PARALLELIZATION OF RSA CRYPTOGRAPHIC ALGORITHM BASED ON CUDA TECHNOLOGIES

Lesia Mochurad, Ph.D., Ass.-Prof.; Yuriy Kryvenchuk, Ph.D., Ass.-Prof.; Svyatoslav Yatsyshyn, Dr. Sc., Prof.;
e-mail: lesia.i.mochurad@lpnu.ua
Lviv Polytechnic National University, Lviv, Ukraine

Abstract. The paper examines the efficiency of the application of CUDA technologies for the parallelization of the cryptographic algorithm with the public key. The speed of execution of several implementations of the algorithm is compared: sequential implementation on the CPU and two parallel implementations – on the CPU and GPU. A description of the public key algorithm is presented, as well as properties that allow it to be parallelized. The advantages and disadvantages of parallel implementations are analyzed. It is shown that each of them can be suitable for different scenarios. The software was developed and several numerical experiments were performed. The reliability of the obtained results of encryption and decryption is confirmed. To eliminate the influence of external factors at the time of execution the algorithm was tested ten times in a row and the average value was calculated. Acceleration coefficients for message encryption and decryption algorithms were estimated based on OpenMP and CUDA technology. The proposed approach focuses on the possibility of further optimization through the prospects of developing a multi-core architecture of computer systems and graphic processors.

Key words: The public key algorithm, Graphics processor, Efficiency indicator, OpenMP standard.

1. Introduction

The demand and need to increase the speed of software applications continues to grow as programs are developed that require more computing power. In the field of measuring technology, there is often an increase in the share of virtual measuring instruments relative to real ones. With the help of elements of the graphic library, many measuring equipment has been implemented to date. Such changes most noticeable in more developed countries, such as the United States, Germany, France, etc. It is known that NASA also very often uses similar tools on the Lab View platform, which reduces the time to build various information and measurement systems for aircraft, and a variety of auxiliary measuring equipment. As known, such systems are quite cumbersome and large and require quite powerful computer equipment. Until 2004, Moore's Law allowed the number of transistors in a processor to automatically increase software performance [1]. That is, if the processor executes more instructions per second, the software will also run faster. However, because of the physical limitations of processors and heating of components, it is impossible to continue to rely on Moore's Law to achieve faster executions of algorithms, consequently the collection of measurement data in real-time, which makes a more instrumental error in the measurement system.

One of the alternatives is parallel computing, which uses the advantages of a multi-core computer architecture [2-5]. In this model, multiprocessors communicate with each other through a shared cache contained in the hardware of the device. However, the software needs to be adapted so that it can use multiple processors to work on a single task, that is, use techniques of parallel computing to change the way code is written and executed.

One of the most interesting and popular areas of modern development is the transition from the implementation of computing on the CPU to computing on the GPU. In particular, Nvidia proposed its solution by developing the CUDA parallel computing architecture. Nvidia offers examples of the practical application of CUDA to solve general-purpose problems [6-8]. The observed increase in performance compared to the CPU in these examples is from 10 to 100 times. This approach can be applied to virtual measuring instruments.

Parallel execution of a sequential stream of instructions on the CPU has certain basic limitations and simple addition of executable blocks cannot achieve a significant increase in speed. At the same time, the GPU was created to execute parallel instructions. Most of the graphics processor, in contrast to the central, is occupied by executable units, which gives an advantage in the speed of tasks execution related to parallel data processing, namely when the same sequence of operations is applied to a large amount of data and the number of instructions exceeds the number of memory accesses. Thus, the architecture of the processor allows you to achieve greater efficiency in parallel computing. Also, an important advantage of using a graphics processor for general-purpose computing is that the computing CPU remains less busy and can be used to perform other tasks.

This paper analyzes different approaches to the parallelization of the block encryption algorithm and the decryption of the public key algorithm. This algorithm is used in a wide range of security applications [9], it can also be used to encrypt data when transmitting measurement information over long distances.

2. Problem Statement and Analysis

It is known [10], the RSA algorithm is one of the most well-known public-key cryptographic algorithms. The cryptographic stability of the algorithm is based on the computational complexity of factorization of large numbers [11]. RSA is a block algorithm that makes it suitable for parallelization.

The RSA public key consists of a module n of a certain length (product of prime numbers p and q) and an exponent e. The length of the number n is determined in bits. The associated secret key consists of the same number n, and the value d of $d*e = 1 \bmod f(n)$, where $f(n)$ – it is the Euler function. Ideally, the prime numbers generated by the random number generator used to build the module n , should not be reused. The probability that random numbers p or q will be repeated when generating keys should go to 0.

The RSA algorithm is characterized by perfect security, simple key management, it is practical in execution and understanding. However, in the RSA encryption algorithm, modular power seriously affects the performance of the algorithm and can become a bottleneck, which limits its technical application. Therefore, the optimization of the RSA encryption algorithm focuses on research related to the development of multi-core processor architecture. The use of OpenMP, Pthreads and other multithreaded technologies allows to use of more computing power, increases efficiency, and reduces data processing time.

CUDA technology makes it possible to perform general-purpose calculations on a graphics processor. Nvidia's CUDA platform enables software developers to use thousands of stream processors capable of providing parallel acceleration in a single node [12]. CUDA technology allows studying the feasibility of running a parallel program of the RSA algorithm and use JCUDA to implement parallelization.

There is no correlation or relationship between packet data in the RSA encryption process. Therefore, you can use the domain decomposition method. In this case, all data will be classified as domains. Each thread or process will process its subdomain. Due to data parallelism, the method [13] increases the computational speed of RSA. The bottleneck of the RSA encryption process is the need for big data processing. CUDA allows parallel encryption of data between packets.

3. The Goal of the Work

The goal of this work is to analyze the effectiveness of CUDA technology in parallelization of the RSA encryption and decryption algorithm and increase the speed of the measuring system, as well as to compare the results with parallelization based on OpenMP technology and its sequential version.

4. Parallelization of the RSA Algorithm

Cryptographic algorithms, in particular the RSA algorithm, usually work with large numbers that are hundreds and thousands of bits in size. The operation of modular exponentiation such numbers is associated with significant costs, which limits the operation of the algorithm at the hardware level and, as a consequence, slows down its operation. Because in practice the encryption process must take place in real-time, this shortcoming is critical. To solve this problem, experts have been developed various algorithms, one of such algorithms is the RSA algorithm [14].

The main idea of the algorithm is cyclic partial multiplication with decreasing length of intermediate results. In the first stage, the indicator is converted into binary form. Then the value received is read from right to left, that is, starting with smaller bits. The number of iterations is equal to the number of bits in the binary representation of the indicator, but the result includes only those in which the corresponding bit is equal to 1. The result is obtained according to the following formula [15]:

$$res = \prod_{i=0}^{n-1} (b^{2^i})^{a^i} \ (\mathrm{mod}\ m),$$

where n is the number of bits of the indicator, b is the basis, i is the iteration number, m is the module (absolute value). The following is the pseudocode of this algorithm:

```
result = 1
exponent = e
while exponent > 0
if (exponent mod 2 == 1):
result = (result * g) mod m
exponent = exponent >> 1
g = (g * g) mod m
return result
```

The Open MP standard is used to implement a parallel algorithm on the CPU. The parallelization process was carried out directly in the encryption/decryption units. Threads for parallel code processing and distribution of instructions between them occurred dynamically using a combination of directives #pragma omp parallel for (up to 256 threads). The previously described algorithm "Right-to-left binary method" was used to implement the modular exponentiation.

The following is the source code for the encryption feature:

```
void encrypt(int *num, int *key, int *den, unsigned int *result, size_t len) {
#pragma omp parallel for
for (int i=0; i < len; i++) {
result[i] = mod(num[i], *key, *den);
}
```

At parallelization utilizing CUDA the algorithm of interaction between the central and graphic processors constructed as follows:

Step 1: Reception by the CPU of the original message and parameters.
Step 2: Check for available graphics devices and initialize them.
Step 3: Initialize copies of host and device variables:
* dev_num, * dev_key, * dev_den, * dev_res.
Step 4: Allocate GPU space for these variables.
Step 5: Transfer the input data to the GPU.
Step 6: Calling the CUDA kernel on the GPU:
RSA <<< nblocks, nthreads >>> (dev_num, dev_key, dev_den, dev_res)
Step 7: Copy the results back to the host.
Step 8: Clear the memory.

Here, the RSA kernel is a function that is executed on a group of threads. According to the researched problem, the RSA kernel implements the calculation of the encrypted text, using the algorithm of exponentiation of "Right-to-left binary method". For all calculations on the GPU, 1 block is allocated, which contains 512 threads.

The following is the source code of the encry-ption-decryption function of the RSA algorithm:

```
_global_ void rsa(int *num, int *key, int *den, unsigned int
*result) {

int i = threadIdx.x + blockIdx.x * blockDim.x;
int temp = mod(num[i], *key, *den);
atomicExch(&result[i], temp);
},
```

where mod (num[i], *key, *den) – the function of the modular erection in the degree described above. Here num[i] – the corresponding processing unit, *key – an indicator of the degree, *den – module. The result is assigned to the temp variable and returned to the host.

To identify a specific thread, its index is calculated that is the unique identifier of each thread in the block, which executes one instruction, but with a different set of data. The data are distributed among the threads [16]. Such property as multithreading is also investigated here. The optimal number of threads should be equal to the size of the input file, but this size is variable. When the number of threads exceeds the optimal one, some threads consume only resources. In the case when the number of threads is lesser than the optimal value, the load on the thread increases. The grid 128 * 128 seems to be the optimal number able to operate with large and small files.

5. The Results of Parallelization of the RSA Algorithm

To evaluate the performance of the implemented programs, the same file was used for the encryption and decryption algorithm. Thus, the reliability of the obtained results was checked, because the decrypted file corresponds to the same original file. Similarly, the execution time was measured to make appropriate comparisons. The algorithm was performed 10 times in a row to average the results, to eliminate the influence of external factors.

To study the work and compare the efficiency of the written algorithms, numerous experiments were conducted, consisting of the following test groups:

1. Fixed message size (760 bits). Encryption and decryption algorithms are performed with keys of different sizes (from 768 bits to 8192 bits).

2. Incoming messages of variable size. The size is determined by the size of the encryption key.

Numerical experiments were performed on a PC with the following characteristics:
CPU: Intel (R) Core ™ i7-8550U 1.8GHz, No of Cores: 2
Memory: 16,00 Gb
GPU: NVIDIA GeForce GT 550m, 96 CUDA cores
System: Windows 10

Table 1-Table 4 is represented the results of experiments in groups. The results of the experiments of group 1 are given in Table 1 and Table 2, and the results of experiments of group 2 – in Table 3 and Table 4. Visualization of the obtained results is presented in Fig. 1-Fig. 4, respectively.

Table 1

The execution time of the encryption algorithm for a message size of 760 bits with a variable key size

Key size, bit	Execution time, s		
	The sequential algorithm on the CPU	The parallel algorithm on the CPU	The parallel algorithm on the GPU
768	0.112	0.92	1.06
1024	0.129	0.93	0.93
2048	0.46	1.27	0.86
3072	0.84	1.2	0.77
4096	1.48	1.56	0.92
6144	4.07	3.09	1.89
8192	5.89	3.68	1.93

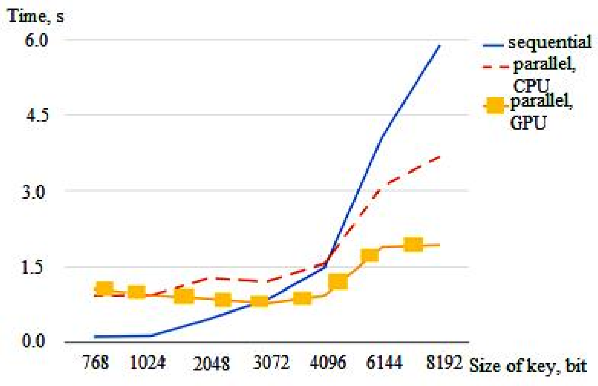

Fig. 1. Visualization of execution time

Table 2

The execution time of the decryption algorithm for a message of 760 bits with a variable key size

Key size, bit	Execution time, s		
	The sequential algorithm on the CPU	The parallel algorithm on the CPU	The parallel algorithm on the GPU
768	5.03	5.45	2.32
1024	8.82	7.78	2.74
2048	76.27	36.67	9.26
3072	252.04	75.68	22.44
4096	410.22	143.78	40.57
6144	1724.45	368.89	94.35
8192	2234.38	446.8	125.16

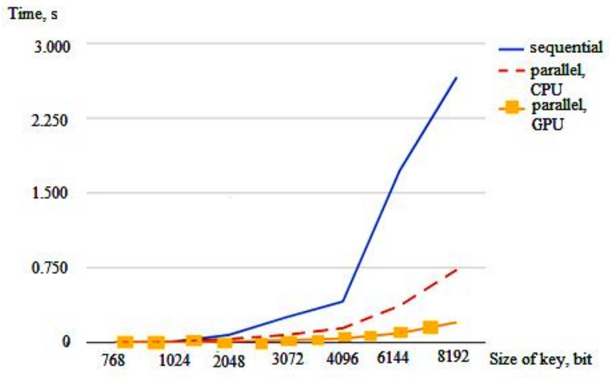

Fig. 2. Comparative graph of execution time

Table 3

The execution time of the encryption algorithm for the variable size message

Key size, bit	Execution time, s		
	The sequential algorithm on the CPU	The parallel algorithm on the CPU	The parallel algorithm on the GPU
768	0.55	0.83	1.04
1024	0.69	0.92	1.17
2048	1.05	2.6	1.44
3072	2.96	3.89	2.26
4096	6.7	6.7	4.23
6144	25.69	11.54	7.98
8192	42.9	19.24	10.82

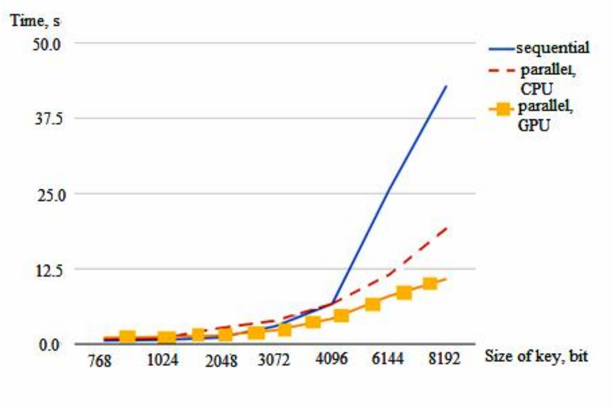

Fig. 3. Comparative graph of execution time

Table 4

The execution time of the decryption algorithm for the variable size message

Key size, bit	Execution time, s		
	The sequential algorithm on the CPU	The parallel algorithm on the CPU	The parallel algorithm on the GPU
768	4.67	5.34	2.54
1024	10.87	8.78	2.98
2048	149.67	62.89	16.45
3072	603.66	187.54	55.87
4096	1887.7	614.43	123.76
6144	9534.87	2165.34	449.52
8192	26893.05	4598.69	1489.92

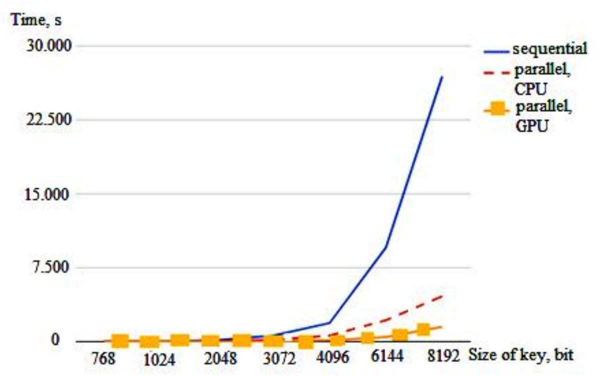

Fig. 4. Comparative graph of execution time

The results are given in Tables 1 and 3 show that the implementation of the algorithm on the GPU starts working faster than the other two implementations when the key size becomes larger than 3072 bits.

Comparing the results given in Tables 2 and 4 with the corresponding Table 1 and 3, we can conclude that the duration of decrypting the message exceeds the time required for encryption. This is due to significantly exceeding the values of the generated private keys the similar values of public keys and increases the number of calculations. For keys of any size, the results of the decryption algorithm on the GPU are better than the results on the central one. The large acceleration in speed is obtained by CUDA to parallelize the RSA algorithm on the GPU.

To assess the effectiveness of parallel implementation for 4 tables, the acceleration factor was

calculated [16]. The results of the calculation of the latter are given in Tables 5 and 6, where S_1 is the acceleration factor for multithreaded implementation on the CPU; S_2 – for realization on a graphical processor.
From Tables 5-6, we can conclude that the acceleration factor for the encryption process is small, but there exist significant advantages for decrypting.

Table 5

Coefficients of acceleration for the tables of experimental group 1

Key size, bit	Encryption		Decryption	
	S_1	S_2	S_1	S_2
768	0.121	0.105	0.922	2.168
1024	0.138	0.138	1.133	3.218
2048	0.362	0.534	2.079	8.236
3072	0.7	1.09	3.33	11.231
4096	0.948	1.608	2.853	10.111
6144	1.317	2.153	4.674	18.277
8192	1.6	3.051	5.234	18.98

Table 6

Coefficients of acceleration for the tables of experimental group 2

Key size, bit	Encryption		Decryption	
	S_1	S_2	S_1	S_2
768	0.662	0.528	0.874	1.838
1024	0.75	0.589	1.238	3.647
2048	0.403	0.729	2.379	9.098
3072	0.76	1.309	3.218	10.804
4096	1	1.583	3.072	15.252
6144	2.226	3.219	4.403	21.211
8192	2.229	3.964	5.847	22.049

6. Conclusions

To speed up the transmission/reception of measurement information between remote devices, CUDA technology has been applied for the parallelization of the RSA algorithm. The obtained results with parallelization based on OpenMP technology are compared with its sequential variant. To evaluate the efficiency of the proposed parallel algorithm, the acceleration factor was calculated: for the encryption algorithm with the key size of 8192 the mentioned factor is ~ 4, and for the decryption algorithm under the condition of GPU application this factor is estimated as 22. That is, CUDA technology could be applied to accelerate the decryption algorithm for large public keys.

In general, the developed method confirms the effectiveness of parallel computing on a graphical processor to increase the productivity of cryptographic algorithms and accelerating the virtual system.

7. Conflict of interest

The authors state that there are no financial or other conflicts regarding the work.

References

[1] H. Sutter, “The free lunch is over: A Fundamental Turn Toward Concurrency in Software”, *Dr. Dobb's Journal*, vol. 30, no. 3, p.7, 2005.

[2] M. Balandin, E. Shurina, “The Methods for Solving High-dimensional SLAE”, NSTU, pp. 28 – 35, 2000.

[3] B. Chapman, G. Jost, “Ruud van der Pas: Using OpenMP: portable shared memory parallel programming”, Sc. and Eng. Comp., Cambridge, pp. 164 – 172, 2008.

[4] L. Mochurad, N. Boyko, V.Sheketa, “Parallelization of the Process of Calculating the Optimal Route for a Strike Aircraft Flight”, Proc. of 2nd Int. Workshop on Control, pp. 63 – 75, 2020.

[5] C. Yang, C. Huang, C. Lin, “Hybrid CUDA, OpenMP, and MPI parallel programming on multicore GPU clusters”, Comp. phys. com., vol. 1, pp. 266 – 269, 2011.

[6] A. Grama, A. Gupta, G. Karypis, V. Kumar, “Introduction to Parallel Computing”, Addison Wesley, p. 856, 2003.

[7] L. Mochurad, N. Boyko, “Technologies of distributed systems and parallel computation:”, Publ. House “Bona”, 2020.

[8] R. Farber, “CUDA Application Design and Development”, Morgan Kaufmann, p.336, 2011.

[9] J. Sanders, E. Kandrot, “CUDA by Example: An Introduction to General Purpose GPU Programming”, Addison-Wesley Professional, p. 312, 2010.

[10] S. Barychev, V. Honcharov, R. Serov, “"Fundamentals of Modern Cryptography: A Textbook", RF: Hot Line, 2002.

[11] A. Metolkin, V. Kardashuk, “Studies of the Methods of enhancing the cryptographic stability”, Bull East-Ukr. University named after V. Dal, vol. 6, pp. 90-95, 2018 (in Ukr.).

[12] Official page of CUDA technologies, 2020. [Online]. Available: https://developer.nvidia.com/cuda-zone.

[13] O. Klochko, E. Kovalenko, “RSA Data encryption algorithm”, J.: Science, technics and education, vol 3., pp. 1-11, 2016.

[14] S. Prasanth, K. Jegadish, B. Partibane, “Efficient Modular Exponentiation Architectures for RSA Algorithm”, Int. J. Eng. Res. in Electronic and Com. Eng., vol. 3, no. 5, pp. 230–234, 2016.

[15] S. Saxena, B. Kapoor, “State of the Art Parallel Approaches for RSA Public Key Based Cryptosystem”, Int. J. on Comp. Sc. & Appl. (IJCSA), Vol.5, No.1, Febr. 2015.

[16] D. Chang, M. Kantardzic, M. Ouyang, “Hierarchical Clustering with CUDA/GPU”, ISCA PDCCS, pp. 7–12, 2009.

[17] L. Mochurad, N. Boyko, N. Stanasiuk, “Forecasting stock prices and accounting for stock market on multicore computers”, Int. Workshop on Conflict Man. in Glob. Inf. Networks, pp. 276–289, 2019.