Procedia Computer Science

Volume 51, 2015, Pages 2888–2892

ICCS 2015 International Conference On Computational Science

Parallelization of an Encryption Algorithm Based on a Spatiotemporal Chaotic System and a Chaotic Neural Network

Dariusz Burak

West Pomeranian University of Technology, Szczecin, West Pomerania, Poland
dburak@wi.zut.edu.pl

Abstract

In this paper the results of parallelizing a block cipher based on a spatiotemporal chaotic system and a chaotic neural network are presented. A data dependence analysis of loops was applied in order to parallelize the algorithm. The parallelism of the algorithm is demonstrated in accordance with the OpenMP standard. The efficiency measurements of a parallel algorithm working in standard modes of operation are shown.

Keywords: parallelization, OpenMP, encryption algorithm, chaotic neural network, coupled map lattice

1 Introduction

One of the very important functional features of cryptographic algorithms is cipher speed. This feature is significant in case of block ciphers since they usually work on large data sets. Thus even small differences of speed may cause the choice of the faster cipher by the user. Therefore, it is all-important to parallelize encryption algorithms in order to achieve faster processing using multi-core processors or multiprocessing systems. In recent years, besides classical ciphers such as AES or IDEA, alternative approaches of constructing ciphers based on application of the theory of chaotic dynamical systems has been developed. Furthermore neural networks are introduced to design encryption algorithms considering the complicated and time-varying nature of the structures. Chaotic neural networks (CNNs) are particulary suitable for data protection. Nowadays, there are many descriptions of various ciphers based on chaotic neural networks, for instance [8, 11, 13, 7, 9, 4]. The critical issue in such ciphers is program implementation.

Unlike parallel implementations of classical block ciphers, for instance AES [2], IDEA [1], there are only a few parallel implementations of block ciphers based on chaotic neural networks, for example [3]. Being seemingly a research gap it is absolutely fundamental to show real functional advantages and disadvantages of the encryption algorithm using software or hardware implementation. The main contribution of the study is developing a parallel algorithm in

 Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2015

doi:10.1016/j.procs.2015.05.453

accordance with OpenMP standard of the cipher designed by Wang Xing-Yuan and Bao Xue-Mei and presented in [12] based on transformations of a source code written in the C language representing the sequential algorithm.

2 Description of the Block Cipher Based on a Spatiotemporal Chaotic System and a Chaotic Neural Network

The encryption process consists of the following steps:

1. Splitting a 160 bit binary sequence K into sixteen groups.

   They are mapped into the sixteen initial conditions of the coupled map lattice (CML) [5] using the following rule:

   $x(0) = \frac{K_j}{2^{10}}$, $j = 1, 2, ..., 16$.

2. Dividing the plain image P into l blocks of 4 pixels.

3. Encrypt the i-th plaintext block P_i (the initial value of i is 1), and iterate the CML once to obtain $x_i(j)(j = 1, 2, ..., 16)$. Then, construct matrices P_i (the input of the 4-neuron layer), W_i (4×4 weight matrix), A_i (4×1 integral matrix), and B_i (the bias matrix).

4. To generate the ciphertext block C_i the following operations are applied to the plaintext block P_i:

   $Y_{1,i} = W_i P_i + B_i$, ($Y_{1,i}$ is a 4×1 matrix).

5. Perform the following preprocessing operation:

   $P_i = P_i \oplus Y_{3,i-1}$, ($Y_{3,i-1}$ is a 4×1 matrix).

   For the first plaintext block, skip this operation. Then, go to Step 3.

   The decryption process is the reverse of the encryption one. More detailed description of cipher designed by Xing-Yuan and Xue-Mei is given in [12].

3 Parallelization Process of Encryption Algorithm

Given that the proposed algorithm can work in block manner it is necessary to prepare a C source code representing the sequential encryption algorithm working in Electronic Codebook (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), Output Feedback (OFB) and Counter (CTR) modes of operation. The source code of the encryption algorithm in the essential ECB mode contains twenty eight for loops. Some of these loops are time-consuming. Thus their parallelization is critical for reducing the total time of the parallel algorithm execution.

In order to find dependencies in program a research tool for analyzing array data dependencies called Petit was applied [6].

The process of the encryption algorithm parallelization can be divided into the following stages: carrying out the dependence analysis of a sequential source code in order to detect parallelizable loops and non-parallelizable loops, selecting parallelization methods based on source code transformations and constructing parallel forms of program loops in accordance with the OpenMP standard.

To find the most time-consuming loops of the algorithm, it was carried out experiments for an about 4 megabytes input file.

It appeared that the algorithm has two computational bottlenecks: the first is enclosed in the function *xing_enc*() and the second is enclosed in the function *xing_dec*(). The *xing_enc*() function enables enciphering of any number of data blocks and the *xing_dec*() one does the same for deciphering process (analogically to similar functions of the classic block ciphers like DES- *des_enc*(), *des_dec*() presented in [10]). Thus the parallelization of for loops included in these functions is a crucial for parallelization process.

Taking into account of both functions only the first one is examined. Subsequently this analysis is valid in the case of the second one. The body of the *xing_enc*() function is as follows:

```
void xing_enc(xing_context *ctx,UINT8 *input,UINT8 *output,int input_length){
for (int i = 0; i<NBLOCKS; i++) {
    Encryption(ctx, input, output);
    input+= BLOCKSIZE;
    output+= BLOCKSIZE;
    }
}.
```

In order to apply the data dependencies analysis of the loop included in *xing_enc*() function the body of the *Encryption*() function should be put in this loop.

The actual parallelization process of the loop included in *xing_enc*() function consists of the five following stages:

- removal of the preprocessing operation, the chaotic parameter generation operation and construction of matrices P_i, W_i, A_i, and B_i from *xing_enc*() function; all calculations placed in the preprocessing operation, the chaotic parameter generation and the construction of matrices (step 3 of encryption process) have to be executed sequentially before starting the chaotic neuron layer operations;
- insertion in the beginning of the loop body the following statements: plaintext = &input[BLOCKSIZE*i]; and ciphertext= &output[BLOCKSIZE*i];
- removal from the end of the loop body the following statements: input+= BLOCKSIZE; and output+= BLOCKSIZE;
- suitable variables privatization using OpenMP (based on the results of data dependence analysis) for the loop indexing by i;
- adding appropriate OpenMP directive and clauses for the loop indexing by i.

The steps above result in the following parallel form of the loop include in *xing_enc*() function function in accordance with the OpenMP standard:

```
#pragma omp parallel private (i,ii,plaintext,ciphertext,pi,wi,bi,ai,y1i,y21i,y3i)
#pragma omp for
for (i=0; i<nblocks; i++) {
plaintext=&input[BLOCKSIZE*i];
ciphertext = &output[BLOCKSIZE*i];
for(ii=0; ii<t; ii++) {
sum_blocks(ciphertext,pi,wi,bi,S);
normalization(ciphertext,y1i,S);
xor_blocks(ciphertext,ai,y21i,S);
f(y3i,ciphertext);
}
}.
```

4 Experimental results

In order to study the efficiency of the presented encryption algorithm eight Quad- Core Intel Xeon Processors 7310 Series - 1.60 GHz and the Intel C++ Compiler (version 13.1.1 20130313 that supports the OpenMP 4.0) were used. The results received for an about 4 megabytes input file (8 bit per pixel image) using two, four, eight, sixteen and thirty-two cores versus the only one have been shown in Table 1, Table 2 and Table 3. The number of threads is equal to the number of processors.

The total running time of the presented encryption algorithm consists of the following operations: reading data from an input file, data encryption, writing encrypted data to an output file, data decryption and writing decrypted data to an output file.

Thus the total speed-up of the parallel encryption algorithm depends heavily on the following factors: the degree of parallelization of the loop included in the xing_enc() function, the degree of parallelization of the loop included in the xing_dec() function, the method of reading data from an input file and the method of writing data to an output file.

The results confirm that the loops included both in xing_enc() function and in the xing_dec() function are parallelizable with high speed-ups (see Table 1).

Number of threads	1	2	4	8	16	32
Speed-up of encryption process	1	1.95	3.78	6.02	6.22	5.98
Speed-up of decryption process	1	1.99	3.92	6.32	6.45	6.04
Speed-up of whole algorithm	1	1.48	1.91	2.32	2.52	2.29

Table 1: Speed-up of the parallel encryption algorithm in the ECB mode

The block method of reading data from an input file and writing data to an output file was used. The following C language functions and block sizes was applied: fread(), 1024-bytes blocks for data reading and fwrite(), 128-bytes blocks for data writing.

In accordance with Amdahl's Law the maximum speed-up of the encryption algorithm is limited to 4.739, because the fraction of the code that cannot be parallelized is 0.211.

The encryption algorithm was also parallelized in the following standard modes of operation (CTR, CBC and CFB). The results are presented in Table 2 and Table 3.

Number of threads	1	2	4	8	16	32
CTR mode	1	1.90	3.60	5.90	6.10	5.90
CFB mode	1	1.00	1.00	1.00	1.00	1.00
CBC mode	1	1.00	1.00	1.00	1.00	1.00

Table 2: Speed-up of encryption process in the CTR, CFB and CBC mode

Number of threads	1	2	4	8	16	32
CTR mode	1	1.95	3.60	6.00	6.15	5.95
CFB mode	1	1.95	3.60	6.00	6.15	5.95
CBC mode	1	1.95	3.60	6.00	6.15	5.95

Table 3: Speed-up of decryption process in the CTR, CFB and CBC mode

When the encryption algorithm operates in the ECB and CTR modes of operation, both the encryption and decryption processes are parallelizable and speed-ups of the whole algorithm

are similar (see details- Table 2 and Table 3).

For the CBC and CFB modes only the decryption process is parallelized so the values of speed- up are lower than for the ECB and CTR modes of operation (see- Table 2 and Table 3).

5 Conclusions

In this paper, I describe the parallelization process of the encryption algorithm designed by Wang Xing-Yuan and Bao Xue-Mei which was divided into parallelizable and non-parallelizable parts. I have shown that the time-consuming *for* loops included in the functions responsible for the encryption and decryption processes are parallelizable. The experiments have shown that the application of the parallel encryption algorithm for multiprocessor and multi-core computers would considerably boost the time of the data encryption and decryption. I believe that the speed-ups received for these operations are satisfactory. Moreover, the developed parallel encryption algorithm can be also helpful for hardware implementations or GPGPU implementation.

References

[1] Vladimir Beletskyy and Dariusz Burak. Parallelization of the idea algorithm. In *Computational Science-ICCS 2004*, pages 635–638. Springer, 2004.

[2] Wlodzimierz Bielecki and Dariusz Burak. Exploiting loop-level parallelism in the aes algorithm. *WSEAS Transactions on Computers*, 5(1):125–132, 2006.

[3] Dariusz Burak. Parallelization of encryption algorithm based on chaos system and neural networks. In *Parallel Processing and Applied Mathematics*, pages 364–373. Springer, 2014.

[4] Tariq Adnan Fadil, Shahrul Nizam Yaakob, R Badlishah Ahmad, and Abid Yahya. A chaotic neural network–based encryption algorithm for mpeg–2 encoded video signal. *International Journal of Artificial Intelligence and Soft Computing*, 3(4):360–371, 2013.

[5] Kunihiko Kaneko. Spatiotemporal intermittency in coupled map lattices. *Progress of Theoretical Physics*, 74(5):1033–1044, 1985.

[6] Wayne Kelly, Vadim Maslov, William Pugh, Evan Rosser, Tatiana Shpeisman, and David Wonnacott. New user interface for petit and other extensions. *User Guide*, 1:996, 1996.

[7] Shiguo Lian. A block cipher based on chaotic neural networks. *Neurocomputing*, 72(4):1296–1301, 2009.

[8] Shiguo Lian, Guanrong Chen, Albert Cheung, and Zhiquan Wang. A chaotic-neural-network-based encryption algorithm for jpeg2000 encoded images. In *Advances in Neural Networks-ISNN 2004*, pages 627–632. Springer, 2004.

[9] Shiguo Lian and Xi Chen. Traceable content protection based on chaos and neural networks. *Applied Soft Computing*, 11(7):4293–4301, 2011.

[10] Bruce Schneier. *Applied cryptography: protocols, algorithms, and source code in C*. john wiley & sons, 2007.

[11] Di Xiao and Xiaofeng Liao. A combined hash and encryption scheme by chaotic neural network. In *Advances in Neural Networks-ISNN 2004*, pages 633–638. Springer, 2004.

[12] Wang Xing-Yuan and Bao Xue-Mei. A novel image block cryptosystem based on a spatiotemporal chaotic system and a chaotic neural network. *Chinese Physics B*, 22(5):050508, 2013.

[13] Wenwu Yu and Jinde Cao. Cryptography based on delayed chaotic neural networks. *Physics Letters A*, 356(4):333–338, 2006.