On Some Parallel Implementations of Chaotic maps

Introduction

- Microprocessors based on single processing were the industry standard for very long period of time. But unfortunately they have a hindrance because of heat dissipation and energy consumption issues. These problems restrict enhancement of CPU clocks which means slowing down the number of tasks that could be performed within each clock period. The solution to this problem is to change the architecture in order to use multiple processing units known as cores.

Encryption

In recent years a great deal of concerns have been raised for the security of the information transmitted or stored open channels. This problem can be solved by using an effective method to encrypt information so that only the authorized users with specific key could decrypt the information .

- The chaotic systems have excellent confusion and diffusion properites. And also are sensitive to the initial conditions and control parameters.
- Some of the cryptographic systems are based on single core CPU's
- This type of system has major flaws the serial algorithms cannot take advantage in processing large scale files and It's also important for the algorithm to decrypt and encrypt data as fast as possible.

The Approach

Two approaches have become predominant in parallel programming models:Open Multiprocessing

1. (OpenMP) for the shared memory and Message
2. Passing Interface (MPI) for the distributed memory.

OpenMP provides applicaton-oriented synhronization primitives which makes it easy for writing parallel programs.

Chaotic cryptosystem

The chaotic cryptosystem use shared memory model which distribute that control threads in partitions and distribute the data. The algorithm has main thread for initialization of specific parameters of the multi core system and a logistic map before encrypting the data. This parallel process is made possible using the C programming language and the library Open MP. Also the main thread is responsible for the input and output of the encryption/decryption.

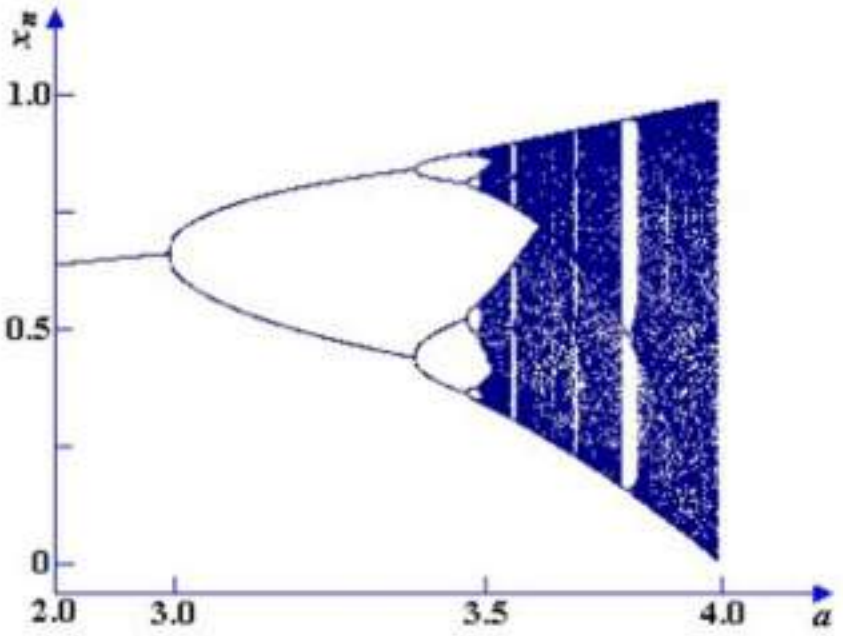
Formula

$$x_{n+1} = a \times x_n \times (1 - x_n) .$$

Formula explanation

X(n) -has a range in the interval of (0,1)
A-is the growth parameter which has a range in the interval(0,4)
If A has a value of 3.569946 this offsets a chaotic behavior .
Because after A>3.569946 the function becomes erratic and chaotic.
An interesting thing to note is that it will reach every possible point
for every pair of elements between (0,1)

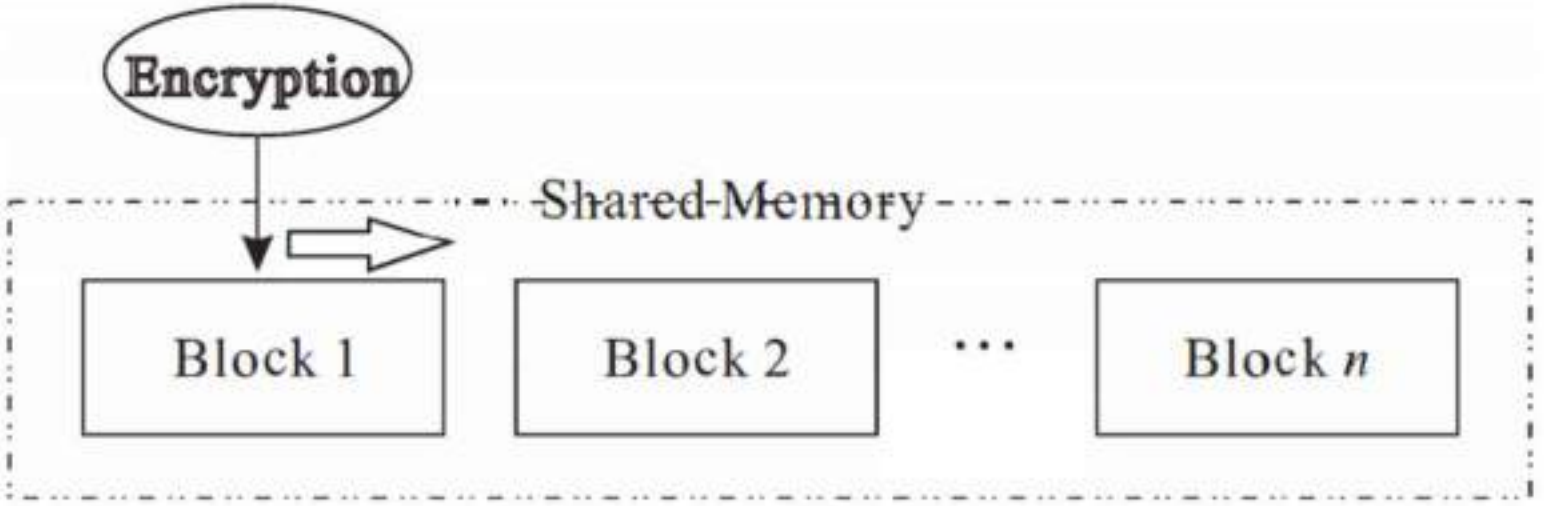
Logistic map diagram



Different types of Encryption

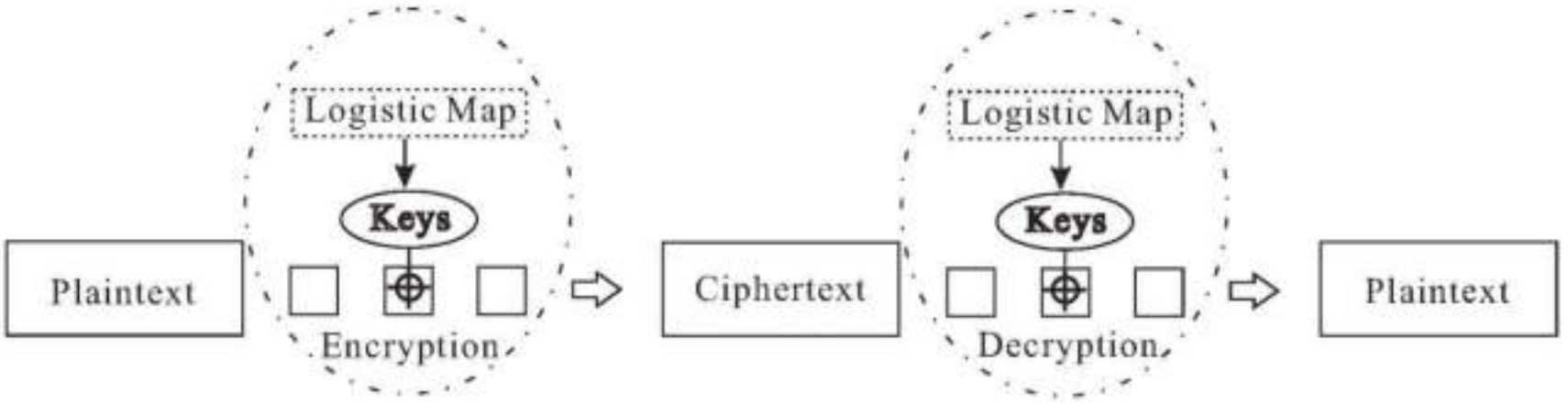
The diagram bellow shows that they are encrypted in order.

But for the diagram on the left. Each thread is assigned to specific block staring for example with thread 1 processing block one until we have more blocks that threads in which case we start over from thread one.


Encryption
Shared Memory
Block 1
Block 2
...
Block n

Cryptosystem

The diagram below shows that if we take plain text and it goes through encryption it will be ciphered. And the only way It can be deciphered is using the logistic map program for decryption and getting the keys.


Logistic Map
Keys
Plaintext
Encryption
Ciphertext
Logistic Map
Keys
Decryption
Plaintext

Parallel Algorithm code

```
#pragma omp parallel private(OpenMP _ MyJD,
block_i)
{
OpenMP _ MyJD = omp _geUhread_ numO;
OpenMP _NumThread = omp_get_num rocsO;
for (block_i OpenMP _ MyID;
block _ i<N umBlock;
block_i = block_i + OpenMP _NumThread)
EncryptData(Key _i, block j,OpenMP _ MyJD);
}
}
```

Encryptic transformation

This diagram shows how the logistic map creates the keys.User's key is converted into the control parameterOf the initial value X(o) in the logistic map

1. The Logistic map 1 generates the keys which are transformed into the control parameter and the initial value of X(o) in the logistic map .

The total number of the keys generated by the logistic map 1 is equal to the total number of the blocks. The threads encrypt the blocks by using the logistic map 2 and Key _i (i = 1,2, ... , n). Every thread corresponds to the real core in the multicore processor.

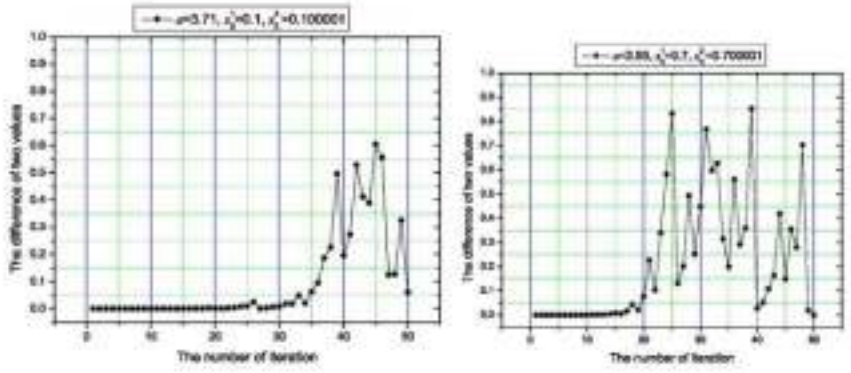
Experimental analysis

It’s very important not to downgrade system performance and consequently encryption security that why It is important to use data type double to get a precise output of the chaotic map.Fig. 6 shows the difference of the initial value X(o)between the case of X(o) and the case of X(o) + 10-6 for a =3.71 and a = 3.93. Let Nbe the number of iteration.

The horizontal axis represents the number of iterations(N=50) the vertical the differences between two values.

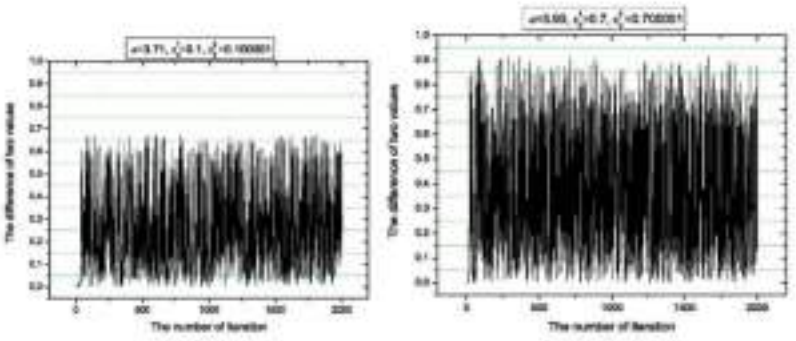
From Fig. 7 we can see that the difference between the two values of the logistic map has changed nonunifonnity when setting larger value of the control parameter a. In general, we can easily obtain the difference sequences by using the logistic map.

Diagrams



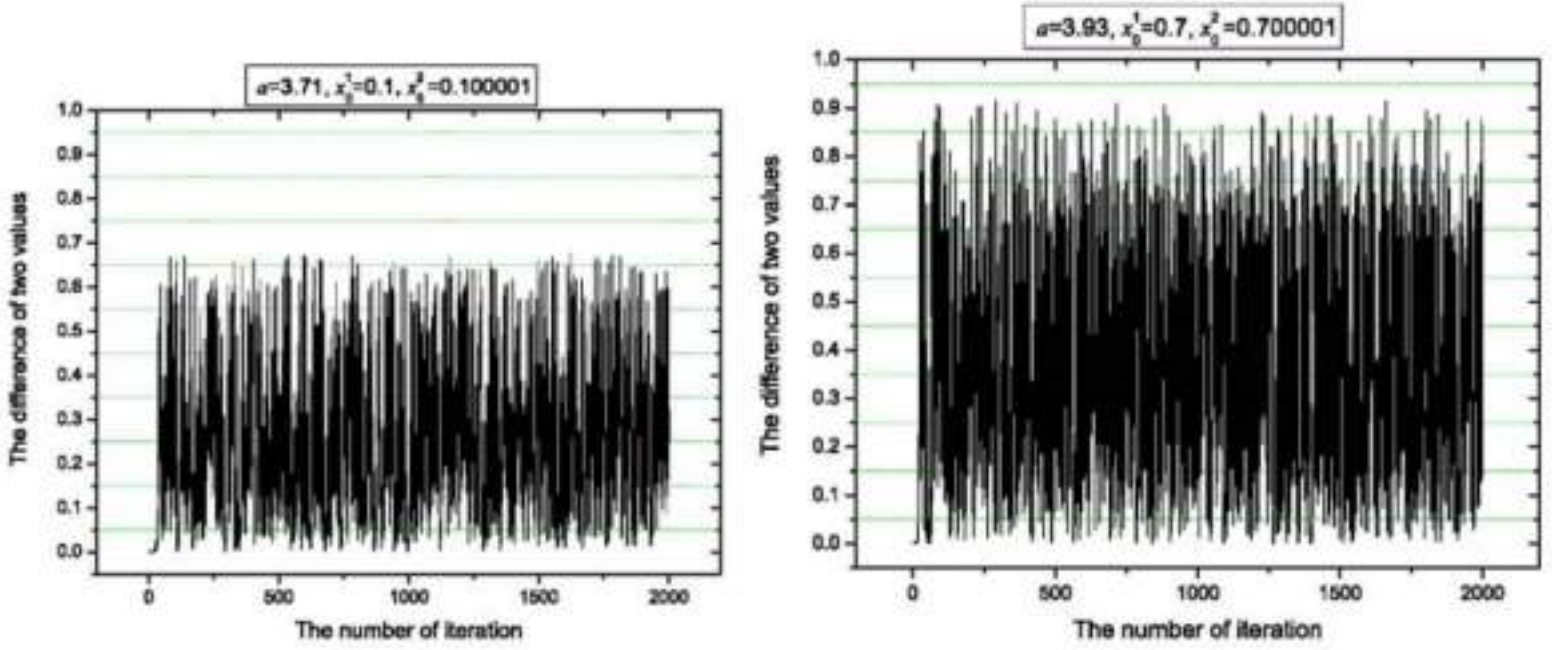
(a) (b)

Fig. 6. N = 50: (a) a = 3.71; (b) a = 3.93.



(a) (b)

Fig. 7. $N = 2000$: (a) $a = 3.71$; (b) $a = 3.93$.



(a) (b)

Fig. 7. $N = 2000$: (a) $a = 3.71$; (b) $a = 3.93$.

Iterational Relation

The keys sensitivity corresponds to the number of iteration. The logistic map has the sensitivity to

hinges in the control parameters a. Small changes

of keys produce large variations by iterating the logistic

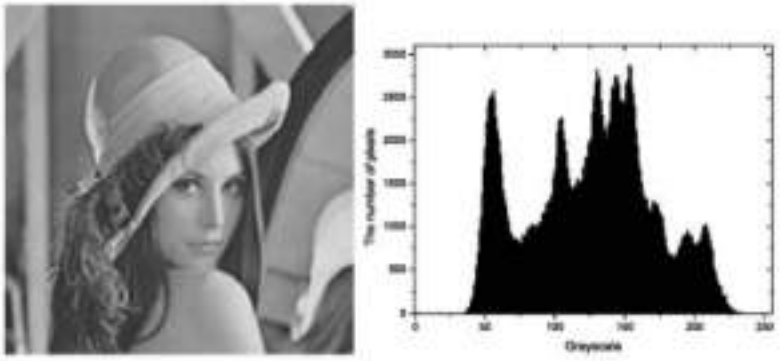
map. Therefore, cipher-breaking becomes difficult by

increasing the number of iteration. However this becomes a problem if the iterations are small and the iterations become predictable therefore they produce undesirable keys.

We take 512 x 512 size 8 bits Lena's image as an

example, where a = 3.99, X(o) = 0.11, the number of the

initial iteration of the logistic map is set to 300.



(a) (b)

Image observation

The original diagram is

shown in Fig. 8. The encrypted image of Lena and its

histogram are shown in Fig. 9.From Fig. 9 .The gray-scale distribution is of good quality and balance.

property, which is secure against known plain text

attack.

Experiment

This experiment is conducted with Authentic AMD Dual-Core

CPU at 2.712 Ghz, 2 GB RAM and MS-windows OS.

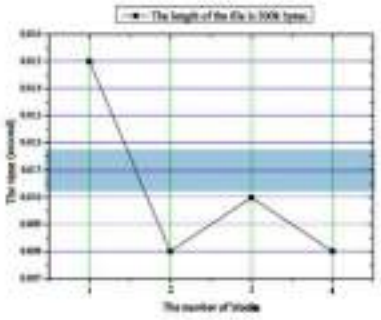
Fig. 10 the horizontal axis shows the number of

blocks, and the vertical tile shows the computational time for the different memory lengths: 200k and 1000k

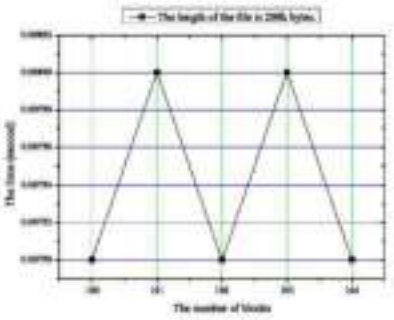
bytes. In a multi core processor 2 threads correspond to 2 cores . The results show that

in large memory tasks if the number of threads could be divisible on even blocks the performance improvement is efficient.

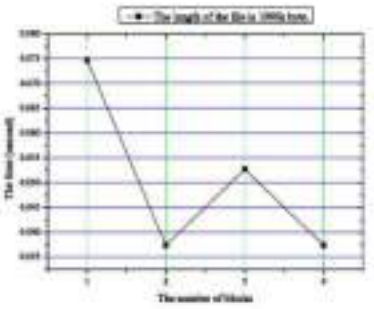
Diagram proof



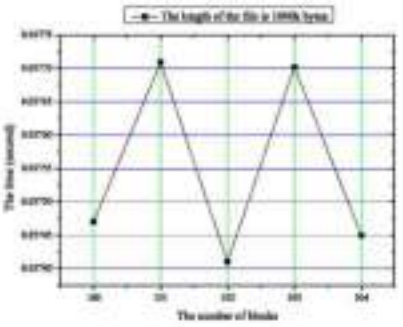
(a)



(b)



(c)



(d)

Time difference

The time varies tremendously when the number of blocks is close to the number of real cores. However, in contrast the time varies slightly when the number of blocks is considerable. If , the difference of the time approximates to 0.002s when the number of blocks for 200k bytes in the length of the file is equal to 2 and 3, respectively (see Fig. 10(a)). The time gap enlarges of the time approximates to 0.0 15s when the number of blocks for 1000k bytes in the length of the file is equal to 2 and 3, respectively (see Fig. 10(c)).

Conclusion

This part of the research proved that parallelization methods greatly improve the efficiency of the encryption algorithms used in image gray-scale encryption.

Conclusion

This part of the research proved that parallelization methods greatly improve the efficiency of the encryption algorithms used in image gray-scale encryption.

Parallel KJ Encryption algorithm

The KJ encryption algorithm is a block encryption algorithm developed by Kocarev and Jakimoski and published in 2001 based on logistic map that operates on 64-bit data blocks with a 128-bit encryption.

An input plaintext block is partitioned into eight sub-blocks, each one consists of 8 bits. The cipher consists of r rounds of identical transformations applied in a sequence to the plaintext block. Encryption transformation is given with:

$$xi\ ,2 = x\ x - 1,1\ f\ 0\ ,$$

$$xi\ ,3 = x\ x - 1,2\ f\ 1\ ,\ (1)\ \ldots$$

$$xi\ ,0 = x\ x - 1,7\ f\ 6\ ,$$

$$xi\ ,1 = x\ x - 1,0\ f\ 7\ ,$$

where: $i=1,\ldots,r.$

The functions f1,...,f7 have the following form:

$$2{:}f\ j$$

$$= f\ [\ xi- 1,1...xi- 1,\ j$$

$$zi- 1,\ j\ ]$$

where: j=1,...,7 and f: M → M, M = {0,255}, is a map derived from a chaotic map. f 0 = z i,0 and z i,0,..., z i,7 are the eight bytes of the sub-key z i which controls the i-th round. The output block becomes input in the next round, except with the last round.

The length of the cipher block is 64 bits. Each round i is controlled by one 8-byte subkey z i. There are r subkeys derived from the key in a procedure for generating round subkeys. f is obtained via discretization of logistic map.

Explanation

The parallelization process of the KJ encryption algorithm consists of the following stages:

-Carrying out the data dependence analysis of a sequential source code in order to detect parallelized loops,

-Selecting parallelization methods based on source code transformations,

- onstructing parallel forms of for loops in

accordance with the OpenMP standard.

There are the following basic types of the data

dependences that occur in "for" loops [16], [17]:

- a Data flow dependancy writebefore read is required for parallel computing
- a Data Anti-dependence and readbefore write dependancy should not be violated in parallezational

computing

- an Output Dependence indicates a write-before

write ordering.

Source Code

```
for(l=0;l< PARALLELITY;l++) {
copy(tmplain, src[8*l+8]);
for (i=1;i<=rounds;i++) {
generatekey(newkey,key,i,sbox,rounds);
dst[8*l+2]=tmpplain[1]^newkey[0];
dst[8*l+3]=tmpplain[2]^sbox[newkey[1]
^tmpplain[1]];
dst[8*l+4]=tmpplain[3]^sbox[newkey[2]
^tmpplain[1]^tmpplain[2]];
```

Source Code

```
dst[8*l+5]=tmpplain[4]^sbox[newkey[3]
^tmpplain[1]^tmpplain[2]^tmpplain[3]];
dst[8*l+6]=tmpplain[5]^sbox[newkey[4]
^tmpplain[1]^tmpplain[2]^tmpplain[3]
^tmpplain[4]];
dst[8*l+7]=tmpplain[6]^sbox[newkey[5]
^tmpplain[1]^tmpplain[2]^tmpplain[3]
^tmpplain[4]^tmpplain[5]];
```

Source Code Continuation

```
st[8*l]=tmpplain[7]^sbox[newkey[6]
^tmpplain[1]^tmpplain[2]^tmpplain[3]
^tmpplain[4]^tmpplain[5]^tmpplain[6]];
dst[8*l+1]=tmpplain[0]^sbox[newkey[7]
^tmpplain[1]^tmpplain[2]^tmpplain[3]
^tmpplain[4]^tmpplain[5]^tmpplain[6]
^tmpplain[7]];
```

Source Code Continuation

```
^tmpplain[7]];

copy(tmplain, dst[8*l+8]);

}

}.

for(l=0;l< PARALLELITY;l++) {

copy(tmpcipher,src[8*l+8]);

for (i=rounds;i>=1;i--) {

generatekey(newkey,key,i,sbox,rounds);
```

Source Code Continuation

```
dst[8*l+1]=tmpcipher[2]^newkey[0];

dst[8*l+2]=tmpcipher[3]^sbox[newkey[1]

^dst[1]];

dst[8*l+3]=tmpcipher[4]^sbox[newkey[2]

^dst[1] ^dst[2]];

dst[8*l+4]=tmpcipher[5]^sbox[newkey[3]

^dst[1] ^dst[2]^dst[3]];
```

Source Code Continuation

```
dst[8*l+5]=tmpcipher[6]^sbox[newkey[4]

^dst[1] ^dst[2]^dst[3]^dst[4]];

dst[8*l+6]=tmpcipher[7]^sbox[newkey[5]

^dst[1] ^dst[2]^dst[3]^dst[4]^dst[5]];
```

```
dst[8*l+7]=tmpcipher[0]^sbox[newkey[6]
^dst[1]^dst[2]^dst[3]^dst[4]^dst[5]^dst[6]];
dst[8*l]=tmpcipher[1]^sbox[newkey[7]^dst[1]
^dst[2]^dst[3]^dst[4]^dst[5]^dst[6]^dst[7]];
copy(tmpcipher,dst[8*l+8]);}
}.
```

Code explanation

This analysis is valid for the two loops.

The actual parallelization process of the first loop

consists of the three following stages:

- Filing the body of he loop with generatekey()(otherwise, we cannot apply a data dependence analysis)
- Required variable privatizarion (l, i, ii, k, newkey, sbox, tmpplain) using OpenMP (based on the results of data dependence analysis);

- Adding OpenMP library directive and clauses (#pragma omp parallel for private() shared()).

The steps above result in the following parallel form of loop in accordance with the OpenMP standard:

```
#pragma omp parallel for private(l, i, ii,
k, newkey, sbox, tmpplain)
for(l=0;l< PARALLELITY;l++) {
…
}.
```

The second loop was parallelized in the same way as the first one.

Experimental Results:

In order to study the efficiency of the presented KJ parallel code we used a computer with two Quad-Core Intel® Xeon Processors 5300 Series - 1,60 GHz and the Intel® C++ Compiler ver. 12.1 (that supports the OpenMP 3.1). The results received for a 20 megabytes input file using two, four and eight cores versus the only one are shown in Table 1.

Table 1

Table1. Speed-ups of the parallel KJ encryption algorithm in ECB mode of operation

Number of processors	Number of threads	Speed-up		
		Encryption	Decryption	Total
2	2	1.92	1.99	1.45
4	4	3.50	3.70	1.90
8	8	6.00	6.40	2.30

The total running time of the KJ algorithm consists of the following operations:
Reading from input file,
generation of subkeys,
encryption of data,
decryption of data,
writing the encrypted and decrypted data in output file
Thus the total speed-up of the KJ parallel algorithm depends heavily on the four factors:
the degree of parallelization of the loop included in the kj_enc() and kj_dec function,

and the method of reading data from an input file,
the method of writing data to an output file.

The experiments have shown that the application of the parallel KJ encryption algorithm for multiprocessor and multi-core computers would considerably boost the time of the data encryption and decryption. We believe that the improvements received for these operations are satisfactory. Moreover, the parallel KJ encryption algorithm can be also helpful for hardware implementations such as GPU implementations.

Conclusions

The results conclude that the loops included int the

kj_enc() and the kj_dec() functions can sped up be using parallezation

By using block method by writing in an input and output file .The following C language functions and block sizes were applied:

fread() function and 8192-bytes block for data

reading,

fwrite() function and 128-bytes block for data writing.

Using the fwrite() function is especially important; choosing, for example, the fprintf() function we got much longer time of executing our tasks.

Sources:

Dariusz Burak,Parallelization of the Block Encryption Algorithm Based on

Logistic Map,PRZEGLĄD ELEKTROTECHNICZNY (Electrical Review), ISSN 0033-2097, R. 88 NR 10b/2012

Liu, J., Hongli Zhang, Song, D., Guanglu Sun, Wenchong Bi, & Buza, M. K. (2013). A parallel encryption algorithm of the logistic map for multicore with OpenMP. Ifost. doi:10.1109/ifost.2013.6616857