AES ON GPU USING CUDA

TOMOIAGĂ RADU DANIEL, STRATULAT MIRCEA
Faculty of Automatics and Computer Science
“Politehnica” University of Timișoara
Timișoara, V. Parvan 2, 1900
ROMANIA
radugam@mailcity.com, mircea.stratulat@cs.upt.ro

Abstract: - In the pursuit of gaining more performance for cryptographic primitives, the authors 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 for data stored in local memory of the GPU the CUDA implementation can offer speedups of almost 40 times in comparison to the CPU.

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

1 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 integer and byte operations, GPU had become an eligible competitor for cryptographic coprocessors.

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 [7], 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 medium data rate 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 legacy processor [2].

Luken speaks about encrypting with AES and DES using GPU hardware acceleration [3]. 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 the CPU.

Manavski tested CUDA compatibility in hardware acceleration for AES on NVIDIA graphic cards [4]. 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.

2 G80- Architecture and Specifications

NVIDIA 8800 GT is based on G80 Core, which is the first 65nm NVIDIA Kernel. It 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 are located in global memory [1].
3 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 it can be rewritten to be executed on GPU as independent thread.

If a kernel is invoked it will be executed in the 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 from 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.

All threads within a block will be executed on one multiprocessor. This allows 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].

4 AES Implementation and results.

When adapting AES on GPU, and more specifically on 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 [www41] 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.

\[ K_i = E(K, \text{Nonce} \Vert i), \text{for } i = 1,...,k \text{ and } C_i := P \oplus K_i \] (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 faster memory read/writes. Access to global memory is done in the initial phase, before processing data. The data is moved into the shared memory where it can be accessed faster. If every thread loads data in the shared memory from 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 [2].

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 [2]. Although in the tests done in [8] 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 S-BOX 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[3].

In [2] 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 easily chose to implement AES with lookup tables as GeForce 8800 GT/G80, is a scalar processor and it is not necessary to 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 [5] an [6].

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

Input sizes are the same to the ones used in [5] an [6]. 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.

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

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: \( \text{buffer} = \text{algorithm}\_\text{encrypt}(\text{buffer}) \). The chosen iterations number is 1.000.000 like the tests we did in [5] an [6].

The obtained results were compared with the results that we obtained using the algorithm developed by [8]. In table 1 are three values presented. The first represents the AES algorithm implementation done in [8], the second
one represents the time used by the GPU to encrypt data using AES on GPU based on S-BOX 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 S-BOX and lookup tables.

The algorithm that uses S-BOX 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 from 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 [5] an [6] 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 [8] and the other one is slower. Times are not so different and performance ratio between these algorithms can be observed in figure 5.

Using the results obtained for this platform in [5] an [6] and analyzing figure 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 figure 6 are obtained taken from [6]. 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.

5 Mathematical Complexity of AES

AES algorithm handles all bytes as Finite Field Elements. Finite Field Elements can be added and multiplied, but these operations differ from the ones used on numbers [10].

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 $\cdot$, and in polynomial representation it corresponds with the multiplication of polynomials modulo irreducible polynomial of 8 degree. In case of AES this polynomial is $f(x) = x^8 + x^4 + x^3 + x + 1$ [10].

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 $(x^5 + x^4 + x^2 + x + 1) \oplus (x^7 + x^6 + x^4 + x^3 + x^2 + x + x^6 + x^4 + x^2 + x + 1) = x^{13} + x^{11} + x^9 + x^8 + x^7 + x^6 + x^5 + x^3 + x^2 + x + x^6 + x^4 + x^2 + x + 1$.

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” $\cdot$ „83” = „C1” because $(x^5 + x^4 + x^2 + x + 1) \cdot (x^7 + x^6 + x^4 + x^3 + x^2 + x + x^6 + x^4 + x^2 + x + 1) = x^{13} + x^{11} + x^9 + x^8 + x^7 + x^6 + x^5 + x^3 + x^2 + x + x^6 + x^4 + x^2 + x + 1$ modulo $f(x) = x^2 + x^6 + 1$ [10].

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 $a(x) \cdot b(x) \bmod f(x) = 1$, then the following can be concluded $b^{-1}(x) = a(x) \bmod f(x)$ and $a(x) \cdot (b(x) + c(x)) = a(x) \cdot b(x) + a(x) \cdot c(x)$ [10].
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.

---

**Table 1. CPU Comparison**

<table>
<thead>
<tr>
<th>Processor</th>
<th>AES with SBOX</th>
<th>AES without SBOX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel (R) Core(TM)2 Duo CPU E6750 @ 2.66GHz</td>
<td>0.00118259</td>
<td>0.00115233</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.00121234</td>
</tr>
</tbody>
</table>

**Table 2. CPU results obtained in [6]**

<table>
<thead>
<tr>
<th>PC1</th>
<th>C#</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.16534</td>
<td>0.15581</td>
<td>0.15632</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PC2</th>
<th>C#</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2118</td>
<td>0.40544</td>
<td>0.40544</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PC3</th>
<th>C#</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.03266</td>
<td>0.03196</td>
<td>0.20374</td>
</tr>
</tbody>
</table>

---

**Test Platform Description**

<table>
<thead>
<tr>
<th>Processor</th>
<th>Intel (R) Core(TM)2 Duo CPU E6750 @ 2.66GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Name</td>
<td>Conroe</td>
</tr>
<tr>
<td>LGA775,</td>
<td></td>
</tr>
<tr>
<td>L1 I-Cache 32 KB L1 D-Cache 32 KB L2 Cache 496 KB L2 Cache Speed 2666.69 MHz</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memory (4GB)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Slot 1</td>
<td>Manufacturer Kingston Capacity 2048 MBYTEs</td>
</tr>
<tr>
<td>DDR2 SDRAM Speed DDR2-666 (333 MHz) Data Width 64 bits</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hard disk</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Slot 3</td>
<td>Manufacturer Kingston Capacity 2048 MBYTEs</td>
</tr>
<tr>
<td>DDR2 SDRAM Speed DDR2-666 (333 MHz) Data Width 64 bits</td>
<td></td>
</tr>
</tbody>
</table>

| Video Adapter | NVIDIA GeForce 8800 GT 512 MBYTEs |

---

**Fig. 5. Encrypting memory data.**

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 [10]:

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 \( b(x) = d(x) \) and the corresponding notation is \( d(x) = a(x) \otimes b(x) \). \( d(x) = d_6x^6 + d_5x^5 + d_4x^4 + d_3x^3 + d_2x^2 + d_1x + d_0 \), where [10]:

- \( d_0 = (a_0 \oplus b_0) \oplus (a_3 \oplus b_1) \oplus (a_2 \oplus b_2) \oplus (a_1 \oplus b_3) \)
- \( d_1 = (a_1 \oplus b_0) \oplus (a_0 \oplus b_1) \oplus (a_3 \oplus b_2) \oplus (a_2 \oplus b_3) \)
- \( d_2 = (a_2 \oplus b_0) \oplus (a_1 \oplus b_1) \oplus (a_0 \oplus b_2) \oplus (a_3 \oplus b_3) \)
- \( d_3 = (a_3 \oplus b_0) \oplus (a_2 \oplus b_1) \oplus (a_1 \oplus b_2) \oplus (a_0 \oplus b_3) \)

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.

---

**Fig. 6. GPU-CPU Comparison**

---

Applied Mathematics and Informatics

6 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 [5] and [6], results obtained after running the tests processors offer results that vary according to the file size, algorithm tested, memory capacity, programming language or operating system.

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 [8]. 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 the first phase we tested the algorithm implemented in [8] and the results were compared with the ones obtained in [5] an [6].
- 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 an own implementation of AES on GPU(CUDA).
- We propose, implemented and tested two solutions: in one we used S-BOX 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.

As a future research 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.

References: