CUDA COMPATIBLE GPU AS AN EFFICIENT HARDWARE ACCELERATOR FOR AES CRYPTOGRAPHY

Svetlin A. Manavski

ITCIS, Sofia, Bulgaria, svetlin.a@manavski.com

ABSTRACT

This paper presents a study of the efficiency in applying modern Graphics Processing Units in symmetric key cryptographic solutions. It describes both traditional style approaches based on the OpenGL graphics API and new ones based on the recent technology trends of major hardware vendors. It presents an efficient implementation of the Advanced Encryption Standard (AES) algorithm in the novel CUDA platform by Nvidia. AES is currently the most widely adopted modern symmetric key encryption standard. The performance of the new fastest GPU solution is compared with those of the reference sequential implementations running on an Intel Pentium IV 3.0 GHz CPU. Unlike previous research in this field, the results of this effort show for the first time the GPU can perform as an efficient cryptographic accelerator. The developed solutions run up to 20 times faster than OpenSSL and in the same range of performance of existing hardware based implementations.

Index Terms— Cryptography, Data security, Graphics

1. INTRODUCTION

Graphics Processing Units have been the subject of extensive research during the last few years and have been successfully applied to general purpose applications out of the graphical domain. The GPUs are designed to perform hundreds of billions of floating point operations per second on their large bandwidth on-board memory. Until recently, the only available environments to program the GPU were the OpenGL and Microsoft Direct3D Graphics APIs. The major drawbacks of using either of these for general purpose computing are related to the graphics nature of both environments. Mapping a general purpose problem to the graphical domain is not always a simple task and the final performance of the solution is dramatically dependent on the chosen mapping.

The demand of efficient cryptographic solutions has been continuously growing in the last decade as a consequence of using the Internet in critical areas like business, government and healthcare. So many hardware based SSL acceleration solutions have been studied and proposed both in the research and the industrial field. However, efficient GPU based solutions of the major cryptography algorithms were not actually feasible till now. The main reasons are the following:

- missing internal integer representations of the data and relative bitwise operations
- limited programming model of the Graphics API, missing fundamental operations like "scatter"
- restricted memory model of the graphics API
- overhead of the fixed graphics pipeline

Hardware GPU vendors are currently working on relaxing many of these restrictions. Recent announcements of technologies like CTM [10] by AMD and CUDA [9] by NVidia proved their effort to extend both the programming and memory models, specifically to General Purpose Computation on Graphics Processing Units or GPGPU [6]. Furthermore, these environments provide direct access to the underlying hardware. This approach enables a new vision of the GPU as an efficient cryptographic acceleration unit. Here is presented an effort in developing novel approaches in implementation of symmetric key algorithms on the GPU, in order to prove its general applicability as an efficient cryptographic co-processor. The main focus is on the Advanced Encryption Standard or AES [2] because it is a widely adopted symmetric key cipher standard.

2. RELATED WORKS

This section provides some references to previous work related to developing cryptography solutions both in dedicated hardware and on the GPU.

2.1. Hardware based solutions

Since 2001, when AES was accepted as a FIPS standard [2], a lot of hardware implementations, using ASIC and FPGA devices, have been proposed. C. Su et al. [14], J. Wolkerstorfer et al.[15], Hodjat et al.[16], using particular S-box optimizations, rather than pipelining, or combination of S-box and MixColumns, called T-box, provided throughput rates from 1 to 70 Gbit/s.

The presented CUDA-AES implementation, tested on a NVidia GeForce 8800 GTX, performs a peak throughput of 8.28 Gbit/s. This result, interestingly obtained with commodity hardware, is in the same range of the above hardware based solutions. Furthermore, the implementation could linearly improve its throughput exploiting the parallelism of several GPU devices, when available on the same machine.

2.2. Cryptography on the GPU

The only significant attempt to implement symmetric ciphers on the GPU was made by D.Cook et al.[8][13]. Their effort was based on mapping the AES cipher to the standard fixed graphics pipeline using OpenGL. The results of this paper were both limited by the performance of the available hardware, and the limited functionality achievable without exploiting the programmable GPU shaders. Their implementation performed up to 1.53 Mbits/s on NVidia GeForce 3, which was too far from the 64 Mbits/s they reached on the CPU Pentium IV 1.8 GHz. There are not any further known successful attempts in competing with the modern CPU in optimized solutions of largely used cryptography standard algorithms. While the traditional graphics pipeline architecture limits the potential performance of the block cipher key systems like AES, it makes practically unsuccessful any approach in implementing algorithms heavily depending upon *scatter* operations like RSA.

3. THE GRAPHICS PROGRAMMING UNIT

3.1. The traditional GPU programming model

This section is a brief overview of the traditional OpenGL graphics pipeline. It is now present in all the existing hardware up to the recent NVidia G80 unified architecture platform.

In the graphics pipeline, there are three kinds of processor units. The *rasterizer* is the only fixed functionality processing stage while both the *vertex* and the *fragment* processors are programmable in recent architectures. The vertex processors allow for a program to be applied to each vertex in the object, then each group of three vertices is used to compute a triangle, and from this triangle a stream of fragments is generated. Fragment processors, which transform fragments, are able to fetch additional data, so these are capable of *gather,* but the output address of the fragment is always determined before the fragment is processed. Therefore, these units are not natively capable of *scatter*. First of all, the input data mapped to the graphical domain must be transferred to the texture memory and the vertex buffer of the GPU. The texture memory is the basic memory unit where the whole set of the input data is usually stored. For the purpose of the GPGPU applications most of the operations needed to solve the problem are actually performed in the fragment processing unit. There are two main reasons for that. The first is there are usually more fragment processors in the typical GPU. The second one is that the fragment processing is near the end of the pipeline so its output is easier to get straightforward to the memory either as a final result or as an intermediate input to the next processing stage.

3.2. CUDA and the new GPGPU programming model

The two major GPU vendors, NVidia and AMD recently announced their new developing platforms, respectively CUDA [9] and CTM [10]. Unlike previous programming models, these are proprietary approaches, designed to natively access the graphics hardware of the only specific vendor. So these platforms are incompatible between them. While the first, CUDA is announced to be an extension to the C programming language, the other approach, CTM is a virtual machine running proprietary assembler code. However, both platforms provide completely new programming models, aiming to relax the important restrictions on the GPGPU applications, imposed by the traditional pipeline and the relative graphics-oriented programming interfaces. In order to design the efficient AES solution, Nvidia GeForce 8800 was selected and its CUDA development platform. It is the first available GPU on the market capable of performing natively programmable logical bitwise operations on internal integer representation of the data.

Programming through CUDA, the GPU can be seen as a device capable of executing a very high number of threads in parallel. A kernel of code can be launched on different threads and on different blocks of threads, called *grid*. Threads from the same block share data through a fast shared on-chip memory and can be synchronized through apposite synchronization points. In CUDA the programmer has access to the device's DRAM and on-chip memory through a number of memory spaces. The choice of the memory to use depends on different factors as speed, amount of memory needed and operations to do on stored data. This new architecture allows to access memory in a really general way so both *scatter* and *gather* capabilities are available. But the overall performance of the applications dramatically depends on the strategy of use of the memory model mentioned above.

4. BLOCK CIPHER ALGORITHMS AND AES BACKGROUND

A block cipher algorithm is a symmetric key cipher operation on a fixed number of bits, named *block*. It takes a n-bits block of plain text as input, and outputs a corresponding n-bits block of ciphered text. The transformation depends on a second input which is the secret key. Decryption is similar. Messages longer than n bits are divided into n-bits blocks, padding the last if necessary, each ciphered/deciphered separately. AES is the most recent symmetric key algorithm standard used in several security protocols. It was announced by National Institute of Standards and Technology (NIST) as U.S. FIPS PUB 197 [2] in November 26, 2001, and it became effective as a standard in May 26, 2002. The cipher is based on the Rijndael algorithm developed by two Belgian cryptographers, Joan Daemen and Vincent Rijmen, and it accepts a fixed block size of 128 bits and a key size of 128, 192 or 256 bits. AES encryption consists of a variable number of rounds, depending on the key length, operating on a block of 4x4 bytes, termed *state*. Each round is a sequence of four stages: *AddRoundKey, SubBytes, ShiftRows, MixColumns.* Before starting the cipher the key bits must be expanded using a precise key schedule. For complete details about AES refer to [1] and [2]. The CUDA-AES implementation is based on a combination of the round stages, which allows a very fast execution on processors with word length of 32 bits, as described in [1]. If we call *a* the round input, expressing one column of the round output *e* in terms of bytes of *a* we have:

$$e_j = T_0\lfloor a_{0,j}\rfloor \oplus T_1\lfloor a_{1,j+1}\rfloor \oplus T_2\lfloor a_{2,j+2}\rfloor \oplus T_3\lfloor a_{3,j+3}\rfloor \oplus k_j \quad (1)$$

where *T[ ]* is a look-up table, $\oplus$ means XOR and k_j is one column of the stage key. This solution takes only 4 look-ups and 4 XORs per column per round.

5. THE IMPLEMENTATIONS OF AES

This section describes two completely different approaches to address the AES problem on the graphics hardware. Both implementations do not focus on computing the key scheduling but on the main cryptography algorithm. Key scheduling is a procedure of limited number of operations that would not exploit the real GPU processing power. For this reason, it was implemented on the CPU leaving to the graphics card the more computationally intensive tasks.

5.1. The traditional style OpenGL based implementation

A framework of C++ classes was developed, which setups the OpenGL environment and allows to program transparently the fragment processors in several shader programming languages. The same algorithms was developed and tested in GLSL [4], Cg [3] and ARB Assembly [4]. The native internal representation of the data in the OpenGL memory model is the floating point format, which is unsuitable for the purpose of cryptography. Thus every single byte of the input was mapped in an fp16 number (16 bits floating point number) of the texture memory. This allows us to have enough precision during operations on these numbers given that the useful values are limited in the range 0 to 255. The best found OpenGL implementation runs only one AES round in a single kernel call. Each GPU thread computes four components of the AES internal state and takes only four values of the input. Each subsequent 16 entries of the input are mapped to four neighbouring texels in the respective R, G, B and A values. In this approach, as in previous attempts [13], the CPU is still used to subsequently call the AES rounds to setup the output of each round as the input of the next one, although the actual load to make the algorithm computations is all on the GPU. Another major problem in implementing cryptography on traditional GPU architectures is the unavailability of the bitwise logical operations in the programmable shaders. The first option was to enable the native XOR operation while storing data to the output frame buffer, but this solution, faster in terms of time needed to perform the XOR operation, requires to split the single AES round in further four shaders. This because it is needed to be able to compute the T-boxes separately before storing the outputs to the framebuffer with XOR operation enabled on it. The resulting solution would be actually conceptually similar to the work of D.Cook et al [13], which concludes the CPU keeps loaded during all the execution. Alternatively the implementation adopts a square look-up XOR table of 256x256 entries to implement 8 bit xor bitwise operations computing the function on four values at a time. To further benefit of the limited cache size on the GPU, the three S-boxes needed to encrypt (and the five S-boxes needed to decrypt) are included in the free colour components of the same texels, used to map the XOR look-up table. The results of running the most optimized implementation on the ATI X1900 platform, which results to be the ARB Assembly one, are compared in Section 6 to the native CUDA implementation on NVidia G80.

5.2. Efficient CUDA based implementation on G80

According to the programming model of CUDA, the developer has a lot of freedom on, for example, the choice and the type of the memory spaces, the data shared among threads, the dimension of the block of threads, and so on. But this approach leads to a considerable complexity in designing an efficient application compared to the common programming model of the CPU.

The CUDA-AES implementation fully benefits of the most optimized known AES techniques, designed for the 32 bit processors as described in Section 4. Given the flexibility of the memory model, it is possible to efficiently use the four T-look-up tables each one containing 256 entries of 32 bits each. Unlike previous GPU hardware, G80 is a scalar processor, so there is no need to combine instructions in vector operations, in order to get the full processing power. Furthermore, the native availability of the 32 bit logical XOR operation on G80 speeds up by orders of magnitude the execution of that fundamental operation of the cryptographic theory. Thus, a single AES round on a state can be done with 16 table look-ups and 16 32-bit exclusive-or operations (four iterations of the equation (1)).

First of all, the input data and the expanded key are stored in the GPU global memory space. As the final results show, moving the data to and back from the device memory may become the slowest operation when doing cryptography on the GPU. It is due to the bandwidth of the PCIExpress interface which is only about 3,2 GB/s compared to the 50 GB/s of the onboard memory of the GeForce 8800 graphics card. The pre-computed T-boxes are pre-loaded in the specific constant memory of the device. The constant memory is cached and the look-up table fully matches the size of the cache. The input data is then divided in chunks of 1024 bytes which are encrypted or decrypted completely in parallel. One CUDA block of threads is responsible for computing one chunk of the input. The block is composed of 256 GPU threads. An exhaustive research has indicated that size to lead the fastest implementation. When running the kernel code on the device, performance improvement can be observed as the number of blocks increases. So bigger input sizes lead to better performance. Every GPU thread of the block computes four output bytes in each AES round, so four threads encrypt/decrypt the whole input state. Threads from the same block need to share and to frequently access information of the AES expanded key. For this reason it is loaded in shared memory together with the portion of input processed by that block. More precisely, two arrays of 1KB shared memory are used for the input, reading data from the first and saving results of each AES round to the second one. Then the arrays are swapped for the successive round. This strategy allows to complete the encryption of the input chunk without exiting the kernel. So it is done without using the CPU to manage an external for loop to launch sequentially all the AES rounds. At the end of the computation, the resulted output data is written again in the global device memory and then returned to the CPU.

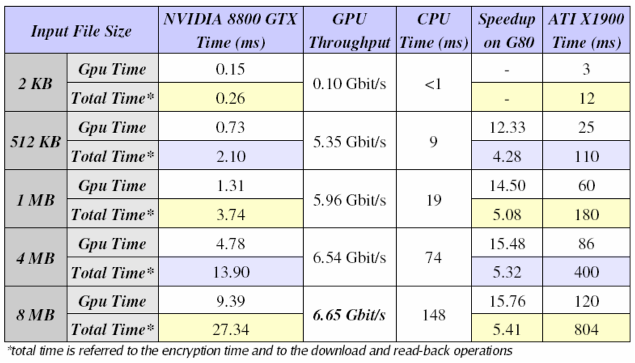
The implementation described above delivered a peak performance of 5,72 Gbit/s in encrypting an 8MB input with AES-256. Some further improvements were then applied, which lead to the final peak performance of 6,65 Gbit/s in the same conditions of the strongest AES-256. The direction was to research strategies to reduce the usage of the shared memory. The GeForce 8800 shared memory is divided into equally-sized memory modules, called banks, which can be accessed simultaneously. But if two addresses of a memory request fall in the same module, there is a bank conflict and the accesses have to be serialized. This behavior may slow down the performance because of the access pattern to the expanded AES key. Furthermore in order to benefit of the shared memory the key must be loaded there at the beginning of the execution of each block of threads. An alternative is to store the expanded key once in a texture memory which is cached as well. A random texture memory access is available directly by all the GPU threads and by the CPU to store the key. The limited size of the expanded key ensures it is always completely stored in the texture cache.

Furthermore, must be noticed this approach allows for the textures of third parties applications to be directly encrypted on the GPU. In OpenGL this is not possible because it does not allow to access the different bytes that compose a single texture value. But in CUDA, unlike OpenGL, it is possible to consider the single fp16/fp32 number as a sequence of separated bytes which may be encrypted. To encode textures this way does not even require to move data to and from the GPU and the host memory.

6. PERFORMANCE ANALYSIS

This section compares the performances of the final implementations of the AES algorithm (Figures 1, 2 and 3). The tests were conducted using an Intel Pentium IV, 3.0 GHz CPU, an NVIDIA GeForce 8800 GTX and an ATI X1900. As it can be seen, a comparison has been made between GPUs and a CPU implementation based on the OpenSSL library. Both the internal GPU elaboration time is shown and the total time, that includes the time needed for the download and read-back operations (that means copying data from the host memory to the GPU device and back). The CUDA-AES implementation on NVIDIA card is faster than the CPU on every input size with reference both to the internal GPU and to the total time. A peak throughput rate of 8.28 Gbit/s is achieved with an input size of 8MB. In that case the GPU is 19.60 times faster than the CPU.

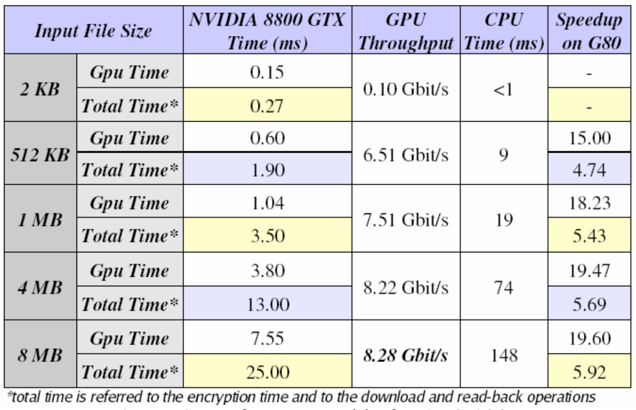
The ATI X1900 implementation is in OpenGL and ARB Assembly shaders. In this case there are not any of the bitwise logical operations so it is based on a look-up table with a precompiled XOR. Despite it, when the input size begins to increase the GPU becomes competitive. Unfortunately, this result is vanished by the enormous amount of time spent for download and read-back operations. M. Houston in [11] proves that ATI X1900 is effectively affected by poor performances for this kind of operations, especially for the read-back one.



Input File Size		NVIDIA 8800 GTX Time (ms)	GPU Throughput	CPU Time (ms)	Speedup on G80	ATI X1900 Time (ms)
2 KB	Gpu Time	0.15	0.10 Gbit/s	<1	-	3
	Total Time*	0.26			-	12
512 KB	Gpu Time	0.73	5.35 Gbit/s	9	12.33	25
	Total Time*	2.10			4.28	110
1 MB	Gpu Time	1.31	5.96 Gbit/s	19	14.50	60
	Total Time*	3.74			5.08	180
4 MB	Gpu Time	4.78	6.54 Gbit/s	74	15.48	86
	Total Time*	13.90			5.32	400
8 MB	Gpu Time	9.39	*6.65 Gbit/s*	148	15.76	120
	Total Time*	27.34			5.41	804

*total time is referred to the encryption time and to the download and read-back operations

Figure 1. Performance table for AES 256. These results are referred to the encryption phase, but decryption performance is nearly the same.



Input File Size		NVIDIA 8800 GTX Time (ms)	GPU Throughput	CPU Time (ms)	Speedup on G80
2 KB	Gpu Time	0.15	0.10 Gbit/s	<1	-
	Total Time*	0.27			-
512 KB	Gpu Time	0.60	6.51 Gbit/s	9	15.00
	Total Time*	1.90			4.74
1 MB	Gpu Time	1.04	7.51 Gbit/s	19	18.23
	Total Time*	3.50			5.43
4 MB	Gpu Time	3.80	8.22 Gbit/s	74	19.47
	Total Time*	13.00			5.69
8 MB	Gpu Time	7.55	*8.28 Gbit/s*	148	19.60
	Total Time*	25.00			5.92

*total time is referred to the encryption time and to the download and read-back operations

Figure 2. Performance table for AES 128.

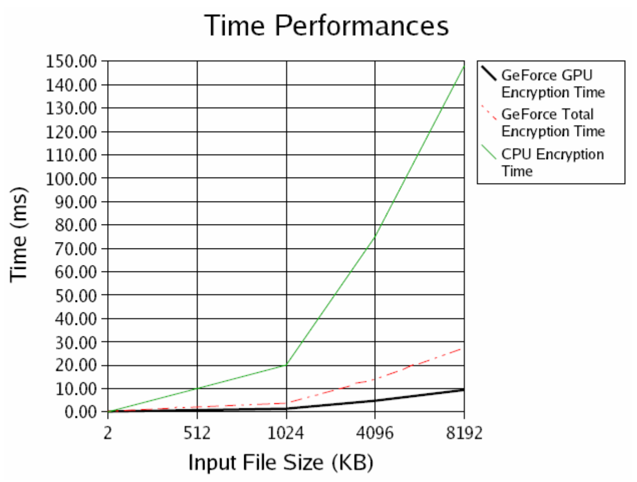


Figure 3. Performance Chart for AES 256

7. CONCLUSIONS AND FUTURE WORK

The presented the most efficient currently known approaches in encryption and decryption of messages with AES on programmable graphics processing units. While the study suggested that the traditional graphics hardware architectures could now be compared with optimized sequential solutions on the CPU, it definitely proved that the novel unified architectures like G80 are needed to significantly outperform it in the field of symmetric cryptography. Unlike previous related work, for the first time a GPU implementation of AES performs the encryption and decryption of the input data without the CPU to keep busy in the meantime. So, this effort has to be considered as the proof that the modern unified GPU architecture can perform as an efficient cryptographic acceleration board. Future work will include efficient implementations of other common symmetric algorithms. GPU implementations of hashing and public key algorithms may also be implemented, in order to create a complete cryptographic framework accelerated by the GPU.

8. REFERENCES

[1] J. Daemen, V. Rijmen, "AES Proposal: Rijndael". Original AES Submission to NIST, 1999.

[2] National Institute of Standards and Technology (NIST), "FIPS 197: Advanced Encryption Standard (AES)", 2001.

[3] R. Fernando, M. Kilgard, "Cg Tutorial, The: The Definitive Guide to Programmable Real-Time Graphics", ISBN 0321194969, Addison-Wesley, New York, 2003.

[4] OpenGL Architecture Review Board, M. Woo, J. Neider, T. Davis, D. Shreiner, "The OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 2", 5th edition. ISBN 0321335732, Addison-Wesley, New York, 2005.

[5] N. Sklavos, O. Koufopavlou, "Architectures and VLSI Implementations of the AES-Proposal Rijndael", IEEE Transactions on Computers, Vol. 51, Issue 12, pp. 1454-1459, 2002.

[6] General Purpose Computation Using Graphics Hardware, http://www.gpgpu.org.

[7] OpenSSL Open Source Project, http://www.openssl.org.

[8] D. L. Cook, A. D. Keroymytis, "Cryptographics: Exploiting Graphics Cards for Security", Advancements in Information Security series, Springer, 2006.

[9] NVidia CUDA , http://developer.NVidia.com/object/CUDA.html.

[10] AMD CTM, http://ati.amd.com/companyinfo/researcher/documents.html.

[11] M. Houston, "Understanding GPUs through Benchmarking". Supercomputing Conference, Tampa FL, 2006.

[12] I. Buck, A. Lefohn, P. McCormick, J. Owens, T. Purcell, R. Strodka, "General Purpose Computation on Graphics Hardware". IEEE Visualization 05, Minneapolis, USA, 2005.

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

[14] C. Su, T. Lin, C. Huang, C. Wu, "A High-Throughput Low-Cost AES processor". IEEE Communications Magazine, vol. 41, no. 12, pp. 86-91, 2003.

[15] J. Wolkerstorfer, E. Oswald, M. Lamberger, "An ASIC Implementation of the AES Sboxes". RSA Conference 02, San Jose CA, 2002.

[16] A. Hodjat, I. Verbauwhede, "Minimum Area Cost for a 30 to 70 Gbits/s AES Processor". IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2004), Eerging Trends in VLSI System Design, pp 83-88, 2004.