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

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), $W_i$ ($4 \times 4$ weight matrix), $A_i$ ($4 \times 1$ 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 $4 \times 1$ matrix).

5. Perform the following preprocessing operation:
   
   $P_i = P_i \oplus Y_{3,i-1}$, ($Y_{3,i-1}$ is a $4 \times 1$ 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:

```c
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 included in $xing_{enc}()$ function in accordance with the OpenMP standard:

```c
#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);
    }
}
```

2890
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).

<table>
<thead>
<tr>
<th>Number of threads</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed-up of encryption process</td>
<td>1.95</td>
<td>3.78</td>
<td>6.02</td>
<td>6.22</td>
<td>5.98</td>
<td></td>
</tr>
<tr>
<td>Speed-up of decryption process</td>
<td>1.99</td>
<td>3.92</td>
<td>6.32</td>
<td>6.45</td>
<td>6.04</td>
<td></td>
</tr>
<tr>
<td>Speed-up of whole algorithm</td>
<td>1.48</td>
<td>1.91</td>
<td>2.32</td>
<td>2.52</td>
<td>2.29</td>
<td></td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Number of threads</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTR mode</td>
<td>1.90</td>
<td>3.60</td>
<td>5.90</td>
<td>6.10</td>
<td>5.90</td>
<td></td>
</tr>
<tr>
<td>CFB mode</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>CBC mode</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Number of threads</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTR mode</td>
<td>1.95</td>
<td>3.60</td>
<td>6.00</td>
<td>6.15</td>
<td>5.95</td>
<td></td>
</tr>
<tr>
<td>CFB mode</td>
<td>1.95</td>
<td>3.60</td>
<td>6.00</td>
<td>6.15</td>
<td>5.95</td>
<td></td>
</tr>
<tr>
<td>CBC mode</td>
<td>1.95</td>
<td>3.60</td>
<td>6.00</td>
<td>6.15</td>
<td>5.95</td>
<td></td>
</tr>
</tbody>
</table>

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