j)$ will be spread to the whole area of the cipher image after it is diffused from the four different directions. In Fig. 4, " $\cdot$" denotes the pixel $\bar{I}(i, j)$, "$\cdot\cdots\cdot\cdot$" means that the pixel is not diffused while " $-$ " means that the pixel is diffused.

The above analysis shows that our algorithm is a symmetric algorithm, and so the decryption process is similar to the encryption process. We can easily recover the original image by using the inverse operations of the encryption.

4 OpenCL implementation

From the above description, we know that the computational overhead of our algorithm is mainly focused on both the generation phase of R , S , P and Q and the image encryption phase. The two phases are highly suitable for parallel processing which can greatly improve the performance of our algorithm. Our parallel OpenCL implementation consists of two different kernel functions which will be described in the following two subsections.

4.1 The parallel implementation for generating the sub-key matrices

Algorithm 2. Implementation of Kernel Function $\_kernel\ void\ ChaosKeyMatGenRSPQ()$.
Input: $m, n, key_1 = (x_0, y_0, \lambda_1, \delta_1, \gamma_1, \tau_1)$,
$key_2 = (\alpha_0, \beta_0, \lambda_2, \delta_2, \gamma_2, \tau_2)$;
Output: The sub-key matrices R, S, P and Q.

```
1. $j \leftarrow get\_global\_id(0)$ for $j = 0, 1, \ldots, n-1$;
2. $(x', y') \leftarrow L^{2D}(x_0, y_0, \lambda_1, \delta_1, \gamma_1, \tau_1, j+t)$ for $t > 3000$;
3. $(\alpha', \beta') \leftarrow L^{2D}(\alpha_0, \beta_0, \lambda_2, \delta_2, \gamma_2, \tau_2, j+t)$ for $t > 3000$;
4. $(\tilde{x}, \tilde{y}) \leftarrow (x' \times a + b,\ y' \times c + d)$;
5. $(\tilde{\alpha}, \tilde{\beta}) \leftarrow (\alpha' \times a + b,\ \beta' \times c + d)$.
6. for $i \leftarrow 0$ to $m-1$;
7. $r \leftarrow L^{1D}(x', \tilde{x}, i) \times 10^{14} \bmod m$;
8. $s \leftarrow L^{1D}(y', \tilde{y}, i) \times 10^{14} \bmod n$.
9.    if $i = 0$ then
10.       $p \leftarrow L^{1D}(\alpha', \tilde{\alpha}, r) \times 10^{14} \bmod 256$;
11.       $q \leftarrow L^{1D}(\beta', \tilde{\beta}, s) \times 10^{14} \bmod 256$.
12.   else,
13.       $p \leftarrow L^{1D}(p, \tilde{\alpha}, r) \times 10^{14} \bmod 256$;
14.       $q \leftarrow L^{1D}(q, \tilde{\beta}, s) \times 10^{14} \bmod 256$.
15.   end;
16.   $R(i, j) \leftarrow r$;  $S(i, j) \leftarrow s$;
17.   $P(i, j) \leftarrow p$;  $Q(i, j) \leftarrow q$.
18. end.
```

In Algorithm 2, "*_kernel void ChaosKeyMatGenRSPQ*()" denotes the kernel function used for the generation of the four sub-key matrices R , S , P and Q. These matrices will be used in the image confusion phase and the diffusion phase of our algorithm. In this kernel function, the host parameter NDRange is set to 1, and both the *global_work_size* and the *local_work_size* are set to $[N, 0, 0]$ which means that the *NDRange* only use the first dimensional space having N work items. In addition, $L^{1\mathrm{D}}$ denotes a one-dimensional logistic map and $L^{2\mathrm{D}}$ denotes a two-dimensional logistic map.

4.2 The parallel implementation of our image encryption algorithm HCMO

Algorithm 3. Implementation of kernel function $\_kernelvoid\ imageEncryption()$.
Input: A plain image I of the matrix type $m \times n$,
and four matrices R, S, P and Q;
Output: A cipher image I_c.

```
1. $\bar{I} \leftarrow I$.
2. $j \leftarrow get\_global\_id(0)$ for $j = 0, 1, \ldots, \Delta - 1$, $\Delta = \max(m, n)$;
3. if $j < n$ then
4.    for $i \leftarrow 0$ to $m-1$;
```

```
5. $\bar{I}(i,j) \longrightarrow \bar{I}(R(i,j),j)$;
6.   end.
7. end.
8. $barrier(GLK\_GLOBAL\_MEM\_FENCE)$
// The barrier function will queue a memory fence to ensure the correct ordering of memory operations to the global memory, the same hereinafter. //
9. if $j < m$ then
10.   for $i \leftarrow 0$ to $n-1$;
11.     $\bar{I}(j,i) \longrightarrow \bar{I}(j,S(j,i))$;
12.   end.
13. end.
14. $barrier(GLK\_GLOBAL\_MEM\_FENCE)$.
15. if $j < n$ then
16.   for $i \leftarrow 1$ to $m-1$;
17. $\bar{I}(i,j) \leftarrow \bar{I}(i,j) \oplus P(i-1,j) \oplus Q(i-1,j) \oplus \bar{I}(i-1,j)$;
18. $\bar{I}(i,j) <<< (P(i,j) \oplus \bar{I}(i-1,j))$ mod 8;
19.   end.
20. end.
21. $barrier(GLK\_GLOBAL\_MEM\_FENCE)$
22. if $j < m$ then
23.   for $i \leftarrow 1$ to $n-1$;
24. $\bar{I}(j,i) \leftarrow \bar{I}(j,i) \oplus P(j,i-1) \oplus Q(j,i-1) \oplus \bar{I}(j,i-1)$;
25. $\bar{I}(j,i) <<< (Q(j,i) \oplus \bar{I}(j,i-1))$ mod 8;
26.   end.
27. end.
28. $barrier(GLK\_GLOBAL\_MEM\_FENCE)$
29. if $j < n$ then
30.   for $i \leftarrow m-1$ downto 1;
31. $\bar{I}(i-1,j) \leftarrow \bar{I}(i-1,j) \oplus P(i,j) \oplus Q(i,j) \oplus \bar{I}(i,j)$;
32. $\bar{I}(i-1,j) >>> (Q(i-1,j) \oplus \bar{I}(i,j))$ mod 8;
33.   end.
34. end.
35. $barrier(GLK\_GLOBAL\_MEM\_FENCE)$
36. if $j < n$ then
37.   for $i \leftarrow n-1$ downto 1
38. $\bar{I}(j,i-1) \leftarrow \bar{I}(j,i-1) \oplus P(j,i) \oplus Q(j,i) \oplus \bar{I}(j,i)$;
39. $\bar{I}(j,i-1) >>> (P(j,i-1) \oplus \bar{I}(j,i))$ mod 8;
40.   end.
41. end.
42. $I_c \leftarrow \bar{I}$;
43. end.
```

In Algorithm 3, "*_kernelvoidimageEncryption*()" is another kernel function used to implement the image encryption. the parameters *global_work_size* and *NDRange* are set to the same values as in Sect. 4.1.

The decryption algorithm of Algorithm 3 can be implemented as the following Algorithm 4.

Algorithm 4. Implementation of kernel function $\_kernelvoid\ imageDecryption()$.

```
Input: A cipher image $I_c$ of the matrix type $m \times n$,
       and four matrices $R_{row}^{-1}$, $S_{col}^{-1}$, $P$ and $Q$; where
       $R_{row}^{-1}$ or $S_{col}^{-1}$ denotes their row-based or column-
       based inverse permutation matrix, respectively.
Output: The plain image $I$.
1. $\bar{I} \leftarrow I_c$.
2. $j \leftarrow get\_global\_id(0)$ for $j = 0, 1, \ldots, \Delta - 1$, $\Delta = \max(m,n)$;
3. if $j < n$ then
4.   for $i \leftarrow n-1$ downto 1
5. $\bar{I}(j,i-1) \leftarrow \bar{I}(j,i-1) \oplus P(j,i) \oplus Q(j,i) \oplus \bar{I}(j,i)$;
6. $\bar{I}(j,i-1) <<< (P(j,i-1) \oplus \bar{I}(j,i))$ mod 8;
7.   end.
8. end.
9. $barrier(GLK\_GLOBAL\_MEM\_FENCE)$
10. if $j < n$ then
11.   for $i \leftarrow m-1$ downto 1;
12. $\bar{I}(i-1,j) \leftarrow \bar{I}(i-1,j) \oplus P(i,j) \oplus Q(i,j) \oplus \bar{I}(i,j)$;
13. $\bar{I}(i-1,j) <<< (Q(i-1,j) \oplus \bar{I}(i,j))$ mod 8;
14.   end.
15. end.
16. $barrier(GLK\_GLOBAL\_MEM\_FENCE)$
17. if $j < m$ then
18.   for $i \leftarrow 1$ to $n-1$;
19. $\bar{I}(j,i) \leftarrow \bar{I}(j,i) \oplus P(j,i-1) \oplus Q(j,i-1) \oplus \bar{I}(j,i-1)$;
20. $\bar{I}(j,i) >>> (Q(j,i) \oplus \bar{I}(j,i-1))$ mod 8;
21.   end.
22. end.
23. $barrier(GLK\_GLOBAL\_MEM\_FENCE)$.
24. if $j < n$ then
25.   for $i \leftarrow 1$ to $m-1$;
26. $\bar{I}(i,j) \leftarrow \bar{I}(i,j) \oplus P(i-1,j) \oplus Q(i-1,j) \oplus \bar{I}(i-1,j)$;
27. $\bar{I}(i,j) >>> (P(i,j) \oplus \bar{I}(i-1,j))$ mod 8;
28.   end.
29. end.
30. $barrier(GLK\_GLOBAL\_MEM\_FENCE)$
31. if $j < m$ then
32.   for $i \leftarrow 0$ to $n-1$;
33.     $\bar{I}(j,i) \longrightarrow \bar{I}(j,S_{col}^{-1}(j,i))$;
34.   end.
35. end.
36. $barrier(GLK\_GLOBAL\_MEM\_FENCE)$
37. if $j < n$ then
38.   for $i \leftarrow 0$ to $m-1$;
39. $\bar{I}(i,j) \longrightarrow \bar{I}(R_{row}^{-1}(i,j),j)$;
40.   end.
41. end.
42. $I \leftarrow \bar{I}$;
43. end.
```

5 Simulation experiments

In this section, the experimental results of our algorithm on the CPU and the GPU are, respectively, presented. The

Table 1 Configuration parameters of our test environment

Host-param	Values	GPU-parameters	Values
CPU	Intel i5-3470	Num. of Mp	16
Clock rate	3.2 GHz	Num. of CUDA	512
RAM	4GB	Global memory	1503 MB
GPU	GTX 580	Local memory	48 KB
OS	Windows 7 64-bit	Constant buffer	64 KB
OpenCL	1.1	Work-group sizes	1024

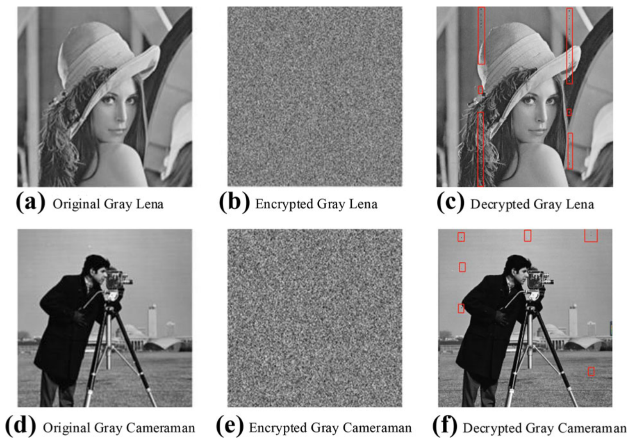
(a) Original Gray Lena
(b) Encrypted Gray Lena
(c) Decrypted Gray Lena
(d) Original Gray Cameraman
(e) Encrypted Gray Cameraman
(f) Decrypted Gray Cameraman

Fig. 5 Encrypted and decrypted results of the gray Lena and Cameraman image by our HCMO algorithm

OpenCL implementation work of our algorithm is carried out by using MATLAB R2011b on the CPU and the GPU, respectively. All the simulations are performed on a workstation computer equipped with NVIDIA GeForce GTX 580 device. The control parameter η is set 1, key is chosen as $key = (key_1, key_2)$ with $key_1 = (0.8765, 0.6123, 0.8, 0.2, 0.3, 0.4)$ and $key_2 = (0.7654, 0.4123, 0.7, 0.3, 0.2, 0.1)$, and the constant parameters used in the linear transformation Eq. (5) are chosen as $(a, b) = (0.43, 0.57)$ and $(c, d) = (0.37, 0.63)$. Table 1 shows the host's parameters and the main properties of NVIDIA GeForce GTX 580, where Host-param denotes Host-parameters, Num. of Mp denotes the number of multiprocessors (that is, Compute Units) and Num. of CUDA denotes the number of CUDA cores.

In our experiment, we choose two standard gray images "Lena" and "Cameraman" from the USC-SIPI image database[2] to demonstrate the effects of our algorithm both on the CPU and the GPU, respectively. Figure 5 shows the original images, the encrypted and decrypted results of a gray Lena image and a gray Cameraman image (512×512) by our HCMO algorithm, respectively. In Fig. 5, the parts marked in red show that the two decrypted images are somewhat different from their original images, but it does not affect our visual perception that the two decrypted images are the same as their original images, respectively.

[2] USC-SIPI image database, http://sipi.usc.edu/database/.

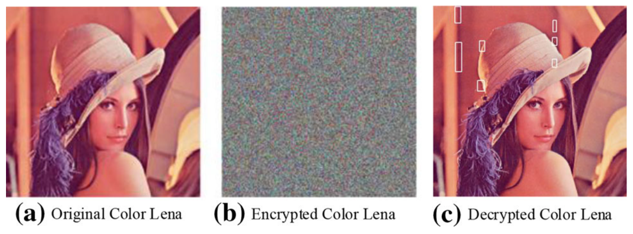

Fig. 6 Encrypted and decrypted results of a color Lena image by HCMO algorithm

In addition, our algorithm is also suitable for encrypting color images. For a given color image I, we first perform RGB three-primary decomposition on each pixel of the color image to obtain three image matrices I_R, I_G and I_B, that is, we can suppose $I = (I_R, I_G, I_B)$, and then, we apply our encryption algorithm HCMO to encrypt I_R, I_G and I_B in parallel to produce three cipher images I_{Rc}, I_{Gc} and I_{Bc}, respectively. Finally, I_{Rc}, I_{Gc} and I_{Bc} are composited into one color image to obtain the encrypted image I_c, as shown in the following equation, where $\mathrm{EN}_{\mathrm{HCMO}}(X)$ denotes encrypting an image matrix X using the algorithm HCMO, and $\mathrm{Compositing}(I_{Rc} \cup I_{Gc} \cup I_{Bc})$ denotes that the three RGB component cipher images I_{Rc}, I_{Gc} and I_{Bc} are composited into one color image.

$$\begin{aligned}
&\mathrm{EN}_{\mathrm{HCMO}}(I)\\
&\quad = (\mathrm{EN}_{\mathrm{HCMO}}(I_R), \mathrm{EN}_{\mathrm{HCMO}}(I_G), \mathrm{EN}_{\mathrm{HCMO}}(I_B))\\
&\quad = (I_{Rc}, I_{Gc}, I_{Bc}).\\
&I_c = \mathrm{Compositing}(I_{Rc} \cup I_{Gc} \cup I_{Bc}).
\end{aligned}$$

We can also get the decrypted result from the encrypted color image by the decryption algorithm Algorithm 4. Figure 6 shows the original, the encrypted and decrypted results of a color Lena image (512×512) by our HCMO algorithm, respectively.

5.1 Analyses on key space and sensitivity

As described in Sect. 3, we know that the key of our HCMO algorithm consists of two parts, that is, $key_1 =$

$(x_0, y_0, \lambda_1, \delta_1, \gamma_1, \tau_1)$ and $key_2 = (\alpha_0, \beta_0, \lambda_2, \delta_2, \gamma_2, \tau_2)$. In our algorithm, these parameters are all represented as decimal number with the precision 10^{-14} in the interval (0, 1), [0.2, 0.9] or [0.1, 0.9]. It will give a key space size $KS_1 = (10^{14} - 1)^4 (7 \times 10^{13} + 1)^4 (8 \times 10^{13} + 1)^4$. In addition, we use four parameters a, b, c and d in Eq. (5) with $a + b = 1$ and $c + d = 1$, and it will contribute a key space size $KS_2 = (10^{14} - 1)^2$. Hence, our key space size $KS = KS_1 \times KS_2 = (10^{14} - 1)^4 (7 \times 10^{13} + 1)^4 (8 \times 10^{13} + 1)^4 (10^{14} - 1)^2 \approx 56^4 \times 10^{196} \approx 2^{675}$. Therefore, our key space size is much large and can resist any brute-force attacks. Compared with AES's key space size, our key space size is more than three times larger. And our algorithm HCMO's key space size is much more larger than most of the existed chaotic-based image encryption algorithms, such as Zhang's Algorithm 2 (2013), Farajallah's second algorithm (2016), Hanis's algorithm (2019) and Mondal's algorithm (2018) with the key space size 2^{52} , 2^{124}, 2^{212} and 2^{228}, respectively.

For considering our algorithm's trade-off on security and efficiency in practical applications, we can reduce the key space based on the required security level by setting $\lambda = \delta$, or $\gamma = \tau$ or both of them. And we can also set $(a, b) = (c, d)$ to reduce our key space. In addition, we can also reduce our key space by reducing the key parameters' precision. For example, if our key parameters' precision is set 10^{-10}, then our key space size will be reduced to 2^{462}.

To ensure the safety of this key, we can secretly store the $key_1 = (x_0, y_0, \lambda_1, \delta_1, \gamma_1, \tau_1)$ and $key_2 = (\alpha_0, \beta_0, \lambda_2, \delta_2, \gamma_2, \tau_2)$ in two smart cards, respectively, or store the initial parameters $initial\_paras = (x_0, y_0, \alpha_0, \beta_0)$ and the control parameters $control\_paras = (\lambda_1, \lambda_2, \delta_1, \delta_2, \gamma_1, \gamma_2, \tau_1, \tau_2)$ in two different cards or other mediums. We can employ a fingerprint-based fuzzy vault scheme (You et al. 2017) to safely keep the key_1 and key_2.

Figure 7 shows that just a slight change in the given key will make us unable to obtain any meaningful information about the original gray Lena image from the decrypted images. It proves that our proposed image encryption algorithm is sensitive to the key. In our key sensitivity test, for the cipher image obtained by using our HCMO algorithm with the control parameter $\eta = 1$, $key_1 = (0.8765, 0.6123, 0.8, 0.2, 0.3, 0.4)$ and $key_2 = (0.7654, 0.4123, 0.7, 0.3, 0.2, 0.1)$, we have performed our HCMO's decryption algorithm (Algorithm 4) on this cipher image a lot of times by changing some parameters a little but failed to recover the original image, as shown in Fig. 7.

5.2 Encryption efficiency analysis

We simulate our algorithm for different sizes of Cameraman image and Lena image on the (OpenCL-based) CPU and the GPU (GeForce GTX 580), respectively. The encrypting efficiency is measured in terms of speedup which is the ratio of CPU time to GPU time. Table 2, respectively, lists the average speed time (in second) on the (OpenCL-based) CPU and on the GPU, and the ratio of CPU time to GPU time (in the form CPU time/GPU time/Speedup) for our HCMO algorithm, Vihari's algorithm (on the GPU-Tesla C2050) (2012), Zhang's second algorithm (shortly denoted as Zhang's) (2013), Hanis's algorithm (2019) and Mondal's algorithm (2018). Compared to the other four algorithms, our algorithm HCMO shows some greater improvement in terms of operational speedup, that is, our HCMO is some more encrypting efficiency than the others. In addition, our OpenCL-based implementation results on the CPU also indicates that our algorithm HCMO is more efficient than Vihari's algorithm and Mondal's algorithm.

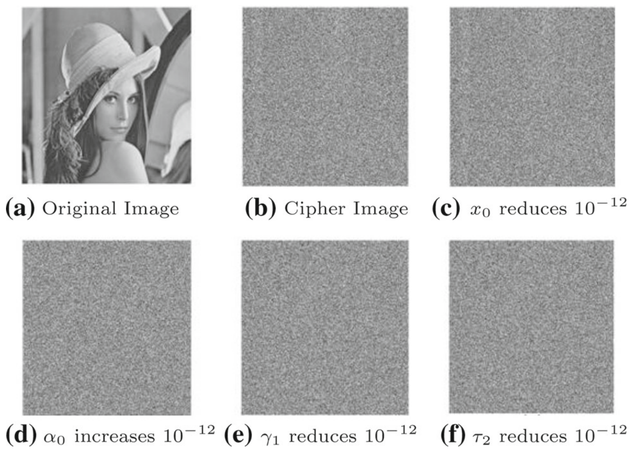
(a) Original Image
(b) Cipher Image
(c) x_0 reduces 10^{-12}
(d) α_0 increases 10^{-12}
(e) γ_1 reduces 10^{-12}
(f) τ_2 reduces 10^{-12}

Fig. 7 Key sensitivity test

5.3 Plaintext sensitivity analysis

In the digital image encryption, the number of pixel change rate (NPCR) measures the rate of different pixel numbers between two cipher images whose plain images only have one-pixel difference, while UACI (unified average changing intensity) measures the average intensity of differences between two cipher images. NPCR and UACI are usually taken as the measure standard for how tiny change of the plaintext or the key affects the cipher images. The ideal value for NPCR is 100%, while the ideal value for UACI is 33.33%. When its NPCR gets closer to 100%, the encryption algorithm is more sensitive to the plain image changing; hence, it makes the algorithm more effectively resist the known-plaintext attack and the chosen-plaintext attack. Also when its UACI gets closer to 33.33%, the encryption algorithm can more effectively resist the differential attack.

UACI and NPCR can be represented by the following formulas, respectively:

Table 2 Average speed time and speedup of our HCMO algorithm and others on the images of three different sizes

Image size	HCMO	Vihari's	Zhang's	Hanis's	Mondal's
256 × 256	0.0365/0.0086/4.2442	0.0373/0.0089/4.1910	0.0046/0.0011/4.1196	0.0267/0.0064/4.1762	0.5124/0.1240/4.1321
512 × 512	0.1034/0.0114/9.0700	0.1401/0.0162/8.6481	0.0105/0.0012/8.9632	0.0661/0.0076/8.7254	1.1307/0.1260/8.9736
1024 × 1024	0.3864/0.0241/16.0332	0.5121/0.03592/14.2567	0.0408/0.0026/15.7235	0.3042/0.0198/15.3736	4.0120/0.2576/15.5733

$$\mathrm{UACI} = \frac{1}{mn}\left(\frac{\sum_{i,j}(C_1(i,j)-C_2(i,j))}{255}\right)\times 100\%. \tag{13}$$

$$\mathrm{NPCR} = \frac{\sum_{i,j} D(i,j)}{mn} \times 100\%. \tag{14}$$

where m and n are the row number and the column number of the plain image matrices, respectively. C_1 is the cipher image matrix of the plain image that has no pixels changed. C_2 is the cipher image matrix of the plain image that has one pixel changed. $C_1(i,j)$ and $C_2(i,j)$, respectively, shows the pixel values of C_1 and C_2 at the location (i,j). $D(i,j)$ takes 0 or 1 as the following expression:

$$D(i,j) = \begin{cases} 1, & \text{if } C_1(i,j) \neq C_2(i,j), \\ 0, & \text{others.} \end{cases} \tag{15}$$

and so,

$$\sum_{i,j} D(i,j) = \sum_{C_1(i,j)\neq C_2(i,j)} 1. \tag{16}$$

By using the equations from Eqs. (13) to (16), we can compute the UACI and the NPCR of an image encryption algorithm. Tables 3 and 4 shows the UACI and the NPCR about our algorithm HCMO, Farajallah's V2-32bit algorithm (shortly denoted as Fara's) (2016), Zhang's second algorithm (shortly denoted as Zhang's) (2013), Hanis's algorithm (2019) and Mondal's algorithm (2018) on four images, respectively. They show that the UACI and the NPCP of our algorithm and the other four algorithms are all closer to the values 33% and 99% for the different images, respectively, except for that Mondal's algorithm's UACI is 37.76 (for Lena image) or 37.39 (for Peppers image) which are some larger than the ideal value 33%. Furthermore, our algorithm's UACI and NPCP are some closer to their respective ideal values than that of the four others in average. Hence, our algorithm HCMO has better plaintext sensitivity than the other four algorithms, and so it can more effectively resist the differential attack, the known-plaintext attack and the chosen-plaintext attack.

Table 3 UACI for five different images (512 × 512)

Image	HCMO	Zhang's	Fara's	Hanis's	Mondal's
Lena	33.28	–	33.46	33.47	37.76
Pirate	33.25	–	–	33.40	–
Peppers	33.50	–	33.47	33.46	37.39
Barb	33.39	33.42	33.46	–	–

Table 4 NPCR for five different images (512 × 512)

Image	HCMO	Zhang's	Fara's	Hanis's	Mondal's
Lena	99.65	–	99.608	99.60	99.6937
Pirate	99.74	–	–	99.61	–
Peppers	99.85	–	99.609	99.60	99.6971
Barb	99.61	99.58	99.609	–	–

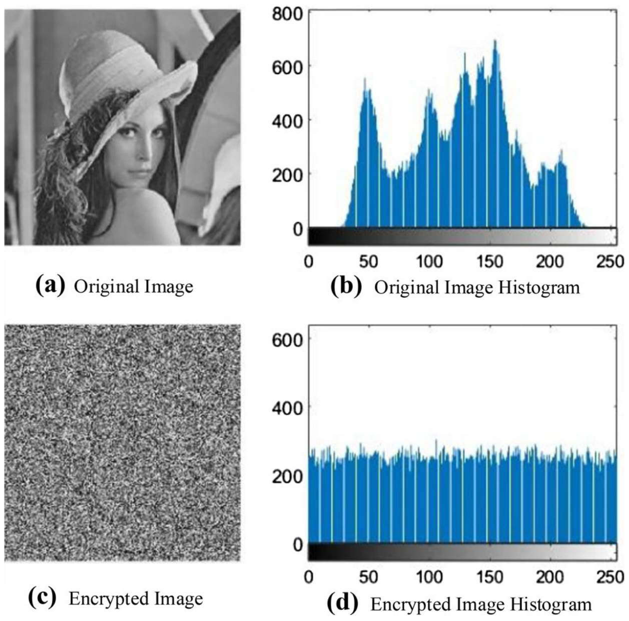

Fig. 8 Histogram analysis

5.4 Histogram analysis

A histogram shows the distribution of the pixel values of an image. The ideal histogram of the cipher image should be uniformly distributed and is significantly different from that of the plain image (Zhu et al. 2011). From Fig. 8, we know that the pixel values have been uniformly distributed after the original gray image is encrypted by our algorithm.

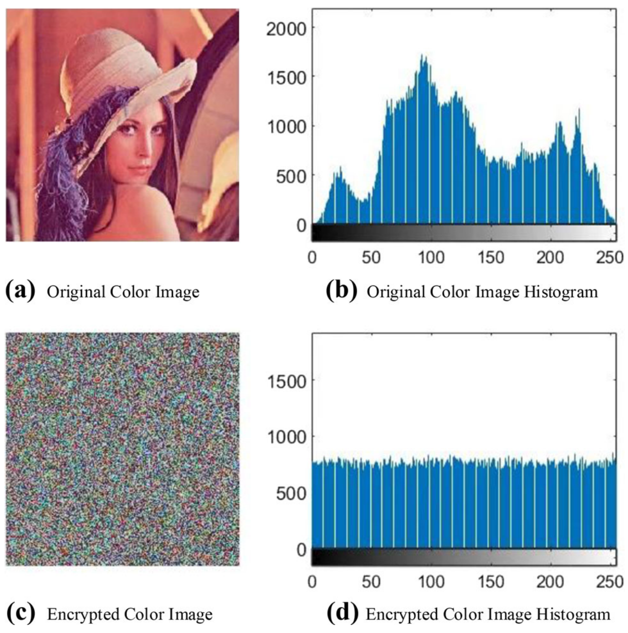

Fig. 9 Histogram analysis

Our minor revised algorithm's simulating result on Lena color image, as shown in Fig. 9, also proves that our algorithm has secure encryption effect for color image.

5.5 Correlation analysis

The pixel correlation can reflect the scrambling effect of an image encryption algorithm. We randomly select 4000 pairs of adjacent pixels from the horizontal direction, the vertical direction and the diagonal direction in the cipher image, respectively. Using the following Eqs. (17)–(20), we can easily obtain the correlation coefficients between two adjacent pixels in Lena image. The results are expressed in Table 5.

$$E(x) = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{17}$$

$$D(x) = \frac{1}{n}\sum_{i=1}^{n} [x_i - E(x)]^2 \tag{18}$$

$$\operatorname{cov}(x, y) = \frac{1}{n}\sum_{i=1}^{n} [x_i - E(x)][y_i - E(y)] \tag{19}$$

$$r_{xy} = \frac{\operatorname{cov}(x, y)}{\sqrt{D(x)}\sqrt{D(y)}} \tag{20}$$

where x and y are the gray value of two adjacent pixels. $E(x)$, $D(x)$ and $\operatorname{cov}(x, y)$ denote the expectation of x, variance of x and covariance of x and y, respectively. r_{xy} denotes the correlation coefficient between x and y. N denotes the total number of the samples.

Table 5 Correlation coefficients of the Lena image (512 × 512) for HCMO algorithm

Model	Plain image	Cipher image
Horizontal	0.9715	0.0149
Vertical	0.9855	0.0151
Diagonal	0.9587	0.0113

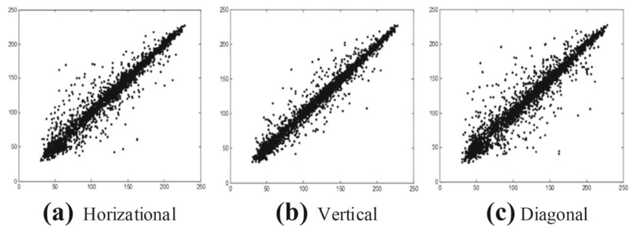

Fig. 10 Correlation plots of adjacent pixels in the plain image Lena in three directions

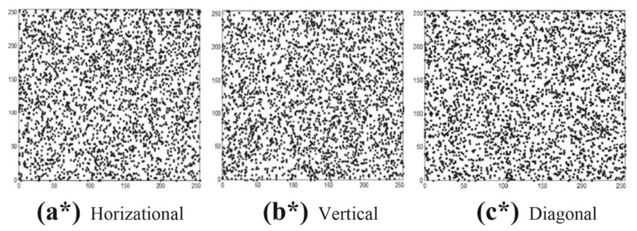

Fig. 11 Correlation plots of adjacent pixels of the Cipher image obtained by HCMO on three directions

From Table 5, we know that the correlation coefficients of the plain gray image are close to 1, while the correlation coefficients are close to 0 after the plain image is encrypted by our algorithm HCMO. As shown in Figs.10 and 11, the pixel gray values are gathered around in an oblique straight line at the horizontal direction, the vertical direction or the diagonal direction in the plain image Lena, but the pixel gray values are scattered over the entire cipher image Lena after the image Lena is encrypted at the three directions.

According to our experimental result analyses, an encrypted image by our algorithm would lost the whole intrinsic space location information of the original image, and it means that we cannot get any useful information of the original image by the visual system. Hence, our HCMO algorithm has good scrambling effect.

5.6 Information entropy analysis

Information entropy is one of the criteria to measure the strength of a cryptosystem, which was proposed by Shannon (1949). Suppose that $H(-)$ is the general information entropy of the image A, then we have

Table 6 Information entropies of the different five images (512 × 512) obtained by HCMO

Origin image	Origin entropy	Cipher entropy
Cameraman	7.0480	7.9895
Lena	7.4451	7.9457
Mandrill	7.2925	7.9316
Pirate	7.2367	7.9846
Peppers	7.5937	7.9472

$$H(T) = -\sum_{i=0}^{2^T-1} p(t_i)\log_2 p(t_i) \tag{21}$$

where $p(t_i)$ means the probability of the occurrence of t_i in I and T is the number of the bits used to represent a pixel. For an ideal random source which emits 2^8 symbols, its information entropy is 8, as given in Eq. (21). The information entropy of our algorithm can be found in Table 6. As indicated by the calculated values, the entropy values are very close to the ideal value 8, and it means that the information leakage by the cipher image is negligible and our algorithm is secure against entropy attacks.

6 Conclusions

A novel parallel image encryption algorithm HCMO is proposed based on hybrid chaotic maps and some other operations with the OpenCL-based implementation in this paper. Our algorithm can implement the position encryption and gray value encryption simultaneously. For example, for every fixed $j < n$, on one core (CUDA), if the position scrambling operations $I(i, j) \longrightarrow I(R(i, j), j)$ (for $i \leftarrow 0$ to $m - 1$) is performed, then the gray value encryption operations with Eq. (9) (for $i \leftarrow 1$ to $m - 1$) and Eq. (11) (for $i \leftarrow m - 1$ to 1) can be simultaneously performed. Hence, N such operations can be parallel performed on N cores simultaneously, respectively.

Compared to the Vihari's algorithm and some other algorithms referred in this paper, our algorithm shows remarkable improvement in terms of the speedup on the CPU or on the GPU, respectively. Our simulation results and performance analyses have also shown that our algorithm have some superior to Zhang's algorithm, Hanis's algorithm and Mondal's algorithm in resisting brute-force attack, differential attack, the known-plaintext attack and the chosen-plaintext attack. Our algorithm can also be applied to encrypt color images in parallel with the OpenCL-based implementation through some minor improvement as we described in Sect. 5.

Acknowledgements The authors wish to thank the anonymous reviewers for their insightful comments and suggestions which help improve this paper. This work is partially supported by the Key Program of the Nature Science Foundation of Zhejiang province of China (No. LZ17F020002) and the National Science Foundation of China (No. 61772166).

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

Askar S, Karawia A, Alshamrani A (2015) Image encryption algorithm based on chaotic economic model. Math Probl Eng 2015:1–10
Bhogal RS, Li BH, Gale A, Chen Y (2018) Medical image encryption using chaotic map improved advanced encryption standard. I. J Inf Technol Comput Sci 8:1–10
Chen M, Ping XJ (2006) Image steganography based on Arnold transform. Appl Res Comput 1:235–238
Eklund A, Dufort P, Forsberg D, LaConte SM (2013) Medical image processing on the GPU—past, present and future. Med Image Anal 17(8):1073–1094
Farajallah M, Assad SE, Deforges O (2016) Fast and secure chaos-based cryptosystem for images. Int J Bifurc Chaos 26(2):1–21
Fridrich J (1998) Symmetric ciphers based on two-dimensional chaotic maps. Int J Bifurc Chaos 8(6):1259–1284
Fu C, Meng WH, Zhan YF, Zhu ZL et al (2013) An efficient and secure medical image protection scheme based on chaotic maps. Comput Biol Med 43(8):1000–1010
Gupta R, Pachauri R, Singh AK (2018) An effective approach of secured medical image transmission using encryption method. Mol Cell Biomech 15(2):63–83
Hanis S, Amutha R (2019) A fast double-keyed authenticated image encryption scheme using an improved chaotic map and a butterfly-like structure. Nonlinear Dyn 95:4210–432
Hu JK, Han FL (2009) A pixel-based scrambling scheme for digital medical images protection. J Netw Comput Appl 32(4):788–794
Lee J, Yi FL, Saifullah R, Moon I (2014) Graphics processing unit-accelerated double random phase encoding for fast image encryption. Opt Eng 53(11):139–152
Lian S, Sun J, Wang Z (2005) A block cipher based on a suitable use of the chaotic standard map. Chaos Solitons Fractals 26(1):117–129
Mondal B, Kumar P, Singh S (2018) A chaotic permutation and diffusion based image encryption algorithm for secure communications. Multimed Tools Appl 77:31177–31198
Munshi A, Gaster B, Mattson TG, Fung J, Ginsburg D (2011) OpenCL programming guide. Pearson Education, Boston
Pareek NK, Patidar V, Sud KK (2006) Image encryption using chaotic logistic map. Image Vis Comput 24:926–934
Pareek NK, Patidar V, Sud KK (2013) Diffusion-substitution based gray image encryption scheme. Digit Signal Process 23(8):894–901
Patidar V, Pareek NK, Sud KK (2009) A new substitution-diffusion based image cipher using chaotic standard and logistic maps. Commun Nonlinear Sci Numer Simul 14(7):3056–3075
Qiu DY (2011) GPGPU programming techniques—from GLSL, CUDA to OpenCL. Mechanical Industry Press, Beijing
Refregier P, Javidi B (1995) Optical image encryption based on input plane and Fourier plane random encoding. Opt Lett 20(7):767–769

Rodrguez-Vzquez J, Romero-Snchez S, Crdenas-Montes M (2012) Speeding up a chaos-based image encryption algorithm using GPGPU. In: Eurocast 2011. LNCS 6927, pp 592–599

Shannon CE (1949) Communication theory of secrecy systems. Bell Syst Tech J 28(4):656–715

Vihari P, Manoj M (2012) Chaotic image encryption on GPU. In: Proceedings of the CUBE international information technology conference, New York, pp 753–758

Wang Y, Wong KW, Liao XF, Xiang T (2009) A block cipher with dynamic S-boxes based on tent map. Commun Nonlinear Sci Numer Simul 14(7):3089–3099

Wong KW, Bernie SK, Law WS (2008) A fast image encryption scheme based on chaotic standard map. Phys Lett A 372(15):2645–2652

Wu Y, Yang G, Jin H, Noonan JP (2012) Image encryption using the two-dimensional logistic chaotic map. J Electron Imaging 21(1):1–15

Xiang DS, Xiong YS (2005) Digital image scrambling based on Josephus traversing. Comput Eng Appl 10:44–46

Ye GD (2010) Image scrambling encryption algorithm of pixel bit based on chaos map. Pattern Recogn Lett 31(5):347–354

You L, Yang L, Yu WK, Wu ZD (2017) A cancelable fuzzy vault algorithm based on transformed fingerprint features. Chin J Electron 26(2):236–243

Zhang W, Wong KW, Yu H, Zhu Z (2013) An image encryption scheme using reverse 2-dimensional chaotic map and dependent diffusion. Commun Nonlinear Sci Numer Simul 18(8):2066–2080

Zhao JF, Wang S, Chang YX, Li XF (2015) A novel image encryption scheme based on an improper fractional-order chaotic system. Nonlinear Dyn 80(4):1721–1729

Zhu ZL, Zhang W, Wong KW, Yu H (2011) A chaos-based symmetric image encryption scheme using a bit-level permutation. Inf Sci 181(6):1171–1186