Accelerating Solution Proposal of AES Using a Graphic Processor

Radu-Daniel TOMOIAGĂ, Mircea STRATULAT
Politechnica University of Timişoara, 720229, Romania
a-raduto@microsoft.com,mircea.stratulat@cs.upt.ro

***Abstract*—The main goal of this work is to analyze the possibility of using a graphic processing unit in non graphical calculations. Graphic Processing Units are being used nowadays not only for game engines and movie encoding/decoding, but also for a vast area of applications, like Cryptography. We used the graphic processing unit as a cryptographic coprocessor in order accelerate AES algorithm. Our implementation of AES is on a GPU using CUDA architecture. The performances obtained show that the CUDA implementation can offer speedups of 11.95Gbps. The tests are conducted in two directions: running the tests on small data sizes that are located in memory and large data that are stored in files on hard drives.**

***Index Terms*—AES, benchmark, cryptography, CUDA, GPU.**

I. INTRODUCTION

Legacy processor development is close to its technological limits. For a better performance of the processor vendors usually raise the frequency, but in doing so, the temperature raises and also the power consumption. A solution to this problem is offered by multi core processors. Inserting more cores offers a performance boost but it is restricted by using special developed software for more cores. An alternative to multi core processors could be Cloud Computing, but this alternative is complex and it needs a large number of resources (computers). These two solutions can prove to be expensive. A multi core processor is much more expensive than an average video graphic card and can not offer the performances that a GPU can offer [12]. Cloud Computing is based on using a lot of hardware resources from many computing systems, and this reflects in larger expenses.

A solution to the problems raised before is offered by the graphic processors (GPU). A GPU contains many cores and can run more threads in parallel. From all cryptographic primitives we chose AES algorithm as this is the symmetric key standard algorithm. In [32], the author starts from Cook [9] success in 2005 of implementing a cryptographic algorithm on a GPU, analyses the performances obtained using DirectX and OpenGL, and concludes that an Intel Core 2 Quad (QX6850) processor is capable of 96 GFLOP, while a NVIDIA GEFORCE 8800GTX graphic card is capable of 330 GFLOP. Also AES algorithm obtains a 4.5 Gbps performance and DES a 2.8 Gbps performance.

Michael Kipper, in [17], runs AES on GPU and obtains a performance that is 14.5 times faster then the one obtained on the CPU. He also states that brute force attempts for cracking AES using GPU are will not have success on this speedup. In a similar work Luken Brandon, [19], uses adapted AES and DES algorithms on GPU. The tests were conducted on files up to 100 MB sizes, and the performances obtained were as follows: AES is 3.75 times faster than the CPU and DES is 4.5 times faster than a CPU.

Manavski Svetlin presents in [20] the results he obtained when accelerating AES using CUDA capable GPU from NVIDIA. The best performances were obtained when AES 128 was used. The input file size was 8 MB and the speedup was 19.60 times compared to the CPU.

Another AES implementation on GPU was realized in [26]. In this paper the author develops two tests called „Vertex Program" and „Fragment Program". The program was written in OpenGl based on assembler language. Using a Pentium 4 processor at 3 Ghz 2MB level 2 cache, 1 GB RAM and a GeForce 8800 GTS 640 MB video card as a platform, for input sizes between 4KB and 16 MB he obtains performances between 10Mbps and 95 Mbps. In his tests the author used CBC mode which can be parallelized only for decryption. At the end, he concludes that for a better performance CTR mode can be used because this mode is recommended for high parallelism and simplifies the code as for encryption and decryption the same source code is used.

In [5] the author presents more implementations of AES for AVR microcontroller on 8 bit, Cell and on GPU (8800 GTX and GTX295). Using CUDA for NVIDIA the results obtained are 59.6 Gbps (GTX295) and 14.6 Gbps (8800 GTX) for encryption and 52.4 Gbps (GTX295) and 14.3 Gbps (8800 GTX) for decryption. Implementations in [5] use only lookup tables as a main optimization method. It is not specified for which input size were the results obtained. CTR mode was used, but in this case the source code for encryption being identical with the source code of decryption, there should not be so big differences between the performances obtained for encryption and decryption. [7] presents a methodology for evaluating software applications. It uses a new analysis concept designed to significantly ease the process and it presents a set of statistical and experimental data collected using the new analysis structure on a representative set of scientific and commercial applications. [33] presents a method of modifying MixColumns for AES and concludes that it is much faster than the original, but does not present the testing process used and the speedup obtained. In [6] an AES on FPGA acceleration was done. The performance obtained was 2 Gbps. The main disadvantage of such a solution is the acquisition of a FPGA. This performance was beaten in 2007, when [20] obtained 8.28 Gbps, using a video card similar to the one used in this our tests [22] implements

Digital Object Identifier 10.4316/AECE.2011.04016

AES on FPGA and obtains an astonishing 21.56 Gbps performance. A similar implementation done in [22] is the one realized in [14], where results of 25 Gbps are presented, but being an AES implementation on FPGA it presents the drawbacks from [6] and the need of hardware equipment with a high number of gates (1-2 millions). Another paper is [4], where an AES parallelization on a multiprocessor system is proposed. During the tests processors like 64x Itanium2 1.5GHz (SGI Altrix 3700) with 16 processors where used and the results showed that these implementations were 5.554 times faster for encryption and 12.357 times faster for decryption. [18] presents a processor implementation for accelerating AES with performances of 5.8 Gbps. [3] realizes an AES implementation on CUDA and speaks about performances that go over 10Gbps. [15] does a comparison similar to the one proposed in this paper and compares the CPU and the GPGPU. The conclusion drawn is that legacy processors are not enough to reach and compare with the objectives proposed and finding a solution to accelerate AES on a GPU proves to be extremely difficult. [34] proposes an AES on CUDA implementation in which it uses ECB mode. In [13] a cryptographic coprocessor based on 0.18-µm CMOS technology is presented. The performances obtained, after running the tests are 3.84Gbps, and this processor is capable of running in ECB, OFB and CBC mode. [24] presents a Rijandael implementation on FPGA in which it speaks of a 18.5 Gbps performance. Another AES implementation is presented in [8]. This speaks of using a 10 $ FPGA and of performance that reaches 166 Mbps. In [1] and [2] the need of identifying new solutions for co processing in case of cryptography is presented. A solution like this is offered by [16] when proposing a cryptographic co processor, called HSSec, is presented. This is capable of using SHA1 and SHA512 as hash functions and AES 128 as a symmetric algorithm. The best performances it obtains are 1 Gbps.

II. GPU AND CUDA

CUDA, introduced in 2006 as new parallel computation architecture, has a new set of instructions and a new parallel programming model. CUDA offers a new software environment which allows the programmers to use C as a programming language for the GPU. The GPU is seen as computational device capable of executing a high number of threads in parallel, when using CUDA. The GPU acts as a coprocessor for the CPU. If a section of a program that can be executed in parallel can be isolated this one can be adapted to be run on GPU as more independent threads in parallel. When a function is called it will run on a grid in multiple blocks of threads. The number of blocks and threads on a grid can be configured offering flexibility [29].

Using CUDA, threads can be used to 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 from all blocks can access the global memory [30].

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 [12]

The maximum number of threads is given by the maximum number of thread for each block multiplied by the number of blocks. These blocks are organized in one or two dimensional grids[29].

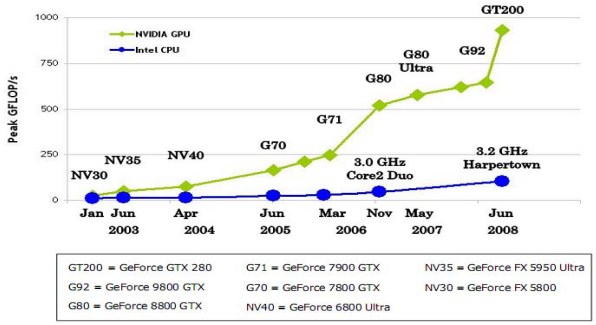

Figure 1 CPU VS GPU. Gflop/s [10]

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[31]. 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 [12]. 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 [21].

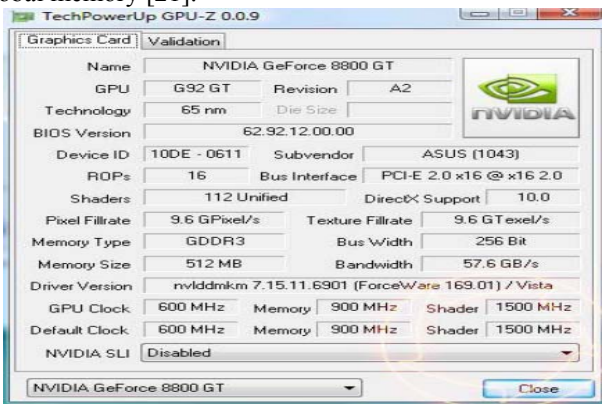

Figure 2 Nvidia 8800 GT Specs [21]

TABLE I. COMPARISON 8800 GT VS 8800 GTS VS 8800GTX

	8800 GT	8800GTS	8800 GTX
Memory	512 MB 256 bit GDDR3	640 MB 384 bit GDDR3	768 MB 384 bit GDDR3
Code	G92-200	G92-400	G80
Frequency	600 Mhz	650 Mhz	612 Mhz
GB/s	57.6 GB/s	62.1 GB/s	86.4 GB/s
Processors	128	128	128

III. ADAPTING AES TO RUN ON GPU

Implementing a cryptographic algorithm to run on a graphic processor is justified because, theoretically, it is cheaper to use a GPU as a co processor to relieve the CPU from intense computing tasks, than purchasing a dedicated cryptographic co processor that is more expensive.

Adapting an algorithm, that has a computational complexity that consists in simple byte operations (AND, OR, XOR, Shifting, 32 bit adding etc.), on a video card

which is designed to run complex operations and floating point operations, is the starting point of our scientific research.

Implementing and adapting AES on a GPU must be done taking in consideration the following aspects [23]: CPU AES optimization must be reanalyzed because this is based on lookup tables which, could theoretically slow down the process in the GPU case. In order to affirm this exactly the implementation of AES on GPU was done in two steps: in the first step was implemented AES algorithm using lookup tables and in the second step AES without lookup tables, based only on operations, was implemented.

Graphic processors were designed for parallel operations and are not adequate for cryptographic algorithms, which, in general are based on the previous block for encryption (are serial algorithms). Algorithm implementation must be based more on arithmetic complexity than on memory operations. The advantages of using a graphic card that supports CUDA environment, reside in the fact that CUDA can use shifting operations, offers flexibility on memory access, data can be defined directly in the GPU without the need of an extra copy operation from the CPU [23].

In case of legacy processors, optimization of AES algorithm was based on replacing arithmetic operations with lookup operations of data stored in tables (memory). Unlike the CPU, the GPU can do more operations in a single clock cycle than the CPU. According to [23] the GPU has greater latencies than the CPU, when memory access is analyzed, because GPU holds the execution during this process. In the test conducted in [25], the author concluded that when a large number of operations are used, the GPU operates quicker when compared with memory accesses.

When running AES algorithm on a GPU, taking in consideration that a better performance that obtained on CPU is desired, and the GPU is specially designed to run many tasks in parallel, only the AES modes that offer parallelization should be selected. These modes are ECB and CTR. Also CBC mode can be parallelized but only in the decryption phase. The modes for AES are Electronic Codebook ECB, Cipher Block Chaining CBC, Cipher Feedback CFB, Propagating Cipher Block Chaining PCBC, Output Feedback OFB and Counter Mode CTR. In the figures bellow, only the encryption step of each mode is presented.

From the two modes that are capable of parallelization only the CTR mode was chosen. The choice was taken based on the fact that Counter Mode offers more security compared to ECB and is suitable for massive parallelization. The disadvantage of ECB mode is that the same block of clear text will encrypt in the same cipher text block. This is why ECB is not recommended in cryptographic protocols at all. So ECB is not recommended for practical use in real cryptographic applications. Due to the fact that the implementation done during this research is desired to have a practical use CTR mode was chosen. Analyzing the operation mode of CTR we can say that the decryption phase needs not to be implemented anymore as for this phase the same source code is used as for the encryption phase. For testing only the encryption function was implemented and if it is needed, for the decryption step the encryption function is called again.

CTR uses a simple method to generate necessary keys. It concatenates a nonce with the value of the counter and encrypts it in a single block. The nonce needs to be smaller than the size of a block, because there must be sufficient space for the i^{th} value in the counter. O facility offered by CTR is the easiness of accessing random text parts from the clear text. The main advantage of CTR is that it can be parallelized for high speed applications [11]. The main disadvantage of CTR is that the algorithm can be the victim of specialized Hardware Fault Attacks [27].

A simplified scheme for implementing the algorithm in parallel is presented in fig. 3.

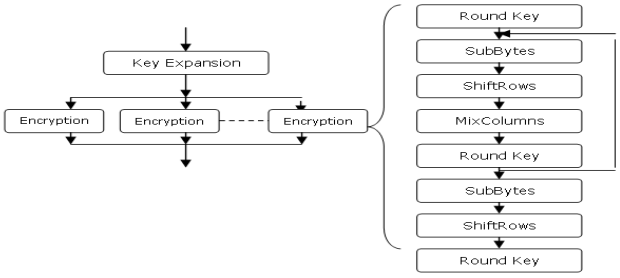

Figure 3. AES parallelization

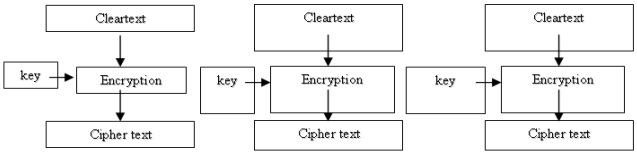

Figure 4. Electronic Code Book ECB mode

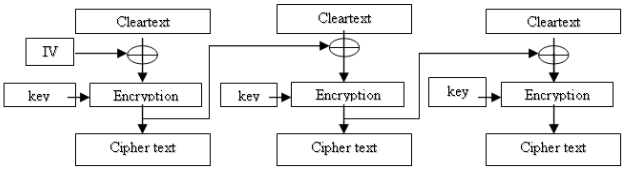

Figure 5. Cipher Block Chaining CBC mode

The key expansion is done on CPU and the actual encryption is done on the GPU. The key expansion consists of operations that execute in serial and this is the reason why this step is done on the CPU. The unnecessary GPU load with serial operations is not desired.

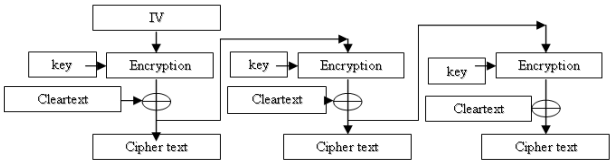

Figure 6. Cipher Feedback CFB mode

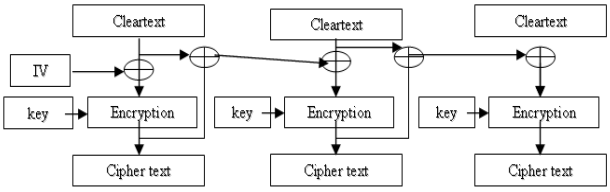

Figure 7.Propagating Cipher Block Chaining PCBC mode

As an optimization all threads will be set to use the global memory. In doing this a memory access grouping can be done. Data accesses coalesce to permit the GPU to do faster read/write memory operations [17]. Global memory access is done, in the initial phase, before data processing. Data are then moved in shared memory where it can be accessed faster. If every thread loads data in shared memory from the

global memory, which is possible never to use, a synchronization step is needed before using the shared memory. Synchronization is necessary also for writing data back into the global memory [17].

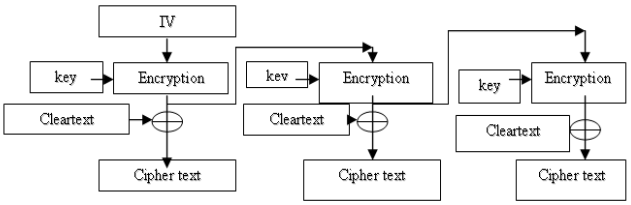

Figure 8. Output Feedback OFB mode and Counter Mode CTR

A possibility of AES implementation in C for CUDA, is the one in which, similar to AES optimization for CPU, 16x16 bytes lookup tables are used. These lookup tables, being constant, can be loaded in the shared memory of the GPU and can be accessed by threads in the same block. For small data sizes CPU will outperform the GPU [17]. In the tests conducted in [23], it has been proven that for larger volume sizes better encryption performances are obtained in case of calculations when compared to lookup tables. Being accessed by all the threads the lookup tables can be loaded in the global memory where the data can be accessed by every thread.

Another solution would be the use of constant memory for keeping the S-BOX and the round keys. The advantage in this solution is that the data can be assigned in the design phase and can be accessed also by the CPU [19].

A CPU parallel implementation of AES is presented in [17], but the results obtained are not even close to the ones obtained by running the algorithm in parallel on GPU and the CPU is not capable of running as many threads in parallel as a GPU does. This is why CPU parallelization will not be as effective as GPU parallelization.

Developing AES in C for CUDA, two directions were followed: in the first one the computation power of the GPU was used by writing the code to operate the exact operation of AES and in the second one lookup tables were used. These lookup tables were stored in memory and the second direction was based a lot on memory access. Using CUDA, the AES optimization for CPU could be implemented and adapted also for GPU. Geforce 8800 GT/G80, unlike its predecessors, is a scalar processor and does not need to combine instructions in vector operations in order to obtain maximal computational power. Also the ability of G80 to execute logic operations like 32 bit XOR offers a theoretical performance gain to this solution.

The main problems encountered in developing, compiling, implementing and testing the software applications were:

The feedback offered by the nvcc CUDA compiler was poor and eliminating errors was difficult in case of .cu files.

Debugging was also difficult, because usually the errors specified by the compiler were not detailed as needed [23].

When implementing programs in CUDA the most important thing was to see the GPU as a coprocessor and not as a processor. In doing so, when programming in C the point of view had to be changed from a classical programming and the GPU had to be treated as a host and not as a device.

In case of allocating a lot of memory, the program stopped running and was not indicating an error that would specify the problem encountered. This slowed down the process and a similar problem was indicated by [23].

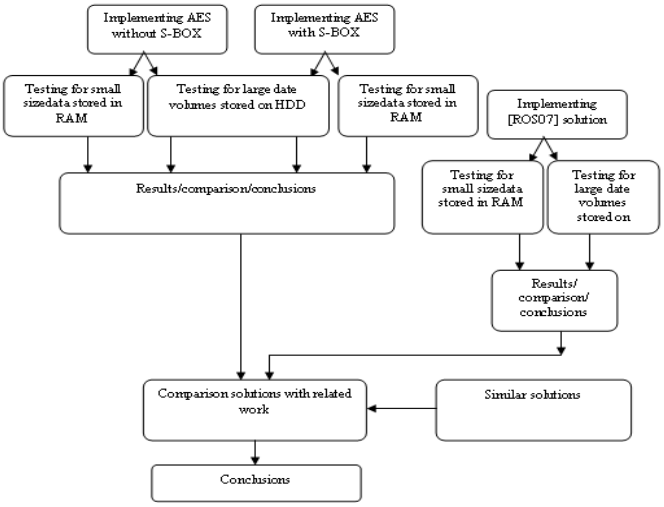

Figure 9. Software Implementing and testing

IV. TESTING AND RESULTS

After implementing the solutions presented in the previous chapter tests were conducted in order to see the performances obtained for the proposed solutions. Two types of results were analyzed: the results of the programs using lookup tables and the programs that were not using lookup tables.

Input sizes were: 16 KB for data stored in RAM and 100 MB, 1 GB, 2 GB... 10 GB for data stored on HDD.

The tests were done in two phases: data encryption stored in memory and data encryption stored on HDD. In each phase were tested three algorithm implementations: the one implemented by [23], AES algorithm implemented to use lookup tables stored in global memory and AES algorithm that executes every operation on the GPU without lookup tables.

At the beginning a random value is generated and will be used in the encryption process. For the tests conducted with data stored in RAM the input size was 128 bit, exactly the size of the AES block. When testing data encryption stored in RAM, "on the fly" data encryption is simulated. The algorithms are run recursively using this scheme: buffer = algorithm_AES(buffer+ random value).The number of iterations used was 1.000.000 and at the end an average value was calculated based on the total time.

The necessary runtime is measured and than divided by the number of repetitions. In this way an accurate average measured time is obtained for an average one iteration.

The obtained results were compared with the results obtained by the algorithm developed in [23]. In table 2 three values are presented. One is obtained after running AES with lookup tables; the second one is obtained after running AES without lookup tables as in [35]. The third value is the one obtained after running the algorithm implemented in [23].

The time obtained for the algorithm that uses S-BOX tables is 1.162E-6 ms. This value is independent from the time necessary to generate the random value but is dependent to the data encrypted from the shared memory, saved and re encrypted for 1.000.000 times. The encryption

time is not affected by the time necessary in bringing data from the CPU memory, because data is stored and processed directly from the GPU memory. Because all the threads need to access the lookup table, this was stored in the global memory so that it would be available to every thread.

When using AES without lookup tables the computational time is 1.212E-6 ms and is greater than the one when lookup tables are used data is accessed from the shared memory of the GPU and written back to the same memory for the next iteration.

Comparing the two results obtained, for the small data stored in memory we can conclude that AES with lookup tables, as in the case of CPU AES optimization, is faster than the one that does not use lookup tables stored in memory.

Comparing the best time obtained by the CPU under Visual Basic, C# and Java [28] with the best time obtained by the GPU it is easy to say that GPU is faster. The best time obtained by the CPU is under Java 1.563E-3 ms. Performance ratio is about 134, meaning that the GPU is 134 times faster than the CPU and AES on GPU is with 1.551E-3 ms quicker than the CPU. AES on GPU without lookup tables is about 129 times faster than the CPU.

Analyzing the three values in table 2 it can be concluded that AES with lookup tables is faster than [23] AES, and the other implementation is slower when compared to [23].

The results in this phase led to the conclusion that AES with S-BOX has a performance of 13.12 Mbps and AES without S-BOX has a performance of de 12.59 Mbps. These performances are weak compared with the performances other authors obtained in similar papers. In [26] the author obtains performances of 10-20 Mbps for input values of 4 KB and 20-35 for input values of 16 KB size.

A reason for these results is that the input size is 16KB, the same size as the block of the algorithm (128 de bit) and the number of parallel encrypted blocks is 1. In this case the performance is affected by the overhead needed to start the kernel. After obtaining these results appeared the need to choose bigger input sizes. The chosen value after some experiments was 10 MB. In this case the times obtained are presented in table 3.

TABLE II. AES MEMORY RESULTS

AES with SBOX	AES without SBOX	[23]
1.162E-6	1.212E-6	1.182E-6

TABLE III. AES. 10 MB INPUT SIZE

AES with SBOX	AES without SBOX
914.2342	817.20711

These two results show a fantastic performance leap compared with the ones for 16 KB input size. The result obtained in table 3 showed that AES with S-BOX has a 10.68 Gbps performance and AES without S-BOX is faster with a 11.95 Gbps performance. [23] implementation had a 1832,34 ms time and a 5.3 Gbps performance for 10 MB input. Comparing table 2 and 3, a notable performance boost can be seen.

In the second phase the tests were done for large date sizes, large file sizes from 1GB, to 10 GB. In a first step only three files were tested: 100 MB, 1 GB and 10 GB. Testing also the 5 GB file the results obtained were in contradiction with the results obtained in the tests for 100 MB, 1 GB and 10 GB. At this point the decision to test from 1GB to 10 GB was taken. The results of these tests are presented in table 4 in seconds. It can be said, analyzing the values obtained for both implementations, that the trend does not respect a uniform growth.

For the files up to 3 GB AES with S-BOX has better results. Between 4 GB and 8 GB AES without SBOX has a better performance and in the end for 9GB and 10GB files AES with SBOX obtains again better results. Taking in consideration the results obtained in table 2, we can say that AES with lookup tables is faster for smaller sizes (up to 3 GB) and over 3 GB AES without lookup tables is quicker. In antithesis with this affirmation come the last two results for 9 GB and 10 GB file sizes where the situation is turned upside down. Factors that could have led to this situation could be the problems that UNIX operating systems have in managing very large files stored on a NTFS partition. The same problem appeared in [28] and [29] when 9 GB and 10 GB file sizes where tested under UNIX and there could be observed a sudden time growth.

TABLE IV. AES. LARGE DATA INPUT SIZES

	[23]	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

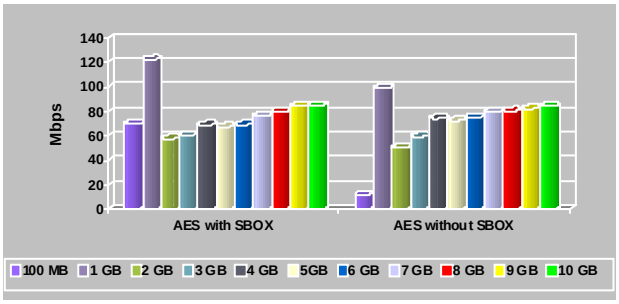

Figure 10. Implementation performances [31]

The best speedup of the GPU is about 17. This value is characteristic for a 100 MB input file size. Beside this speedup that is obtained for 100 MB file, the speedups haven't exceeded 10 except only in one case. For files greater then 1 GB it can be said that the best speedup is about 5.5.

In fig. 10 the performance for AES with S-BOX for 1 GB file is 122 Mbps. In case of AES without S-BOX for 1 GB file the performance is 99 Mbps. For the 2 GB file the performances of the two algorithms drop and for the other files follow a growing trend. When using data stored on HDD the performance is affected by the access time needed to bring the data from the HDD to memory. Also the PCIexpress capable of 3.5 Gbps and 5.5 Gbps bandwidth. Also SATA hard drives are capable of 1.5 Gbps and 3 Gbps bandwidths.

V. CONCLUSIONS

In this paper was presented the related work and the most similar papers were presented. Analyzing the related work appeared the need to implemented AES with and without

lookup tables to see which offers a greater performance.

The possibility of adapting AES for C under CUDA to run on GPU is presented and the software tuning and compiling method are discussed. Two algorithms were implemented and tested. The techniques, algorithms and methods used were presented.

After running the test the results obtained were discussed and explained. The best performance obtained was 11.95 Gbps and it was for AES without lookup tables when 10 MB data size was used for input. This result certifies that this implementation is better than some of the papers presented in the related work section were authors used GPU to encrypt data using symmetric algorithms like[32], [20], [26], [18], [3], [13], [6].

In order to gain the best performance from the GPU implementation we need to have all the cores of the GPU busy running instructions. The results obtained after running the tests showed that the GPU can be 134 times faster than the CPU and showed performances between 12 Mbps and 11.95 Gbps. A comparison between the implementations in this paper and similar work from other authors is presented in fig. 11. As a future research we will integrate AES for GPU algorithm in a software application like OpenSSL, so that this software can benefit from the graphic processor acceleration. Another future research would be to adapt asymmetric algorithms to run on a GPU and benefit from the GPU acceleration.

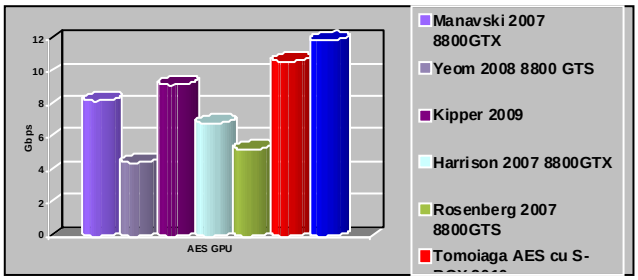

Figure 11. Related Work vs. own solutions [31]

REFERENCES

[1] Atanasiu A. – Secret Sharing Schemes, capitol in Informatics Security Handbook, vol 2 (Ivan I., C. Toma eds), Editura ASE, 2007.

[2] Atanasiu, A. – Securitatea informaţiei, vol. 1 (Criptografie), Ed. Infodata, Cluj, 2007.

[3] Biagio A. D., Barenghi A., Agosta G., Pelosi G., "Design of a parallel AES for graphics hardware using the CUDA framework,"in Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing, 2009, pp. 1–8.

[4] Bielecki W., Burak D., „Parallelization of the AES Algorithm", Proceedings of the 4th WSEAS International Conference on Information Security, Communications and Computers, pp. 224-228, Tenerife, 2005

[5] Bos J. W., Osvik D.A., Deian S., „Fast Implementation of AES on Various Platforms", SPEED-CC -- Software Performance Enhancement for Encryption and Decryption and Cryptographic Compilers, 2009, Berlin, ICT-2007-216676

[6] Brokalakis A., Michail H., Kakarountas A., Milidonis A., Goutis C., „A High-Speed and Area Efficent Hardware Implementation of AES-128 Encryption Standard" Proceedings of the 5th WSEAS International Conference on Multimedia, Internet and Video Technologies,pp. 125-129 Corfu, 2005

[7] R. Cheveresan, S. Holban, "Workload Characterization an Essential Step In Computer Systems Performance Analysis – Methofology and Tools", Advances in Electrical and Computer Engineering, ISSN: 1582-7445, 2009

[8] P. Chodowiec , K. Gaj, „Very Compact FPGA Implementation of the AES Algorithm", CHES 2003, Proceedings, LNCS Vol. 2779, pp. 319-333, 2003

[9] D.L. Cook, J. Ioannidis, A.D. Keromytis, and J. Luck. "CryptoGraphics: Secret Key Cryptography Using Graphics Cards." In RSA Conference, Cryptographer's Track (CT-RSA), pp. 334–350, 2005.

[10] Nvidia CUDA Programming Guide, 2009 , NVIDIA

[11] Ferguson N., Schneier N., "Practical Cryptography", Wiley Publishing , 2003

[12] Harrison O.Waldron J., Practical Symmetric Key Cryptography on Modern Graphics Hardware, 17th USENIX Security '08 Symposium, San Jose USA

[13] Hodjat A., Hwang D., Lai B.C., Tiri K., Verbauwhede I., „A 3.84 Gbits/s AES crypto coprocessor with modes of operation in a 0.18-um CMOS Technology", Proceedings of the 15th ACM Great Lakes Symposium on VLSI 2005, pages 60--63. ACM, ACM Press, April 2005

[14] Good T., Benaissa M., „AES on FPGA: from the fastest to the smallest", Proceedings of CHES 2005, pp. 427-440, LNCS 3659, Springer, 2005

[15] Jacquin L., Roca V., „Parallel arithmetic encryption for high-bandwidth communications on multicore/GPGPU platforms", Proceedings of the 4th International Workshop on Parallel and Symbolic Computation, 2010, Grenoble

[16] Kakarountas A., Michail H., „Implementation of a Cryptographic Co-processor", 6th WSEAS International Conference on Information Security and Privacy, Tenerife, 2007

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

[18] Lee R.B., Chen Y.Y., „Processor Accelerator for AES" 2010 IEEE 8th Symposium on Application Specific Processors (SASP), 2010

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

[20] Manavski Svetlin, "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

[21] NVIDIA GeForce 8800GT Characteristics. Hardware Heaven Forum http://www.hardwareheaven.com/reviews/8800GTs/whatis.php 17.05.2010

[22] Parhi K., Zhang X., "An efficient 21.56 Gbps AES implementation on FPGA," in Signals, Systems and Computers. Conference Record of the Thirty-Eighth Asilomar Conference, Nov. 2004, pp. 465–470.

[23] Urmas Rosenberg, using Graphic Processing Unit in Block Cipher Calculations, Master's Thesis, http://uraes.sasslantis.ee/

[24] Standaert F., Rouvroy G., Legat. J, „Efficient Implementation of Rijndael Encryption in Reconfigurable Hardware: Improvements and Design Tradeoffs", CHES 2003, LNCS Vol. 2779

[25] Ştefănescu G., Arhitectura sistemelor de clacul, curs, 2006 http://funinf.cs.unibuc.ro/~gheorghe/curs/arhCalc/lec/l02four.pdf

[26] Takeshi Y., „AES Encryption and Decryption on the GPU", GPU Gems 3, 2007

[27] Tirtea R., Deconinck G.," Specifications overview for counter mode of operation. Security aspects in case of faults." Electrotechnical Conference, 2004. MELECON 2004. Proceedings of the 12th IEEE Mediterranean, pag. 769–773 Vol.2, 2004.

[28] Tomoiagă R.D., Stratulat M., „AES Performance Analysis on Several Programming Environments, Operating Systems or Computational Platforms", The Fifth International Conference on Systems and Networks Communications ICSNC 2010, Nice, 2010

[29] Tomoiagă R.D., Stratulat M., „AES ON GPU USING CUDA ", Proceedings of European Conference for the Applied Mathematics and Informatics EURO-SIAM, Athens, 2010

[30] Tomoiagă R.D., Stratulat M., AES ALGORITHM ADAPTED ON GPU USING CUDA FOR SMALL DATA AND LARGE DATA VOLUME ENCRYPTION, International Journal of Applied Mathematics and Informatics, 2011

[31] Tomoiagă R.D., ACCELERATING SOLUTION PROPOSAL OF AES USING A GRAPHIC PROCESSOR, ISSN: 2069-8216, 2011

[32] Yeom H., Cho Y., Yung M., "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.

[33] Zajac P., Grosec O., „Searching for a different AES-Class MixColumns operation", Proceedings of the 6th WSEAS international Conference on Applied Computer Science, Tenerife, 2006.

[34] https://github.com/cbguder/aes-on-cuda

[35] http://cboard.cprogramming.com/tech-board/98777-aes-subbytes-step.html