Available online at www.sciencedirect.com
ScienceDirect
Chaos, Solitons and Fractals 38 (2008) 1081–1092
CHAOS SOLITONS & FRACTALS
www.elsevier.com/locate/chaos

Parallel image encryption algorithm based on discretized chaotic map

Qing Zhou [a,*], Kwok-wo Wong [b], Xiaofeng Liao [a], Tao Xiang [a], Yue Hu [a]

[a] Department of Computer Science and Engineering, Chongqing University, Chongqing 400044, China
[b] Department of Electronic Engineering, City University of Hong Kong, Hong Kong, China

Accepted 2 January 2007

Abstract

Recently, a variety of chaos-based algorithms were proposed for image encryption. Nevertheless, none of them works efficiently in parallel computing environment. In this paper, we propose a framework for parallel image encryption. Based on this framework, a new algorithm is designed using the discretized Kolmogorov flow map. It fulfills all the requirements for a parallel image encryption algorithm. Moreover, it is secure and fast. These properties make it a good choice for image encryption on parallel computing platforms.

1. Introduction

In recent years, there is a rapid growth in the transmission of digital images through computer networks especially the Internet. In most cases, the transmission channels are not secure enough to prevent illegal access by malicious listeners. Therefore the security and privacy of digital images have become a major concern. Many image encryption methods have been proposed, of which the chaos-based approach is a promising direction [1–9].

In general, chaotic systems possess several properties which make them essential components in constructing cryptosystems:

(1) Randomness: chaotic systems generate long-period, random-like chaotic sequence in a deterministic way.
(2) Sensitivity: a tiny difference of the initial value or system parameters leads to a vast change of the chaotic sequences.
(3) Simplicity: simple equations can generate complex chaotic sequences.
(4) Ergodicity: a chaotic state variable goes through all states in its phase space, and usually those states are distributed uniformly.

In addition to the above properties, some two-dimensional (2D) chaotic maps are inherent excellent alternatives for permutation of image pixels. Pichler and Scharinger proposed a way to permute the image using Kolmogorov flow map

* Corresponding author.
E-mail address: tzhou@cqu.edu.cn (Q. Zhou).

0960-0779/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.chaos.2007.01.034

before a diffusion operation [1,2]. Later, Fridrich extended this method to a more generalized way [3]. Chen et al. proposed an image encryption scheme based on 3D cat maps [4]. Lian et al. proposed another algorithm based on standard map [5]. Actually, those algorithms work under the same framework: all the pixels are first permuted with a discretized chaotic map before they are encrypted one by one under the cipher block chain (CBC) mode where the cipher of the current pixel is influenced by the cipher of previous pixels. The above processes repeat for several rounds and finally the cipher-image is obtained.

This framework is very effective in achieving diffusion throughout the whole image. However, it is not suitable for running in a parallel computing environment. This is because the processing of the current pixel cannot start until the previous one has been encrypted. The computation is still in a sequential mode even if there is more than one processing element (PE). This limitation restricts its application platform since many devices based on FPGA/CPLD or digital circuits can support parallel processing. With the parallel computing technique, the speed of encryption is greatly accelerated.

Another shortcoming of chaos-based image encryption schemes is the relatively slow computing speed. The primary reason is that chaos-based ciphers usually need a large amount of real number multiplication and division operations, which cost vast of computation. The computational efficiency will be increase substantially if the encryption algorithms can be executed on a parallel processing platform.

In this paper, we propose a framework for parallel image encryption. Under such framework, we design a secure and fast algorithm that fulfills all the requirements for parallel image encryption. The rest of the paper is arranged as follows. Section 2 introduces the parallel operating mode and its requirements. Section 3 presents the definitions and properties of four transformations which form the encryption/decryption algorithm. In Section 4, the processes of encryption, decryption and key scheduling will be described in detail. Experimental results and theoretical analyses are provided in Sections 5 and 6, respectively. Finally, we conclude this paper with a summary.

2. Parallel mode

2.1. Parallel mode and its requirements

In parallel computing mode, each PE is responsible for a subset of the image data and possesses its own memory. During the encryption, there may be some communication between PEs (see Fig. 1).

To allow parallel image encryption, the conventional CBC-like mode must be eliminated. However, this will cause a new problem, i.e. how to fulfill the diffusion requirement without such mode. Besides, there arise some additional requirements for parallel image encryption:

1. Computation load balance
   The total time of a parallel image encryption scheme is determined by the slowest PE, since other PEs have to wait until such PE finishes its work. Therefore a good parallel computation mode can balance the task distributed to each PE.
2. Communication load balance
   There usually exists lots of communication between PEs. For the same reason as of computation load, the communication load should be carefully balanced.

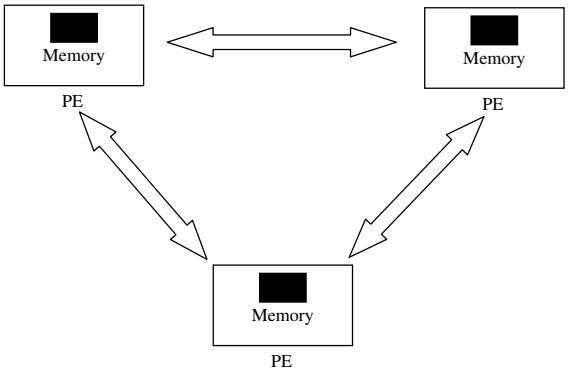

Fig. 1. Parallel computing mode for image encryption.

3. Critical area management
   When computing in a parallel mode, many PEs may read or write the same area of memory (i.e. critical area) simultaneously, which often causes unexpected execution of the program. It is thus necessary to use some parallel techniques to manage the critical area.

2.2. A parallel image encryption framework

To fulfill the above requirements, we propose a parallel image encryption framework, which is a four-step process:

Step 1: The whole image is divided into a number of blocks.
Step 2: Each PE is responsible for a certain number of blocks. The pixels inside a block are encrypted adequately with effective confusion and diffusion operations.
Step 3: Cipher-data are exchanged via communication between PEs to enlarge the diffusion from a block to a broader scope.
Step 4: Go to step 2 until the cipher image reaches the required level of security.

In step 2, diffusion is achieved, but only within the small scope of one block. With the aid of step 3, however, such diffusion effect is broadened. Note that from the cryptographic point of view, data exchange in step 3 is essentially a permutation. After several iterations of steps 2 and 3, the diffusion effect is spread to the whole image. This means that a tiny change in one plain-image pixel will spread to a substantial amount of pixels in the cipher-image. To make the framework sufficiently secure, two requirements must be fulfilled:

1. The encryption algorithm in step 2 should be sufficiently secure with the characteristic of confusion and diffusion as well as sensitivity to both plaintext and key.
2. The permutation in step 3 must spread the local change to the whole image in a few rounds of operations.

The first requirement can be fulfilled by a combination of different cryptographic elements such as *S*-box, Feistel-structure, matrix multiplications and chaos map, etc., or we can just use a conventional cryptographic standard such as AES or IDEA. The second one, however, is a new topic resulted from this framework. Furthermore, such permutation should help to achieve the three additional goals presented in Section 2.1. Hence, the permutation operation is one of the focuses of this paper and should be carefully studied.

Under this parallel image encryption framework, we propose a new algorithm which is based on four basic transformations. Therefore, we will first introduce those transformations before describing our algorithm.

3. Transformations

3.1. *A*-transformation

In *A*-transformation, '*A*' stands for addition. It can be formally defined as follow:

$$a + b = c \tag{1}$$

where $a, b, c \in G$, $G = GF(2^8)$, and the addition is defined as the bitwise XOR operation.

The transformation A has three fundamental properties:

$$\begin{aligned} &(2.1)\quad a + a = 0 \\ &(2.2)\quad a + b = b + a \\ &(2.3)\quad (a + b) + c = a + (b + c) \end{aligned} \tag{2}$$

3.2. *M*-transformation

In *M*-transformation, '*M*' stands for mixing of data.

First, we introduce the sum transformation:

$$sum : G^{m \times n} \to G$$

where

$$G^{m\times n}=\left\{ I \middle| I=\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ & \cdots & \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}, a_{ij}\in G \right\}$$

then $sum(I)$ is defined as:

$$sum(I)=\sum_i \sum_j a_{ij} \tag{3}$$

Now we give the definition of M-transformation as follows:

$$M: G^{m\times n}\to G^{m\times n}$$

Let $M(I)=C$ where $I=(a_{ij})_{m\times n}$, $C=(c_{ij})_{m\times n}$, then

$$c_{i,j}=a_{i,j}+sum(I) \tag{4}$$

It is easy to prove the following properties of the M-transformation:

$$\begin{aligned} &(5.1)\quad M(M(I))=I \\ &(5.2)\quad M(I+J)=M(I)+M(J) \\ &(5.3)\quad M(kI)=kM(I),\quad \text{where } kI=\sum_{i=1}^{k} I, k\in N \end{aligned} \tag{5}$$

It should be noted that all the addition operations from (3)–(5) are the A-transformation indeed.

3.3. S-transformation

In S-transformation, 'S' stands for S-box substitution. There are lots of ways to construct an S-box, among which the chaotic approach is a good candidate. For example, Tang *et al* presented a method to design S-box based on discretized logistic map and Baker map [10]. Following this work, Chen et al. proposed another method to obtain an S-box, which leads to a better performance [11]. The process is described as follows:

Step 1: Select an initial value for the Chebyshev map. Then iterate the map to generate the initial S-box table.
Step 2: Pile up the 2D table to a 3D one.
Step 3: Use the discretized 3D Baker map to shuffle the table for many times. Finally, transform the 3D table back to 2D to obtain the desired S-box.

Experimental results show that the resultant S-box is ideal for cryptographic applications. The approach is also called 'dynamic' as different S-boxes are obtained when the initial value of Chebyshev map is changed. However, for the sake of simplicity and performance, we use a fixed S-box, i.e. the example given in [11] (see Table 1).

3.4. K-transformation

In K-transformation, 'K' stands for Kolmogorov flow, which is often called generalized Baker map [3]. The application of Kolmogorov flow for image encryption was first proposed by Pichler and Scharinger [1,2].

The discrete version of K-flow is given by (6):

$$T_{n,\delta}(x,y)=(q_s(x-F_s)+(y \bmod q_s), F_s+(y \operatorname{div} q_s)) \tag{6}$$

where $\delta=(n_1,n_2,\ldots,n_k)$, n_s is an positive integer, $\sum_s n_s=N$ and n_s divide N for all s, $p_s=1/n_s$, while F_s is still the left bound of the vertical strip s:

$$F_s=\begin{cases} 0 & s=1 \\ n_1+n_2+\cdots+n_{s-1} & s=2,3,\ldots,t-1 \end{cases}$$

Note that the Eq. (6) can be interpreted by the geometrical transformation shown in Fig. 2. The $N\times N$ image is first divided into vertical rectangles of height N and width n_s. Then each vertical rectangle is further divided into boxes of height p_s and width n_s. After K-transformation, pixels from the same box are actually mapped to a single row.

Table 1
The proposed *S*-box is the example given in [11]

161	85	129	224	176	50	207	177	48	205	68	60	1	160	117	46
130	124	203	58	145	14	115	189	235	142	4	43	13	51	52	19
152	153	83	96	86	133	228	136	175	23	109	252	236	49	167	92
106	94	81	139	151	134	245	72	172	171	62	79	77	231	82	32
238	22	63	99	80	217	164	178	0	154	240	188	150	157	215	232
180	119	166	18	141	20	17	97	254	181	184	47	146	233	113	120
54	21	183	118	15	114	36	253	197	2	9	165	132	204	226	64
107	88	55	8	221	65	185	234	162	210	250	179	61	202	248	247
213	89	101	108	102	45	56	5	212	10	12	243	216	242	84	111
143	67	93	123	11	137	249	170	27	223	186	95	169	116	163	25
174	135	91	104	196	208	148	24	251	39	40	31	16	219	214	74
140	211	112	75	190	73	187	244	182	122	193	131	194	149	121	76
156	168	222	34	241	70	255	229	246	90	53	225	100	30	37	237
103	126	38	200	44	209	42	29	41	218	71	155	78	125	173	28
128	87	239	3	191	158	199	138	227	59	69	220	195	66	192	230
198	26	159	6	127	201	144	206	98	33	35	7	105	147	57	110

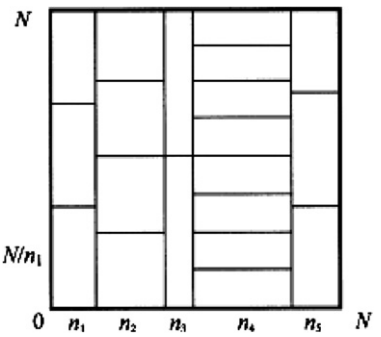

Fig. 2. Pixels in each box will be stretched into a single row by discretized Kolmogorov flow.

4. "MASK" – a parallel image encryption scheme

4.1. Outline of the proposed encryption scheme

Assume the $N \times N$ image is encrypted by n PEs simultaneously, we describe the parallel encryption scheme as follows:

1. Each PE is responsible for some fixed rows of pixels in the image.
2. Pixels of each row are encrypted using transformation M, A, S, respectively.
3. Permute all the pixels according to transformation K to have further diffusion.
4. Go to step 2 for another round of encryption until the cipher is sufficiently secure.

As mentioned above, the permutation in step 3 is very important as it helps to fulfil both the security and speed requirements. Therefore both the permutation map and its parameters must be carefully chosen. In our algorithm, δ is a constant vector with length q, where $q = N/n$. Each element of the vector is equal to n:

$$\delta = (n, n, \ldots, n)_{1,q} \tag{7}$$

Each PE is responsible for q consecutive rows, or more specifically, the ith PE is responsible for rows from $(i-1) * q$ to $i * q - 1$. This algorithm can fulfil all the requirements for parallel encryption, as analyzed below.

1. Diffusion effect in the whole image
   Assume that the operations in step 2 are sufficiently secure. After step 2, a tiny change of the plain pixel will diffuse to the whole row of N pixels. If we choose δ according to Eq. (7), it is easy to prove that those N cipher pixels will be

permuted to different q rows with the help of K-transformation in step 3. In the same way, after another round of encryption, the change is spread to q rows, and after the third round, the whole cipher image is changed. Consequently, in our scheme, the smallest change of any single pixel will diffuse to the whole image in 3 rounds.

2. Balance of communication load
   If the parameter δ of (6) is chosen as (7), it is easy to prove that the data exchanged between two PEs are constant, i.e., equal to $1/q^2$ of the total number of image pixels. For each PE, this quantity becomes $(q-1)/q^2$. Therefore, in our scheme, the communication load of each PE is equivalent, and there is no unbalance of communication load for the PEs at all.
3. Balance of computation load
   The data to be encrypted by each PE is equally q rows of pixels; hence computation load balance is achieved naturally.
4. Critical area management
   In our scheme, under no circumstances would two PEs read from or write to the same memory. Therefore, we do not need to impose any critical area management technique in our scheme as other parallel computation schemes often do.

The above discussions have shown that the proposed scheme fulfils all the requirements for parallel image encryption, which is mainly attributed to the chaotic Kolmogorov map and the choice of its parameters.

4.2. Cipher

The cipher is made up of a number of rounds. However, before the first round, the image is pre-processed with a K-transformation. Then in each round, the transformation M, A, S, K is carried out, respectively. The final round differs slightly from the previous rounds in that the S-transformation is omitted on purpose. The transformations M, A, S operate on one row of pixels by each PE, while the transformation K operates on the whole image which necessarily involves communication between PEs. The cipher is described by the pseudo-code listed in Fig. 3.

4.3. Round key generation

Among the four transformations, only transformation A needs a round key. For an 8-bit grey level image of $N \times N$ pixels, a round key containing N bytes should be generated for transformation A in each round.

Generally speaking, the round keys should be pseudo-random and key-sensitive. From this point of view, a chaotic map is a good alternative. In our scheme, we use the skew tent map to generate the required round keys.

$$x' = \begin{cases} x/\mu & 0 < x \leqslant \mu \\ (1-x)/(1-\mu) & \mu < x < 1 \end{cases} \tag{8}$$

```
Cipher(I)
BEGIN
    //pre-processing
    K(I);
    //rounds
    FOR i = 1 to ROUNDS-1
        M(I);
        A(I);
        S(I);
        K(I);
    END
    //final round
    M(I);
    A(I);
    K(I);
END
```

Fig. 3. Pseudo-codes for the 'MASK' cipher.

The chaotic sequence is determined by the system parameter μ and initial state x_0 of the chaotic map either of which is a real number between 0 and 1. Although the chaotic map equation is simple, it generates pseudo-random sequences that are sensitive to both the system parameter and the initial state. This property makes the map an ideal choice for key generation.

When implemented in a digital computer, the state of the map is stored as a floating point number. The first 8 bits of each state are extracted as one byte of the round key. Accordingly, we need to iterate the skew tent map for N times in each round.

4.4. Decipher

In general, the decryption procedures are composed of a reversed order of the transformations performed in encryption. This property also holds in our scheme. However, with careful design, the decryption process of our scheme can have the same, rather than the reversed, order of transformations as the cipher. This impressive characteristic attributes to two properties of the transformations:

(1) Transformation-S and transformation-K are commutable. Transformation-S substitutes only the value of each pixel and is independent of its position. On the other hand, transformation-K changes only a pixel's position with its value unchanged. Consequently, the relation between the two transformations can be expressed in (9):

$$K(S(I)) = S(K(I)) \tag{9}$$

(2) Transformation-M is a linear operation according to (5). Moreover, the addition defined in (5.2) is actually transformation A. Thus the relation between the two transformations can be expressed in (10):

$$M(A(I,J)) = A(M(I),M(J)) \tag{10}$$

In short, either transformations S and K, or transformations M and A can interchange their computation order with no influence on the final result. Table 2 illustrates how these two properties affect the order of the transformations of decipher in a simple example of 2-round cipher.

It is easy to observe that for a cipher composed of multiple rounds, the decipher process still has the same sequence of transformations as the cipher. Hence, both the cipher and decipher share the same framework.

However, there are still some slight differences between the encryption and decryption processes:

(1) The round keys used in decipher is in a reversed order of that in cipher, and those keys should be first applied the transformation M.
(2) The transformation K and S in decipher should use their inverse transformations.

However, since transformation K and S can both be implemented by look-up table operations, their inverse transformations differ just in content of look-up tables. Consequently, all above difference in computations can be translated into difference in data.

The symmetric property makes our scheme very concise. It also reduces lots of codes for a computer system implementing both the cipher and decipher. For hardware implementation, this property results in a reduction of cost for both devices.

Table 2
The process of equivalent decipher in 2-round encryption

No.	Cipher	Decipher	Equivalent decipher
1	K	K	K
2	M	A	M
3	A	M	A
4	S	K	S
5	K	S	K
6	M	A	M
7	A	M	A
8	K	K	K

The remarkable structure of our scheme looks more concise and saves a lot of codes during implementation. This is definitely an advantage when compared with other chaos-based ciphers.

5. Experimental results

In this section, an example is given to illustrate the effectiveness of the proposed algorithm. In the experiment, a grey level image 'Lena' of size 256 * 256 pixels, as shown in Fig. 4a, is chosen as the plain-image. The number of PEs is chosen as 4. The key of our system, i.e. the initial state x_0 and the system parameter μ are stored as floating point number with a precision of 56-bits. In the example presented here, $x_0 = 0.12345678$, and $\mu = 1.9999$.

When the encryption process is completed, the cipher-image is obtained and is shown in Fig. 4b. Typically, we encrypt the plain-image for 9 rounds as recommended.

5.1. Histogram

Histogram of the plain-image and the cipher-image is depicted in Figs. 4c and d, respectively. These two figures show that the cipher-image possesses the characteristic of uniform distribution in contrast to that of the plain-image.

5.2. Correlation analysis of two adjacent pixels

The correlation analysis is performed by randomly select 1000 pairs of two adjacent pixels in vertical, horizontal, and diagonal direction, respectively, from the plain-image and the ciphered image. Then the correlation coefficient of the pixel pair is calculated and the result is listed in Table 3. Fig. 5 shows the correlation of two horizontally adjacent pixels. It is evident that neighboring pixels of the cipher-image has little correlation.

5.3. NPCR analysis

NPCR means the change rate of the number of pixels of the cipher-image when only one pixel of the plain-image is modified. In our example, the pixel selected is the last pixel of the plain-image. Its value is changed from $(01101111)_2$ to $(01101110)_2$. Then the NPCR at different rounds are calculated and listed in Table 4. The data show that the performance is satisfactory after 3 rounds of encryption. The different pixels of the two cipher-images after 9 rounds are plotted in Fig. 6.

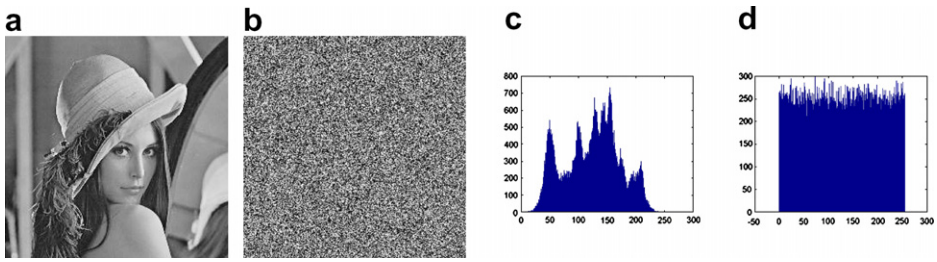

Fig. 4. (a) Plain-image, (b) cipher-image, (c) histogram of plain-image, (d) histogram of cipher-image.

Table 3
Correlation coefficient of two adjacent pixels in plain-image and cipher-image

	Plain-image	Cipher-image
Horizontal	0.9803	−0.0124
Vertical	0.9583	−0.0273
Diagonal	0.9420	0.0074

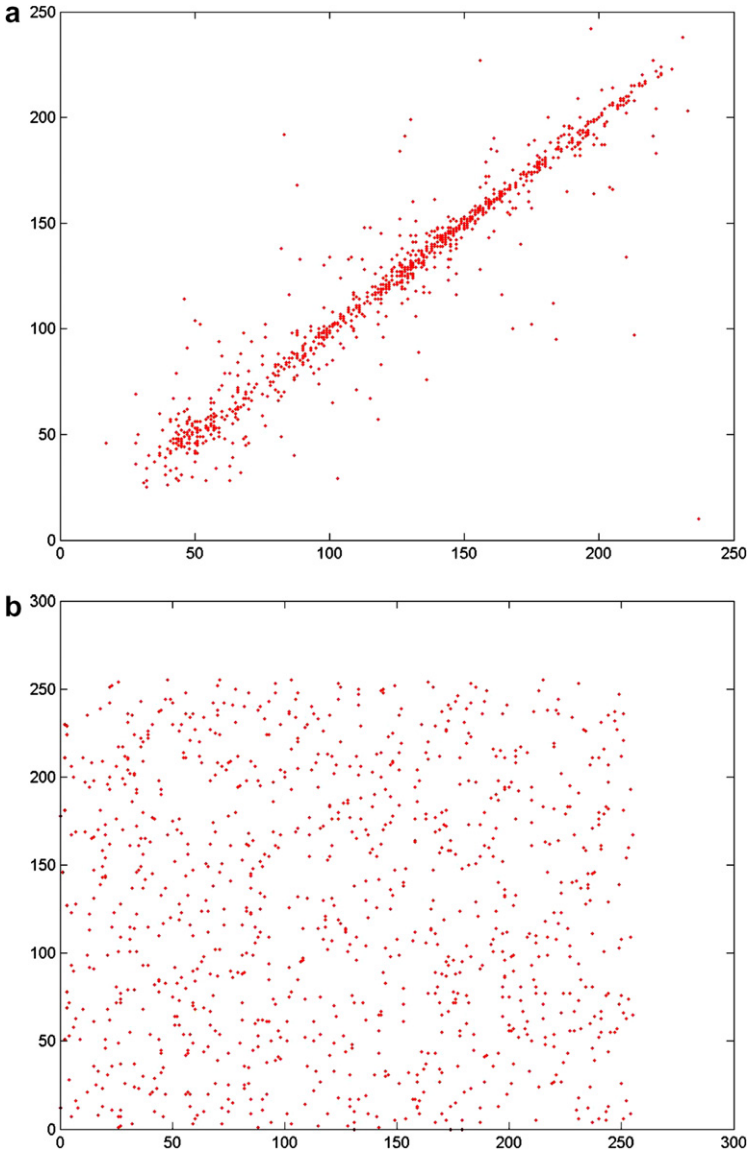

Fig. 5. Correlation of two horizontally adjacent pixels of (a) plain image; (b) cipher image, x-coordinate and y-coordinate is the grey level of two neighbor pixels, respectively.

Table 4
NPCR of two cipher-images at different rounds

Round	1	2	3	4	5	6	7	8	9
NPCR	0.0039	0.2500	0.9961	0.9962	0.9961	0.9962	0.9959	0.9960	0.9964

5.4. UACI analysis

The unified average changing intensity (UACI) index measures the average intensity of differences between two images. Again, we make the same change as in Section 5.3 and calculate the UACI between two cipher-images. The results are shown in Table 5. After three rounds of encryption, the UACI is converged to 1/3. It should be noticed that

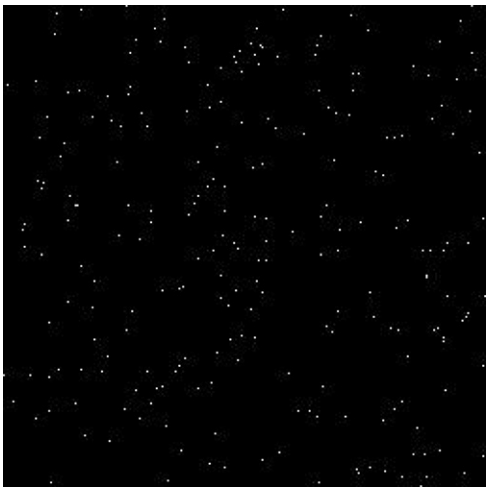

Fig. 6. Difference between two ciphered images after 9 rounds. White points (about 1/256 of the total pixels) indicate the positions where pixels of the two cipher-images have the same values.

Table 5
UACI of two cipher-images

Round	1	2	3	4	5	6	7	8	9
UACI	0.0014	0.0846	0.3357	0.3331	0.3340	0.3328	0.3330	0.3319	0.3343

the average error between two random sequences uniformly distribution in $[0,1]$ is 1/3 if they are completely uncorrelated with each other.

6. Security and performance analysis

6.1. Diffusion

The NPCR analysis has revealed that when there is only 1 bit of the plain pixel changed, almost all the cipher pixels become different although there is still a low possibility of 1/256 that two cipher pixels are equal. This diffusion first attributes to transformation S since change of any bit of the input to the S-box influences all the output bits at a change rate of 50%. Then, the diffusion is spread out to the whole row by transformation M. Finally, transformation K helps to enlarge the diffusion to 25% of the image in 2 rounds, and to the whole image in 3 rounds.

6.2. Confusion

The histogram and correlation analyses of adjacent pixels both indicate that our scheme possesses a good property of confusion. This mainly results from the pseudo-randomness of the key schedule and transformations M and A. They work together to introduce the random-like effect to the cipher image. Transformation K is also helpful in destroying the local similarity of the plain-image.

6.3. Brute-force attack

The proposed scheme uses both the initial state x_0 and the system parameter μ as the secret key whose total number of bits is 112. It is by far very safe for ordinary business applications. Therefore, our scheme is strong enough to resist brute-force attack. Moreover, it is very easy to increase the number of bits for both x_0 and μ.

6.4. Other security issues

Someone may argue that, in our scheme, transformation K is not governed by any key. However, this reduces little security of our scheme. As a matter of fact, most conventional encryption algorithms such as DES and AES use public permutations. There are at least two reasons for it. First, permutations governed by key slow down the speed of encryption, for it costs time to generate those permutations from the key. Secondly and the foremost, weak permutations may be generated from some keys, which harm to the security of the system. Actually, a permutation that helps to achieve diffusion and confusion is a better alternative. More specifically, in a parallel encryption system, if the permutation helps to achieve computation and communication load balance, it is a good alternative. From this point of view, transformation K is a proper choice.

6.5. Performance analysis

The proposed algorithm runs very fast as there are only logical XOR and table lookup operations in the encryption and decryption processes. Although multiplications and divisions are required in transformation K, the transformation is fixed once the number of PE is fixed. Hence, they can be pre-computed and stored in a lookup table. More accurately, there are only 3 XOR operations (2 for transformation M and 1 for transformation A) and 2 lookup table operations (1 for transformation S and 1 for transformation K) for each pixel in each round. On the contrary, for the simplest logistic map with 56-bit precision state variable, one multiplication costs about 28 additions in average. In our key schedule, multiplications are also required in the skew tent map. However, there are only N such multiplications in each round, and hence an average of $1/N$ multiplications for each pixel per round. Furthermore, when the algorithm runs on a parallel platform, the performance can increase nearly n times than ordinary sequential image encryption scheme. Therefore, as far as the performance is concerned, our scheme is superior than existing ones.

7. Conclusion

In this paper, we introduced the concept of parallel image encryption and presented several requirements for it. Then a framework for parallel image encryption was proposed and a new algorithm was designed based on this framework. The proposed algorithm is successful in accomplishing all the requirements for a parallel image encryption algorithm with the help of discretized Kolmogorov flow map. Moreover, both the experimental results and theoretical analyses show that the algorithm possesses high security. The proposed algorithm is also fast; there are only a couple of XOR operations and table lookup operations for each pixel. Finally, the decryption process is identical to that of the cipher. Taking into account all the virtues mentioned above, the proposed algorithm is a good choice for encrypting images in a parallel computing platform.

Acknowledgements

The work is supported by Program for New Century Excellent Talents in University, the National Nature Science Foundation of China under Grant 60573047, the National Science Foundation of Chongqing under Grant 8509 and 8986-3, Chongqing University Postgraduates' Science and Innovation Fund, under Grant 200609Y1B0150174.

References

[1] Pichler F, Scharinger J. Ciphering by Bernoulli shifts in finite Abelian groups. Contributions to general algebra. Proc. Linz-conference 1994. p. 465–76.
[2] Scharinger J. Fast encryption of image data using chaotic Kolmogorov flows. J Electron Imag 1998;7(2):318–25.
[3] Fridrich J. Symmetric ciphers based on two-dimensional chaotic maps. I J Bifur Chaos 1998;8(6):1259–64.
[4] Chen G, Mao Y, Chui C. Symmetric image encryption scheme based on 3D chaotic cat maps. Chaos, Solitons & Fractals 2004;21(3):749–61.
[5] Lian S, Shun J, Wang Z. A block cipher based on a suitable use of the chaotic standard map. Chaos, Solitons & Fractals 2005;26(1):117–29.
[6] Guan Z, Huang F, Guan W. Chaos-based image encryption algorithm. Phys Lett A 2005;346(1–3):153–7.
[7] Zhang L, Liao X, Wang X. An image encryption approach based on chaotic maps. Chaos, Solitons & Fractals 2005;24(3):759–65.
[8] Gao H, Zhang Y, Liang S, Li D. A new chaotic algorithm for image encryption. Chaos, Solitons & Fractals 2006;29(2):393–9.

[9] Pareek NK, Patidar V, Sud KK. Image encryption using chaotic logistic map. Image Vision Comput 2006;24(9):926–34.
[10] Tang Guoping, Liao Xiaofeng, Chen Yong. A novel method for designing *S*-boxes based on chaotic maps. Chaos, Solitons & Fractals 2005;23:413–9.
[11] Chen G, Chen Y, Liao X. An extended method for obtaining *S*-boxes based on three-dimensional chaotic Baker maps. Chaos, Solitons & Fractals 2007;31(3):571–9.