AES Algorithm Adapted on GPU Using CUDA for Small Data and Large Data Volume Encryption

Tomoiagă Radu Daniel, Stratulat Mircea

Abstract: - The goal of this paper is to study the possibility of using alternative computing solution in cryptography, the use of a graphic processing unit in non graphical calculations. We tried to use the graphic processing unit as a cryptographic coprocessor in order to obtain more computing power and better runtime for AES. In this paper we present an implementation of AES on NVIDIA GPU using CUDA. The results of our tests show that the CUDA implementation can offer speedups of almost 40 times in comparison to the CPU. The tests are conducted in two directions: running the tests on small amount of data that is located in memory and a big amount of data that are stored in files on hard drives and as news, the access time for the information on the hard disk is added to the encrypting time.

Key-Words: - Benchmark, AES, CUDA, GPU, Cryptography,

I. Introduction and Related Work

Graphic Processing Units are being used nowadays not only for 3D rendering, games engines and film encoding/decoding, but also for a vast area of applications. One of these is Cryptography. In the field of Cryptography from the moment, when compatible DX10 Graphic Processing Units (e.g. G80) offered support for integers and byte operations, GPU had become an eligible competitor for a cryptographic coprocessor.

It is much easier to develop benchmark applications that are oriented on certain goals by reaching isolated points. The main advantage of benchmarks is the simplicity of the obtained and presented results: they are easier to understand. Any component (software or hardware), is acquired based on its characteristics. Characteristics are obtained and presented after some benchmarks (CPU frequency, memory frequency, hard disk capacity/ read/write speed, etc.).

In order to be up to date with the technological development, benchmarks have to be continuously upgraded, so that the results obtained are accurate [7]. The trend is to sell products mentioning the results obtained for certain tests advantaging the respective product. Moreover, producers design products to run better on certain sets of tests so that the results exceed those of competing products putting them in front. There are some special cases where benchmarks are designed so that they run in the shortest time possible on certain products, in order to advantage them on the market [9].

Generally a benchmark executes a finite number of instructions. The system that finalizes that instructions set within the shortest period is placed in the top of the test. In antithesis, we [9] can present benchmark models that do not require the completing of the test within a period of time, but are directed on calculating the work volume. In [17], it is presented a benchmark called HINT[3][7], which does not belong to any category described above.

In the recent years, because of the slow processors evolution, big computing power application developers oriented towards other type of processors. Graphic Processor Units have been taken in consideration. Video Graphic Card vendors designed more powerful graphical video cards and gave software developers the chance to write their own programs to use co processing on CPU and GPU.

Yeom analyzed the improved performances using DirectX and OpenGL [12], and after finalizing his research he concluded that an Intel Core 2 Quad (QX6850) processor is able of speeds up to 96 GFLOP, while a NVIDIA GeForce 8800GTX is capable of 330 GFLOP. In his tests AES has a 4.5 Gbps on this GPU.

Kipper writes about implementing AES on GPU and concludes that the algorithm is 14.5 faster than on a classic processor [4].

Luken speaks about encrypting with AES and DES using GPU hardware acceleration [5]. The tests were done on data volume up to 100 Mb, and the performances were as following: AES is 3.75 faster on GPU than on CPU.

Manavski tested CUDA compatibility in hardware acceleration for AES on NVIDIA graphic cards [6]. His best result was on AES 128, for an 8 MB input file, the performance being of 8.28 Gbps. The GPU algorithm was 19, 60 times faster than the CPU algorithm.

II. AES Algorithm

At the beginning of AES the message (plaintext or cipher text), being a 128 bit block is segmented in 16 Bytes. The input block has 16 units, each having 8 bit and can be represented as InBl=$m_0,m_1,...,m_{15}$[11].

Rijndael based his theory on Galois Field, meaning that certain operations are defined at byte level and bytes represent elements in a finite field GF(2^8). As every finite field GF(2^8) representation are isomorphic, classic polynomial representation can be chosen that has a positive impact on implementation complexity. [11]

All byte values are represented as eight bit concatenation in this order $\{b_7,b_6,b_5,b_4,b_3,b_2,b_1,b_0\}$. The polynomial representation for a byte is :

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0x^0 = \sum_{i=0}^{7} b_i x^i \text{ [15]}$$

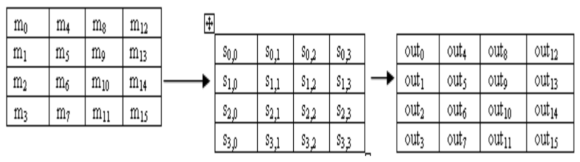
The internal structure of an input bloc is a 4x4 matrix:

$$InBl = \begin{pmatrix} m_0 m_4 m_8 m_{12} \\ m_1 m_5 m_9 m_{13} \\ m_2 m_6 m_{10} m_{14} \\ m_3 m_7 m_{11} m_{15} \end{pmatrix} \text{[11]}$$

Internaly, AES operations are done on a vector called **State**. This is composed of four rows of bytes. The State is similar to a matrix and every element has two indices:

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

Elements from the input block (m_0, m_1,..., m_{15}) are copied in the State. Encrypting or decrypting operations are applied on the State, and in the final step are copied in an exit matrix [15]:



AES algorithm is based on a number of iterations that apply transformations, called **rounds**:

Round (State, RoundKey) [11],

where RoundKey is the round corresponding key, obtained from the key provided at the beginning of the algorithm.

The State, in the first round, will get the input values from InBL, and for the final round it will output the message (encrypted or decrypted) A round (except the final one) contains four transformations :

```
Round (State, RoundKey)
{
SubBytes(State)
ShiftRows(State)
MixColumns(State)
AddRoundKey(State, RoundKey)
}[11]
```

The final Round differs from a normal round through missing the MixColumns transformation. For decription, the inverse function is used: Round^{-1}(State, RoundKey).

Internal functions are applied on a finit field. This is done using modulo f(x) polynomial, where f(x) is an irreducible polynomial:

$$f(x)= x^8+x^4+x^3+x+1. \text{ [11]}$$

and any modulo f(x) polynomial will be maximum a 8 rank polynomial and is represented as follows:

$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0$ [11],

where $b_7b_6b_5b_4b_3b_2b_1b_0$ frotm a byte or an integer on 8 bit.

If we have two hex numbers „57" and „83", then „57" $\oplus$ „83"= „D4", in other words 01010111 $\oplus$ 10000011 =11010100 and in polynomial form $(x^6+x^4+x^2+x+1) \oplus (x^7+x+1)= x^7+x^6+x^4+x^2$[15]. In this example we have modulo 2 adding (or XOR) where 1 $\oplus$ 1=0, 1 $\oplus$ 0=1 and 0 $\oplus$ 0=0.

When multiplying modulo f(x) for the given example „57" and „83" we will have „57" $\bullet$ „83"= „C1" because

$(x^6+x^4+x^2+x+1) \bullet (x^7+x+1)= x^{13}+x^{11}+x^9+x^8+x^7+ x^7 + x^5 + x^3 + x^2 + x + x^6 + x^4 + x^2 +x+ 1 = x^{13}+x^{11}+x^9+x^8 + x^6 + x^5 + x^4 + x^3 + 1$ modulo $f(x)= x^7 + x^6 + 1$ [15].

If $b(x)a(x) + f(x)c(x)=1$ and $a(x) \bullet b(x) \bmod f(x) =1$, then $b^{-1}(x)= a(x) \bmod f(x)$ and so $a(x) \bullet (b(x)+c(x))=a(x) \bullet b(x)+a(x) \bullet c(x)$ [15].

These operations together with byte operations like xtime() are used internally by AES. As it can be seen these operations are not complex and computing power consuming, as the operations used in asymmetric algorithms.

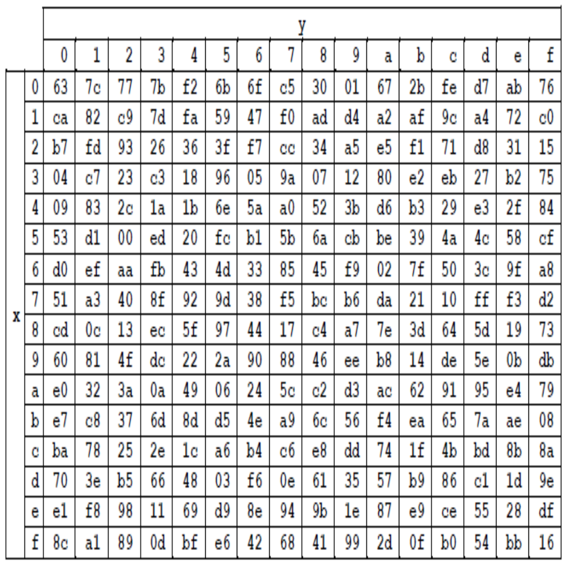
When encrypting, internally transformations as **SubBytes(),ShiftRows(), MixColumns(),** and **AddRound Key()** are used.

SubBytes() is a substitution that operates on every byte using a substitution table (S-box). This table (fig. 1) is built using two transformations:

- ✓ $b(x)a(x) + f(x)c(x)=1$
- ✓ $b_i' = b_i \oplus b_{(i+5)\bmod 8} \oplus b_{(i+6)\bmod 8} \oplus b_{(i+7)\bmod 8} \oplus c_i$ where b_i and c_i is the ith bit from the byte b or c.

In matrix form it can be written as shown next:

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 10001111 \\ 11000111 \\ 11100011 \\ 11110001 \\ 11111000 \\ 01111100 \\ 00111110 \\ 00011111 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \text{ [15]}$$



		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
x	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Fig. 1 S-BOX Subtitution valus in Hexazecimal [15]

In ShiftRows() a byte is shifted cyclic in the last three rows of the State (fig 2). The effect is that a byte is moved in “inferior” positions of the row. The first row is not shifted.

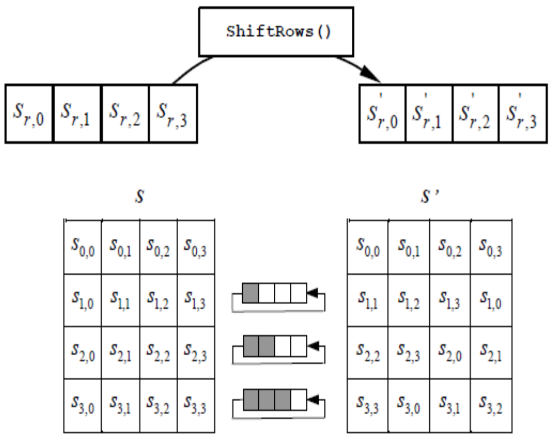

ShiftRows()
$s_{r,0}$ $s_{r,1}$ $s_{r,2}$ $s_{r,3}$
$s'_{r,0}$ $s'_{r,1}$ $s'_{r,2}$ $s'_{r,3}$
s
s'

Fig. 2 ShiftRows(). [15]

A Rijandael computational complexity analysis was done by Fabrizio Graneli and Giulia Boato in [2].

In MixColumns() the **State** is operated column by column. Every column is treated as a polynomial with four elements. This is multiplied modulo x^4+1 with a polynomial a(x), where a(x)= $\{03\}x^3+\{01\}x^2+\{01\}x+\{02\}$ [15]. In other words: $s'(x)=a(x)\otimes s(x)$

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

After these operations the four bytes from the column are replaced by these:

$$s'_{0,c} = (\{02\}\bullet s_{0,c}) \oplus (\{03\}\bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$

$$s'_{1,c} = s_{0,c} \oplus (\{02\}\bullet s_{1,c}) \oplus (\{03\}\bullet s_{2,c}) \oplus s_{3,c}$$

$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\}\bullet s_{2,c}) \oplus (\{03\}\bullet s_{3,c})$$

$$s'_{3,c} = (\{03\}\bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\}\bullet s_{3,c}).$$

Table 1. AES Computational complexity. Operations[2]

In [2] the authors compare Rijndael, Camelia and Shacal-2. They conclude that “*Rijndael is very good and can be used as reference for benchmarks*”. In table 1 values regarding AES are presented according to the tests done by the authors.

In AddRoundKey() a new round key is added (RoundKey) using XOR() as in the Fig. 4.

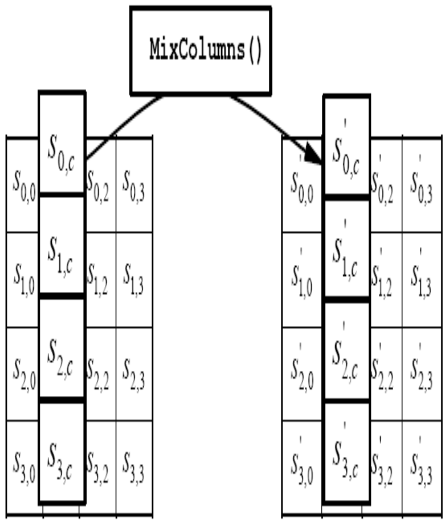


Fig. 3 MixColumns().[15]

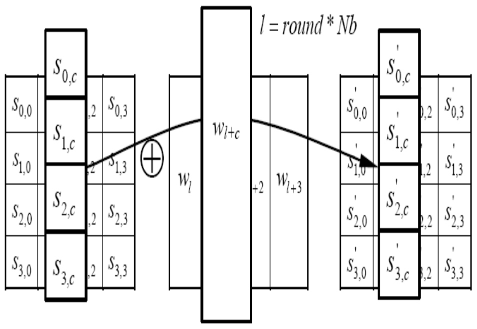


Fig. 4 AddRoundKey().[15]

III. G80- Architecture and Specifications

NVIDIA 8800 GT is based on G80 Core, which is the first 65nm NVIDIA Kernel. This contains 754 millions transistors and 128 processors. 8800 GT operates at 600 MHz, having 512 MB of memory at 900 MHz connected at 256 Bit Bus and each processor has a frequency of 1,5 GHz.

G80 has 16 Multiprocessors that are contained on a single chip. Every Multiprocessor contains 8 ALU, which are controlled by one SIMD(Single Instruction Multiple Data). The Instruction Unit commands a single Instruction from ALU at every four clock cycles [1]. This fact offers a 32 SIMD capacity for every multiprocessor. Every multiprocessor has 32 bit registers, shared memory and constant cache. All other type of memory is located in global memory [1].

	Operations			
	AND	OR	Shift(bytes)	Adding 32 bit
AES General	5836	4254	1336	0
Key expansion	1536	1536	846	144
Encryption	4912	3624	1188	0
Decryption	14896	11112	3654	0

Operation	Algorithm (key size)		
	128	192	256
AND	7236	8784	10334
OR	5418	6536	7667

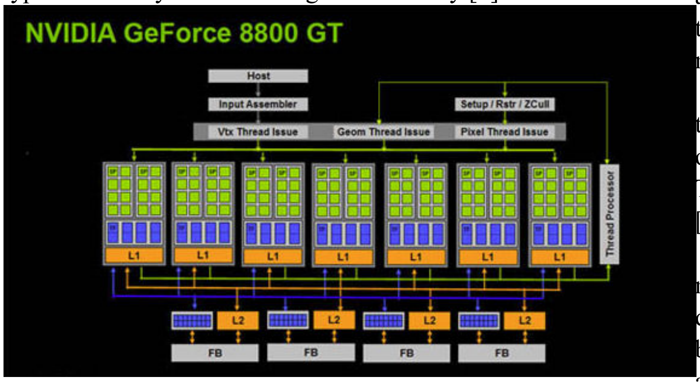


Fig.. 5. NVIDIA 8800 GT. Architecture [16]

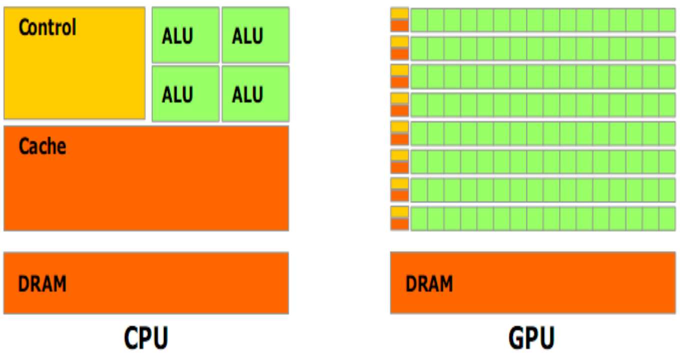


Fig. 6. Processing data oriented GPU [14]

IV. CUDA (Compute Unified Device Architecture)

CUDA was introduced in 2006 as new parallel computation architecture with a new set of Instructions and a new programming parallel model. CUDA offers a software environment which allows the programmers to use C as a high level programming language for the GPU. When using CUDA, the GPU is seen as computational device capable of executing a high number of threads in parallel. The GPU operates as a coprocessor for the CPU. If a part of a program that executes several times independently can be isolated this can be rewritten to be executed on GPU as more independent threads.

When a kernel is invoked it will run on a grid. The number of block and threads on a grid can be configured.

Under CUDA, threads can access different memory locations. Every thread has a private memory. Every Block has a shared memory which is accessible for every thread within the block. All threads form all blocks can access the global memory.

A kernel can be executed by many blocks of threads, so that the maximum number of threads equals the maximum number of thread for each block multiplied by the number of blocks. These blocks are organized in one or two dimensional grids [14].

All threads within a block will be executed on one multiprocessor. This allows for threads within a block to share data using the shared memory. Communications between blocks is not permitted as there is no synchronization solution available [1]

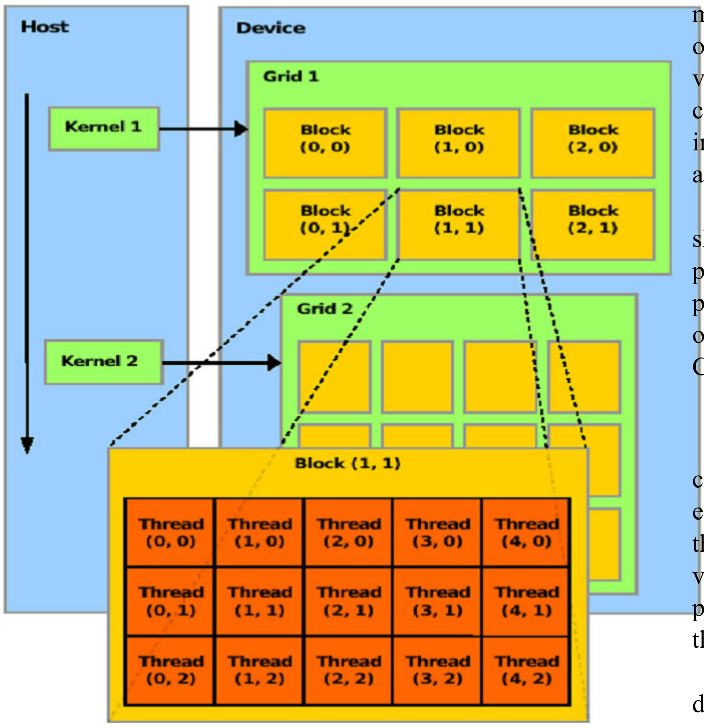


Fig. 7. GPU threads [14]

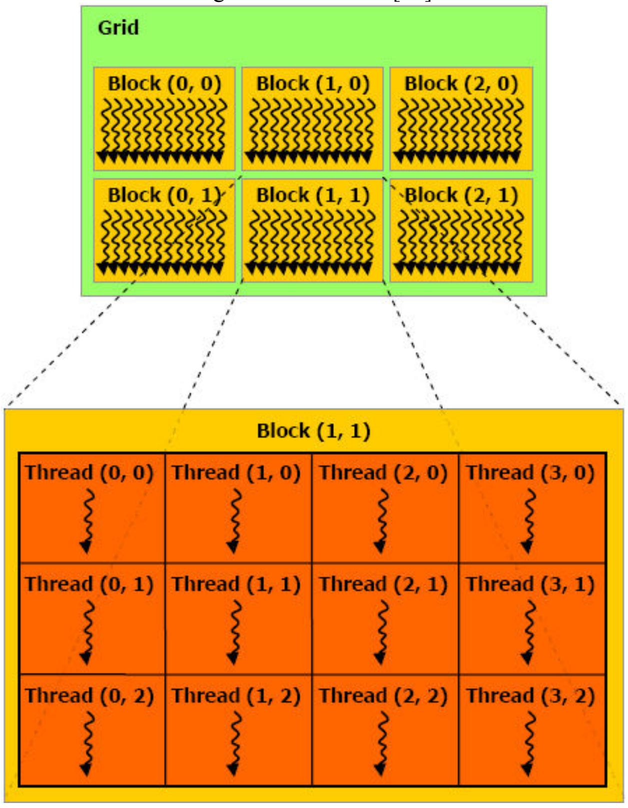


Fig. 8. Grid with bloks and threads [14]

V. AES Implementation and results

When adapting AES on GPU, and more specifically CUDA we had to rethink the optimization done for the CPU, as it would not normal work also for GPU. For instance the CPU optimization would rely on lookup tables which are stored in memory. Our expectations would be that the global memory of the GPU is much slower to access than to compute the values. In tests done in [15] the authors concluded that in the case of high number operations the GPU operates faster than in the case when lookup tables are used and memory is accessed.

The GPU is also designed to run tasks in parallel, so AES should be adapted to run the most part of the algorithm in parallel. The only two modes that permit AES to be run in parallel are ECB and CTR. CBC mode can be parallelized only for decryption. For the tests done in this paper we chose CTR.

$K_i := E(K, Nonce \| i), for, i = 1,..k$ and $C_i := P_i \oplus K_i$ (1)

CTR uses a simple method to generate the keys. It concatenates a nonce with the current counter value and encrypts it in a single block. The nonce must be smaller than the block size, as it must be concatenated with the counter value. The main advantage of CTR is that it can be used for parallelizing high speed applications. Another advantage is that for decryption the same code can be used.

The key expansion is done on CPU and the encryption is done on the GPU. The key expansion is done serially and it will slow the GPU down, so that is why we choose to do it on the CPU.

To try an optimization we will set all the threads to use the global memory. In doing this we group all the access to the memory, all data will coalesce to permit the more rapidly memory read/writes. [4]. Access to global memory is done in the initial phase, before processing data. The data is moved in shared memory where it can be accessed faster. If every thread loads data in the shared memory form global memory which it will possibly not use then we need a synchronization step before the actual load in the shared memory. This synchronization step is necessary and when writing back in the global memory [4].

Another optimization in implementing AES in C for CUDA is in that of using lookup tables, similar to the CPU optimization. The size of these tables is 16x16 bytes. These tables, having constant values, can be loaded in the shared memory of the GPU in order to be accessed by threads faster. For small sizes it is expected that the CPU will encrypt faster than the GPU [4]. Although in the tests done in [15] the authors proved that for large data better performances are obtained if the values are calculated instead of looking them up in tables stored in memory.

An alternative would be the use of constant memory for storing the SBOX and the round keys. The advantage in this solution is that the values can be appended from the design phase and can be accessed even by the CPU [5].

In [4] authors designed a parallel implementation of AES on the CPU, but we must mention that a CPU has not the possibility of running a high number of threads as a GPU does. This is why a parallel CPU implementation will not have the same result as GPU.

In developing AES in C for CUDA we follow two directions: in the first we will try to use the GPUs computing power for calculating all AES operations and the second one we use lookup tables instead of calculating the values. Because we use CUDA we could easy chose to implement AES with lookup tables as GeForce 8800 GT/G80, is a scalar processor and it is not necessary cu combine instructions in vector operations to obtain maximum computing power. In the same time G80s ability to execute 32 bit XOR operations offers a performance boost to this solution.

When writing the application we were designing it similar to the ones in [9] an [10].

In the beginning we generate a random value that will be used for encrypting.

Input sizes are the same to the ones used in [9] an [10]. First the random value is generated that will be used in the encryption. This value has the size of 128 bit, the exact AES block size.

In this paper we test AES algorithm that encrypts data stored in the GPU memory and by doing this we simulate „on the fly” data encryption. The algorithms are run repeatedly recursively: *buffer = algoritm_encrypt(buffer+random value)*. The chosen iterations number is 1.000.000 like the tests we did in [9] an [10].

The runtime is measured and divided by the iteration number, obtaining the average algorithm runtime for one step(iteration).

The obtained results were compared with the results that we obtained using the algorithm developed by [13]. In table 1 are three values presented. The first represents the AES algorithm implementation done in [13], the second one represents the time used by the GPU to encrypt data using AES on GPU based on SBOX and lookup tables and in the third column is the value that we obtained by running AES on GPU that calculates all values necessary without the use of SBOX and lookup tables.

The algorithm that uses SBOX tables got the result 0,00116233 ms. This value does not contain the time necessary to obtain the random value. It contains only the data stored in the shared memory that are read/ written encrypted and re encrypted for 1.000.000 times. The time obtained is not affected by the time necessary to bring data form the CPU because data is stored and read directly from the GPU memory. Taking in consideration that all threads need to access the lookup tables, these were stored in the shared memory so that they would be accessible for every thread

In the case of the algorithm that does not use lookup tables the time we obtained was 0,00121234 ms and is greater than the one obtained using lookup tables. As in the case of the previous algorithm, data was stored form the shared memory of the GPU and was read/ written in the same memory to be accessible for the next iteration.

Comparing the two results obtained for the data stored in memory we can conclude that, as in the case of AES optimization for the CPU, the fastest AES is the one that is using lookup tables, tables that are stored in the GPU memory.

Comparing the best time we obtained on the CPU using the test done in [9] an [10] that were run on the same platform as the GPU tests we can say that the time obtained by the GPU is better than the one obtained by the CPU. On the CPU the best time was obtained by running AES on Java and had the following value 0,156324 ms. The performance ratio in this case was aprox. 134. In the case of AES without lookup tables the performance ratio was aprox. 128.

Analyzing the three results we obtained in the tests done in this paper we can say that one of our implementation is faster than the one described in [13] and the other one is slower. Results are not so different and performance ratio between these algorithms can be observed in Fig. 5.

Using the results obtained for this platform in [9] an [10] and analyzing Fig. 6 we can conclude that AES implementation using CTR in order to benefit of parallelization is much more faster than the standard one that uses standard libraries that are offered by the programming language (VB,C# and Java) when implementing AES for CPU. Values presented in Fig. 6 are obtained taken from [10]. Based on our results we concluded that CUDA AES is up to 134 times faster than AES CPU(JAVA). Of course, Java having the best time in the CPU tests, it is easy to understand that algorithms designed in VB and C# are slower than Java and AES CUDA is much faster than 134 times.

Table 2. GPU Comparison

GPU NVIDIA 8800		
[13]	AES with SBOX	AES without SBOX
0,00118259	0,00116233	0,00121234

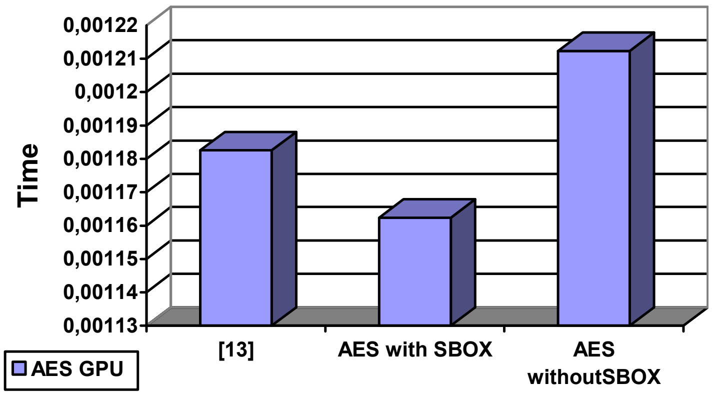


Fig. 9. Encrypting memory data.

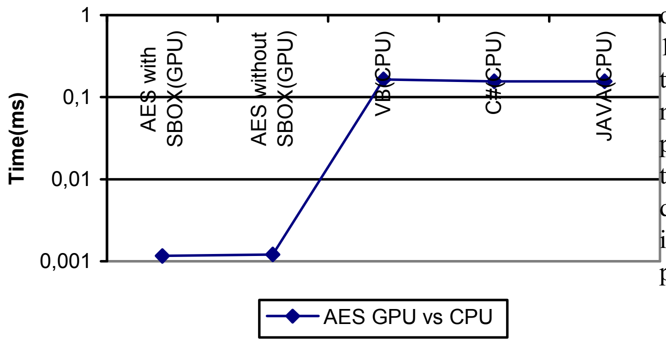


Fig. 10. GPU-CPU Comparison

Table 3. CPU results obtained in [10]

	PC1	
VB	C#	Java
0,16534	0,15681	0,15632
	PC2	
VB	C#	Java
0,2118	0,2118	0,40544
	PC3	
VB	C#	Java
0,03266	0,03186	0,20374

Table 4. Test platform description

Test Platform Descryption	
Procesor	Intel (R) Core(TM)2 Duo CPU E6750 @ 2.66GHz , Code Name Conroe LGA775, L1 I-Cache32 KB L1 D-Cache32 KB L2 Cache4096 KB, L2 Cache Speed2666.69 MHz
Memory (4GB)	Slot 1 Manufacturer Kingston Capacity 2048 MBytes DDR2 SDRAM SpeedDDR2-666 (333 MHz) Data Width64 bits Slot 3 Manufacturer Kingston Capacity 2048 MBytes DDR2 SDRAM SpeedDDR2-666 (333 MHz) Data Width64 bits
Hard disk	Vendor Seagate Model ST3500320AS SATA Size500 GBytes Current UDMA mode 5 (ATA-100) Rotational Speed 7200 RPM (Nominal)
Video Adapter	NVIDIA GeForce 8800 GT 512 Mbytes

The results we obtained in this phase concluded that AES with SBOX has a performance of 13.12 Mbps and AES without SBOX has a performance of 12.59 Mbps. These performances compared to [8] are mediocre. In [8] the author obtains cca. 10-20 Mbps for 4 KB input and 20-35 Mbps for 16 KB input. The reason for these results is that the input has the size of 16 KB, the same size as AES block(128 bit) and the number of encrypted blocks is 1, so, in this case the performance is affected by the overhead necessary starting the kernel. Obtaining these results we concluded that there is desired to pick an input that has a larger size. We chose an input value size of 10 MB. For this test the results are presented in the table 5.

Table 5. GPU results obtained for 10 MB stored in memory

AES with SBOX	AES without SBOX
914,2342	817,20711

Analyzing table 5 there can be seen that the results present a big performance jump if they are compared to the results obtained for 16KB input. The results obtained in this phase lead to the conclusion that AES with SBOX obtains a performance of 10,68 Gbps and AES without S-BOX is much faster obtaining a performance of 11,95 Gbps. [13] obtained 1832,34 ms meaning a performance of 5,3 Gbps. Comparing our results with the one obtained by the algorithm implemented in [13] we can say that the speedup of our algorithms is notable. These results couldn't be compared to the results obtained by the CPU as in the tests done in [9] and [10] the input of 10 MB was not used. The comparison between GPU and CPU can be only done for the 16KB size.

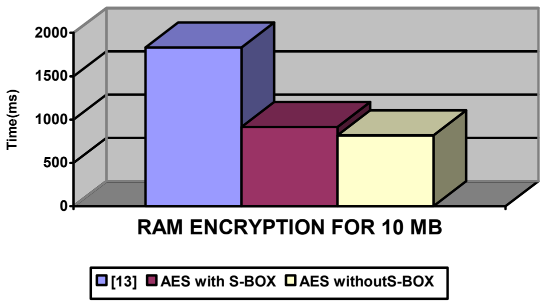


Fig. 11. GPU results obtained for 10 MB stored in memory

In the second phase the same tests were run for large data stored on the hard disk. In this phase files of 1GB, 2 GB..10 GB were encrypted. In the first step only three files with the size of 100 MB, 1 GB and 10 GB were tested. To these three files we added the 5 GB file and observed that the results obtained were in contradiction with the ones obtained for the 100 MB, 1 GB and 10 GB files. Because of this, the decision was taken of running the tests on all 10 files with the sizes of 1GB, 2 GB..10 GB.

The results of the tests are presented in table 5 and are represented in seconds. A comparison between CPU and GPU was done also for this phase. The values for the CPU were obtained from the tests done in [9] and [10].

Regarding the values acquired after running both implementations (with SBOX and without SBOX), it can be said that the trend is not and uniform ascending one. For files smaller than 3 GB AES with SBOX has better results, but starting from 4 GB to 8 GB AES without SBOX is the one that obtains better results. For the 9GB and 10GB files, AES with SBOX has, again, better results than AES without SBOX. Taking in consideration also the results from the tests in the first phase we could assert that AES with SBOX is faster for smaller values then 3 GB, and for values over 3 GB the better one is AES without SBOX. In contradiction with this statement come the last two results obtained for the files with 9 GB and 10 GB size where the situation was turned upside-down. As factors that could intervene in this situation, we could say that UNIX operating system may have problems in manipulating very large files stored on NTFS partition. The same behavior was seen in [9] and [10] were the trend was turned upside-down for the algorithms tested in UNIX/OpenSSL that ware 9 GB and 10 GB.

Analyzing the results from the two tables, 5 and 6, we can try to evaluate the performance difference in performance between CPU and GPU. For the tests done on CPU there are a lot of results on different development platforms, and none of the platforms having an ideal/linear behavior, we chose to calculate the speedup in two steps. We chose the best performance obtained form the GPU and compared it to the best result obtained by the CPU for the same file size. This comparison is found in table 7. In table 7 we also present the best time for GPU compared with the worst time on CPU. The best time on GPU has a maximum speedup of 17 compared to the best time obtained on the CPU for large data. This ratio is characteristic for a 100MB file, where the GPU is 17 faster than the CPU. If we compare the worst performance obtained on the CPU with the best result obtained on the GPU we can say that the GPU is aprox. 40 times faster than the CPU. Both these speed ups are characteristic for 100 MB file. For the other files the ratio is generally smaller than 10. For 1 GB file size the ratio between best time obtained on GPU and best time obtained on CPU is 5.5. If we compare best time obtained on GPU and worst time obtained in CPU for 1 GB file size the ratio is 10.6.

From the values presented in table 9 we can conclude that AES with S-BOX has the best performance for 1 GB file size. The performance in this case is 122 Mbps. In case of AES without S-BOX the best performance is obtained for the same 1GB file size, but is slower than AES with S-BOX. This is 99 Mbps. Starting from the 2 GB file size the performances of the two algorithms drop and than for the next files they are having an ascending trend. In table 9 CPU performances from [9] and [10] are presented for all the programming languages used for the entire pool of tests and for all files used in the test pool stored on the hard drive. It can be seen that if compared to the same tests done on GPU the CPU is much slower.

Table 6. GPU results for large file sizes

	[13]	AES with SBOX	AES without SBOX
100 MB	0,517	0,1435	0,835
1 GB	6,015	8,3556	10,344
2 GB	45,514	35,436	40,653
3 GB	72,048	50,660	51,588
4 GB	80,0260	59,3220	55,1460
5GB	85,5820	75,8590	70,5650
6 GB	92,7550	88,7670	81,6450
7 GB	102,3660	93,6430	89,8940
8 GB	110,725	102,276	101,863
9 GB	119,056	109,236	112,034
10 GB	125,646	120,434	121,470

Table 7. CPU results for large file sizes

	VB	C#	OpenSSL
100 MB	4,867000	2,437500	5,835000
1 GB	50,148000	23,734375	64,344000
2 GB	69,854000	72,687500	128,653000
3 GB	217,92150	155,421875	194,588000
4 GB	263,92150	196,703125	265,146000
5GB	312,00000	255,609375	326,565000
6 GB	474,26400	679,140625	408,645000
7 GB	435,60900	732,487500	507,894000
8 GB	578,79650	589,921875	529,863000
9 GB	606,43900	789,625000	602,034000
10 GB	615,75000	812,812500	656,470000

Table 8. GPU/CPU ratio

	GPU best vs. CPU worst	GPU best vs CPU Best
100 MB	40,662021	16,986063
1 GB	10,697257	3,945865
2 GB	3,630573	1,971272
3 GB	4,301648	3,067941
4 GB	4,808073	3,566952
5GB	4,627861	3,622325
6 GB	8,318215	5,005144
7 GB	8,148347	4,845807
8 GB	5,791326	5,201722
9 GB	7,228615	5,511315
10 GB	6,749029	5,112759

Table 9. GPU performance Mbps

	AES with SBOX	AES without SBOX
100 MB	69,68641	11,97605
1 GB	122,5525	98,99459
2 GB	57,79433	50,37759
3 GB	60,63956	59,54873
4 GB	69,0469	74,27556
5GB	67,49364	72,55722
6 GB	69,21491	75,25262
7 GB	76,54603	79,73836
8 GB	80,09699	80,42174
9 GB	84,36779	82,26074
10 GB	85,02582	84,30065

Table 10. CPU performance Mbps

	VB	C#	OpenSSL
100 MB	2,054654	4,102564	1,713796
1 GB	20,41956	43,14417	15,91446
2 GB	29,31829	28,17541	15,91879

3 GB	14,09682	19,76556	15,7872
4 GB	15,51977	20,82326	15,44809
5GB	16,41026	20,03056	15,67835
6 GB	12,95481	9,046727	15,03505
7 GB	16,45512	9,785833	14,11318
8 GB	14,15351	13,88658	15,4606
9 GB	15,19691	11,67136	15,30811
10 GB	16,63013	12,59823	15,59858

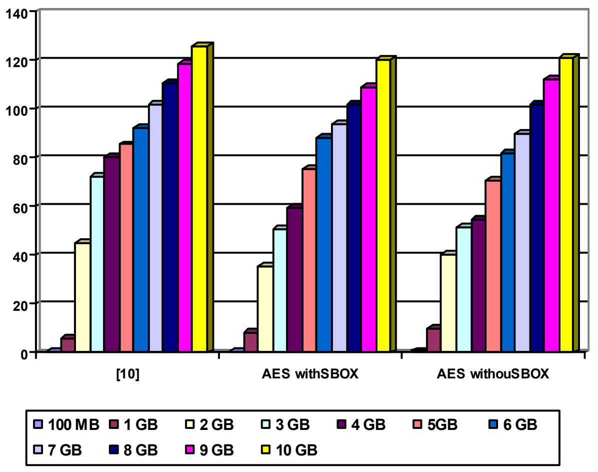


Fig. 12. Large file encryption

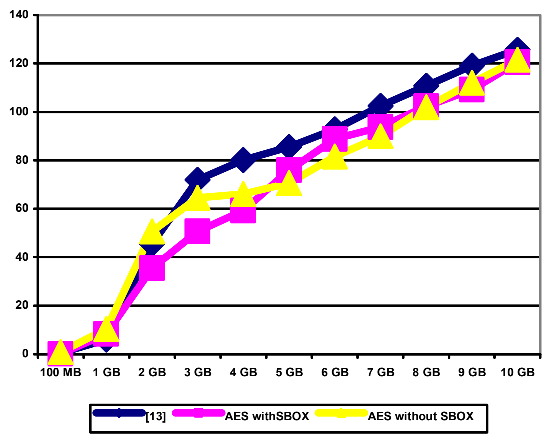


Fig. 13. GPU comparison

VI. Mathematical Complexity of AES

AES algorithm handles al bytes as Finite Field Elements. Finite Field Elements can be added and multiplied, but these operations differ from the ones used on numbers [15].

Adding of two finite field elements is done by modulo 2 adding (XOR operation) coefficients of the polynomials of the corresponding powers. Adding can be considered and addition of every bits in those bytes. Finite field elements adding is represented by: $\oplus$.

Multiplication in finite field is represented by $\bullet$, and in polynomial representation it corresponds with the multiplication of polynomials modulo irreducible polynomial of 8 degree. In case of AES this polynomial is $f(x)= x^8+x^4 +x^3 +x+1$ [15].

If we use two numbers in hexadecimal notation „57” and „83”, then „57” $\oplus$ „83”= „D4”, meaning 01010111 $\oplus$ 10000011 =11010100 and in polynomial form it can be written as follows $(x^6+x^4+x^2+x+1) \oplus (x^7+x+1)= x^7+x^6+x^4+x^2$[15]. Using this example modulo 2 addition is described where $1 \oplus 1=0$, $1 \oplus 0=1$ and $0 \oplus 0=0$.

In case of modulo f(x) multiplication, if we use the same values as in the previous example „57” and „83”, we can write „57” $\bullet$ „83”= „C1” because

$(x^6+x^4+x^2+x+1) \bullet (x^7+x+1)= x^{13}+x^{11}+x^9+x^8+x^7+ x^7 + x^5 + x^3 + x^2 + x + x^6 + x^4 + x^2 +x+ 1 = x^{13}+x^{11}+x^9+x^8 + x^6 + x^5 + x^4 + x^3 + 1$ modulo $f(x)= x^7 + x^6 + 1$ [15].

Multiplication, opposed to addition, has no longer simple byte level operations. Multiplication, as defined before, is associative, and, if another polynomial is used, b(x) with a degree less than 8 then $b^{-1}(x)$ is its inverse. If $b(x)a(x) + f(x)c(x)=1$ and $a(x) \bullet b(x) \bmod f(x) =1$, then the following can be concluded $b^{-1}(x)= a(x) \bmod f(x)$ and $a(x) \bullet (b(x)+c(x))=a(x) \bullet b(x)+a(x) \bullet c(x)$ [15].

Multiplying by x is obtained reducing the result modulo $x^8+ x^4 +x^3 + x + 1$. If the polynomial has the maximum 7 degree than the result of the multiplication is already in reduced form.

Four term polynomials that have coefficients in GF(2^8) are different from the one already presented meaning that these polynomials have as coefficients bytes instead of bits.

In this case we can presume that we have two polynomials $a(x)= a_3x^3 + a_2x^2 + a_1x + a_0$ and $b(x)= b_3x^3 + b_2x^2 + b_1x + b_0$. Multiplication is done in two steps. In the first step c(x) polynomial is obtained this way: $c(x)= c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0$, where[15]:

$c_0=a_0 \bullet b_0$

$c_1=a_1 \bullet b_0 \oplus a_0 \bullet b_1$

$c_2=a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2$

$c_3=a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3$

$c_4=a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_1$

$c_5=a_3 \bullet b_2 \oplus a_2 \bullet b_3$

$c_6=a_3 \bullet b_3$

In the second step , the result, c(x) is reduced modulo polynomial of degree 4. For AES AES this polynomial is x^4+1. a(x) multiplied by b(x) is d(x) and tha corresponding notation is $d(x)=a(x) \otimes b(x)$. $d(x)= d_3x^3 + d_2x^2 + d_1x + d_0$, where[15]:

$d_0=(a_0 \bullet b_0) \oplus (a_3 \bullet b_1) \oplus (a_2 \bullet b_2) \oplus (a_1 \bullet b_3)$

$d_1=(a_1 \bullet b_0) \oplus (a_0 \bullet b_1) \oplus (a_3 \bullet b_2) \oplus (a_2 \bullet b_3)$

$d_2=(a_2 \bullet b_0) \oplus (a_1 \bullet b_1) \oplus (a_0 \bullet b_2) \oplus (a_3 \bullet b_3)$

$d_3=(a_3 \bullet b_0) \oplus (a_2 \bullet b_1) \oplus (a_1 \bullet b_2) \oplus (a_0 \bullet b_3)$

These operations together with byte operations like xtime() are used as internal operations in case of AES. As it can be seen, these are not complex operations and are not consuming a lot of computing power as asymmetric algorithms do.

VII. Conclusion and future work

The goal of this paper is to study the possibility of using an alternative solution in cryptography. The alternative we studied is the use of GPU in non graphic operations. As we concluded in this paper and in [9] and [10], results obtained after running the tests processors offer results that varies according to the file size, algorithm tested, memory capacity, programming language or operating system.

The results obtained have proven that implementing a cryptographic algorithm on a video processor brings a significant performance speedup compared to legacy processor performances obtained on test platforms. The performance gain was 134 faster than the processor, when data is stored in RAM, and up to 17 times faster then the processor when large data, stored on hard drive are used. When testing large file encryption a element that affected the performance was the time necessary to bring the data form the hard disk, these data being large volume data. AES performance on CUDA was between 12 Mbps and 11,95 Gbps

The tests in this paper were done in three phases. All of them were using data stored in the GPU memory. One of them was the implementation done by another researcher [13]. The other two were implementations that we designed. One was using AES with lookup tables and the other implementation uses the GPU computational power to calculate each operation instead of using lookup tables.

AES implementations in this paper were done using parallelization. Parallelization that GPU offers is a greater that CPU can offer. This fact alone is enough reason in doing these tests. The values we obtained proved that GPU high computing power, GPU memory bandwidth are advantages in choosing this processor as a cryptographic coprocessor. The big difference in performance was obtained due to the fact that classic processors are optimized for serial processes, use of big cache sizes and complex instruction sets. In order to obtain a better performance than the CPU we had to ensure the use of all GPU kernels

In this paper we did the followings:

In doing the tests in [9] an [10] we realized that it is necessary to implement cryptographic primitives on a different platform. AES algorithm was chosen and the platform on which the tests had to be done was a video processor (GPU). As a special feature the chosen GPU had to offer is that it is capable to run CUDA code, the programming environment that is NVIDIA proprietary.

In the first phase we tested the algorithm implemented in [13] and the results were compared with the ones obtained in [9] an [10].

In the next phase we implemented our own AES on GPU using CUDA using the video graphic card as a cryptographic coprocessor to help the CPU. In this step we analyzed the existing work done by other researchers and after doing this we proposed our own implementation of AES on GPU(CUDA).

We propose, implemented and tested two solutions: in one we used SBOX lookup table for the SubBytes() transformation and in the second one we used the GPU computing power to calculate the operations necessary in SubBytes() without the use of lookup tables. In doing so we tested and compared how fast data is accessed from memory and how fast the operations are done by the GPU.

In this paper we have not done the followings:

In the research we did we have not done the decryption part of the AES algorithm. The decryption process is identical with the encryption process, because the code is the same. The counter value is concatenated with the nounce and encrypted. The value is XORed with the data block.

The AES algorithm we chose to implement is AES 128 (block size 128 bit, key size 128 bit , 10 rounds),but we didn't implemented AES 192 and AES 256.

We have not done a security analysis and possible attacks, after implementing the algorithm on the GPU in C for CUDA.

The test were done without stopping the X server(Kubuntu 9.04).

Implemented AES algorithms that are adapted for CUDA were not tested on other operating systems (Windows etc), but only in Kubuntu.

AES parallelization implemented for CUDA and GPU was not done also for CPU with more cores.

In a second stage, we will try to integrate the algorithm from the previous step in a software application like OpenSSL, so that this can use the algorithm and benefit from the graphic processor acceleration. Also this OpenSSL implementation will benefit also from the decryption phase.

We will also try to implement and test AES 192 and AES 256 on GPU, encryption and decryption and OpenSSL adaptation to use these algorithms.

ACKNOWLEDGMENT

I offer my regards to all of those who supported me in any respect during the completion of this project.

REFERENCES

[1]O.W. Harrison, „Practical Symmetric Key Cryptography on Modern Graphics Hardware", 17th USENIX Security '08 Symposium, San Jose USA, 2008

[2]F. Granelli , A novel methofology for analysis of the computational complexity of block ciphers, University of Trento, 2004

[3]J.L. Gustafson, HINT: A new way to measure computer performance. Hawaii International Conference on System Sciences, 1995, pp I:392-401.

[4]M. Kipper, J. Slavkin, and D. Denisenko, "Implementing AES on GPU", University of Toronto, http://www.eecg.toronto.edu/ ~moshovos/ CUDA08/arx/AES_ON_GPU_report.pdf, 2009

[5]B. Luken and M. Ouyang "AES and DES Encryption with GPU", Proceedings of the ISCA 22nd International Conference on Parallel and Distributed Computing and Communication Systems, pp 67-70, 2009

[6]S. Manavski, "CUDA Compatible GPU as an efficient Hardware Accelerator for AES Cryptorgraphy" , IEEE International Conference on Signal Processing and Communication, ICSPC 2007, pp. 65–68, Nov. 2007

[7]M.Solga, B.Groza, "Evaluarea performanțelor computaționale pentru funcții criptografice simetrice şi asimetrice, pe platformele Windows şi Unix," unpublished 2008.

[8]Takeshi Y., „AES Encryption and Decryption on the GPU", GPU Gems 3, http://http.developer.nvidia.com/GPUGems3/gpugems3_ch36.html, 2007

[9]R. Tomoiaga and M. Stratulat, "Evaluation of DES, 3 DES and AES on WINDOWS and UNIX platforms", ICCC-CONTI 2010 IEEE International

Joint Conferences on Computational Cybernetics and Tehnical Informatics, Timişoara 27-29 may 2010

[10] R. Tomoiaga and M. Stratulat, “AES Performance Analysis on Several Programming Environments, Operating Systems or Computational Platforms”, The Fifth International Conference on Systems and Networks Communications ICSNC 2010, 22-27 August 2010

[11] M. Wenbo, Modern Cryprography : Theory an Practice (Hewlett-Packard Profesional Books),Ed. Prentice Hall

[12] Y. Yeom, Y. Cho, and M. Yung “High-Speed Implementations of Block Cipher ARIA Using Graphics Processing Units,” in Proceedings of the 2008 International Conference on Multimedia and Ubiquitous Engineering (April 24 - 26, 2008). MUE. IEEE Computer Society, Washington, DC, 271-275. 2008.

[13] http://math.ut.ee/~uraes/openssl-gpu/ 17.05.2010

[14] Nvidia CUDA Programming Guide, 2009 , NVIDIA

[15] Federal Information Processing Standards Publication 197, 26.11.2001, Advanced Encryption Standard(AES)

[16] http://www.hardwareheaven.com/reviews/8800GTs/whatis.php

[17] HINT, 2006, www.sc1.ameslab.gov/sc1/HINT-HINT.html.