Block Ciphers on GPU: Integration and Evaluation of the improvement

Eisa AL SHAMSI
ENSIMAG, Grenoble (France)
eisa.alshamsi@ensimag.imag.fr

Mohamed AL ALI
ENSIMAG, Grenoble (France)
mohamed.al-ali@ensimag.imag.fr

ABSTRACT

In this report we implement three symmetric block ciphers (AES, Serpent and Twofish) on CUDA, then measure and analyze the improvement in performance achieved. The idea is to run the encryption/decryption function in parallel on a high number of input blocks using CUDA threads. Then measure the speed up gained from this implementation and compare between the three block ciphers.

Keywords

GPU, CUDA, block-ciphers, AES, SERPENT, TWOFISH.

1. INTRODUCTION

Data Security is a very important concept in today's life. Governments, corporations and even individuals share the need to protect sensitive data by using the available algorithms used in encrypting data. As the size of data increases, rises the need for more efficient algorithms to encrypt and decrypt data. Block-ciphers are the most common ciphers used today, especially the AES. These ciphers have the advantage of being highly parallelizable if certain operation modes are used.

Recently there have much progress in the field of Graphics Processing Units (GPUs), and since these units have a highly parallel structure, many complex algorithms were redesigned to be more efficient and make use of the parallel structure of the GPU. A GPU has many microprocessors that can be utilized in solving complex algorithms with higher performance than a single core processor. This approach also helps in offloading the calculations off the general purpose CPU leaving it free to do other tasks. Programming on GPUs is becoming easier as recently *NVidia* have been releasing cards that support the CUDA API that extends the C language to give it access to the GPU, allowing C functions to run on the GPU stream processors.

From combining the need for more efficient cryptographic approaches and the improvement in the GPU field, rises the idea of adapting block cipher algorithms to run on GPUs to improve their performance. In this report we chose three block ciphers (*Rijndael*, *Serpent* and *Twofish*) and implemented them on the GPU to monitor the improvement in performance we get in those algorithms and compare between them.

1.1 Organization of this report

This report is divided into seven sections. The first section is a brief introduction that gives the motivation behind our work to implement block ciphers on GPU. Section two introduces the concept of block ciphers and briefly explains some details about the ciphers used in the project. The third section describes GPUs and their programming model. Section four describes the implementation done during this project. The fifth section shows the results obtained and the comments on these results. Section six concludes the work done on the project, and the last section proposes future work that could be done to improve this project.

2. Block Ciphers Background

Block ciphers are symmetric key ciphers that take a block consisting of a fixed number of bits (a block) of plain text as an input, and produce a block of the same number of bits of ciphered text as output. A secret key is also taken as input to make the transformation on the input plain text. This is done in the encryption step. Decryption is the opposite of this process. In 2001 the National Institute of Standards and Technology (NIST) announced the AES contest of choosing a standard algorithm for block chippers. The *Rijndael* algorithm was chosen, and the two runners up are the *Serpent* algorithm and the *Twofish* algorithm.

2.1 AES

AES was adapted from the *Rijndael* algorithm submitted by the two Belgian cryptographers *Joan Daemen* and *Vincent Rijmen*. The AES uses a block size of 128 bits, a key of sizes 128, 192 and 256 bits respectively. The number of rounds done on the plain text to produce the cipher text depends on the key size. There are four stages done in every round which are: *AddRoundKey*, *SubBytes*, *ShiftRows* and *MixColumns*.

The AES starts by expanding the key using the *Rijndael* key schedule. Then an initial round of *AddRoundKey* is done on the output. Then a number of rounds are done on the plaintext depending on the key size. The last round is done without the *MixColumns* stage [1].

2.2 Serpent

Serpent is a symmetric key block cipher which was designed by *Ross Anderson*, *Eli Biham* and *Lars Knudsen*. It was ranked the second after *Rijndael*. *Serpent* cipher has input/output block with size of 128 bits. The idea of the *Serpent* cipher algorithm it to process the input plain text through 32 rounds. Each rounds will adds the 128 bits key, then pass the text into 32 S-boxes, which has a width of 4 bits. After that, a linear transformation will be applied so that it will make each output of one round as the inputs of a number of S-boxes in the next round.

Therefore, any change in the input (plain text) will extremely change the result (cipher text) of encryption. This is an advantage of using Serpent cipher which makes not only linear but also differential attacks much harder [2].

2.3 Twofish

The *Twofish* cipher is a block cipher which uses the same Feistel Network structure of the DES. The process uses 16 rounds to transform the plain text into ciphered text. The Twofish cipher uses pre-computed key-dependant S-boxes and a complex key schedule. It was designed by *Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson*. It ranked third in the AES competition [3].

3. GPU Background

In the few past years, the field of Graphical Processing Units has seen great improvements in term of the processing power, especially parallel processing. Some GPUs even have more transistors than a new quad-core CPU would have. Moreover, GPUs are even progressing much faster. This is due to the fact that they are designed around intensive computing and highly parallel computations, which is the case in graphics rendering.

The concept of using the computational power of such GPUs in applications other than image processing has been an interesting subject that many engineers around the world are exploring with promising results in terms of performance. The two major GPU manufacturers NVidia and AMD have both released development platforms that allow access to the GPU hardware, CUDA and CTM respectively. This have opened the doors for many algorithms to be ported to run on GPUs in search of better performance than on a CPU, but a major drawback is that each platform is only compatible with the hardware of its manufacturer.

In this project we are implementing the three block ciphers on CUDA, which is an extension to the C programming language that allows C programs to run on the GPU instead of using the processor. When programming with CUDA, the programmer is able to launch a very high number of small programs at the same time, or as we did in this project launch a very high number of threads in parallel. This allows the programmer to launch the same kernel of code over different blocks of data in parallel. This requires sets of data that are independent of each other. *Figure 1* below shows the CUDA programming model in terms of the Grid, Block and Thread hierarchy.

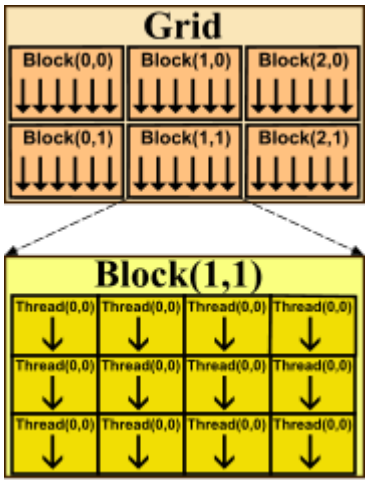

Figure 1. CUDA Grid of Thread Blocks [4]

CUDA imposes some limitations, these include the fact that no recursive calls are allowed in a CUDA function. In addition, the bandwidth of the bus and the latency between the CPU and the GPU could impose a bottleneck in applications.

4. Implementation on GPU

One of the objectives in this project is to implement the three ciphers (*AES*, *Serpent* and *TwoFish*) on GPU. The languages that are used to write the encryption algorithm are *C* and *C++* and the starting source codes were available on the internet. The cipher's designers provided their code on their own websites. However, the codes that are used in this project were taken from a different source. The AES cipher code was originally written to be run on CUDA and some modifications was done on it [5]. The Serpent and Twofish cipher codes were standard C versions of those ciphers and the conversion was done by us to make them run on CUDA [6][7]. The reason for using a source code that was created by other than the cipher designers is the simplicity of the code, which makes converting from C code into CUDA code much easier.

There are several steps needed to convert the C code into CUDA code. The CUDA code is somehow similar to the *C* code with considering that the code might be run several times at the same moment.

4.1 Converting from C code to CUDA code

The two codes used were C codes that implement the Serpent cipher and the Twofish cipher. They do encryption and decryption on an input file. The implementation depends on a block size of 128 bits. The size of the input key used was 128 bits. The decryption functions were not considered as they are similar to the encryption functions.

First, variables must be declared and allocated a memory space for them on the graphics card's global memory. These variables represent the input plain text, the output cipher text and the required data for the encryption function which includes the key schedules expanded from the secret key and any other tables needed in the encryption. These variables are declared as device shared variable so they could be shared among threads from the same block to improve the memory access. The computation of key schedule does not require heavy computation and is only done once so it is left to be done on the CPU side. Specifying the size of the data is very important in order to avoid any unhandled exceptions.

Then, we need to add an *offset* variable in the encryption function that manages the index of the input and output arrays in order to let every thread to process a block of data that is dependent on the thread position in the block and the position of the block in the grid. Moreover, the code needs to be modified to remove any recursive calls or any calls to functions outside the device domain.

After that, the required data must be copied from the *Host* memory to the graphics card memory. Copying data from the CPU memory to the GPU memory may present a bottleneck. When we run a bandwidth test that is provided with the CUDA SDK we notice that the host-to-device bandwidth is 1.3 GB/s and the device-to-host bandwidth is 0.8 GB/s when compared to the internal memory bandwidth of the GPU whish is almost 15.5 GB/s.

The next step is to specify the *Grid* and the *Block*s size. The *Grid* consists of *Blocks* (it can represented as a matrix of blocks), where each block can be identified via *block ID*. Each block consists of number of threads. The performance of the code can vary by changing the specification of the *Grid* the *Blocks*. In this project, the CUDA code retrieves the maximum number of threads that can be invoked in the graphics card. If the input size is less than the maximum number of threads, then we take them to be half that size, but if it was greater we give it the maximum value. Then, the number of blocks can be calculated by dividing the plain text size (input) by four and then divide the result by the number of threads. This operation gives a ratio between the input size and both the number of threads and blocks. Without the ratio some errors rises and the encryption process fails.

The next step is to call the encryption function with the parameters that were specified earlier. The idea is that the same function runs on different blocks with different parts of the input text in parallel. This is how we get the improvement in performance. The block size that is sent to every thread is 16 bytes which is the size of one encryption block.

After the encryption is done, the result of the encryption (cipher-text) should be copied from the graphics card memory to the CPU (*Host)* memory.

It is important to mention that the global variables that declared to be used inside the C code cannot be used inside the CUDA code. In addition, it is not possible to call a C functions inside the CUDA.

4.2 Modes of Operation

Block-ciphers can operate using different modes of operation, but not all operation modes are parallelizable. The ECB mode and the counter mode as well as the decryption of the CBC mode could be parallelized. In this project we are using the ECB mode which is very highly parallel as each block of data is independent of the previous or next block.

5. Performance Analysis

5.1 Testing Environment Specifications

The project has been tested under the following specifications:

- **Operating System:** Microsoft Windows Vista Business (64-bit).
- **CPU Type:** Dual Core Intel Wolfdale, 2400 MHz (12 x 200).
    - **L1 Code/Data Cache:** 32 KB per core
    - **L2 Cache:** 3 MB (On-Die, ASC, Full-Speed)
- **System Memory:** 4 GB (DDR2-667 DDR2 SDRAM)
- **Disk Drive:** TOSHIBA MK3252GSX ATA Device (298 GB, IDE)
- **Video Adapter:** nVIDIA Quadro NVS 320M (HP)
    - **Memory size:** 256 MB
    - **Bus width:** 128-bit
    - **GPU Clock (Geometric Domain):** 169 MHz
    - **GPU Clock (Shader Domain):** 338 MHz
    - **Bandwidth:** 3200 MB/s

5.2 Testing results

After running the three ciphers on the CPU and GPU we can notice that we always get an increase in the throughput in the three ciphers for all input sizes. *Table 1*, *table 2* and *table 3* represent the performance results we got from testing AES, Serpent and Twofish respectively. We have included the timings for the internal computation in the GPU as well as the total time including the time taken to copy data to and back from the GPU memory. The timings of all three ciphers on both CPU and GPU are represented in *figure 2*, while *figure 3* represents the throughput.

We can notice that the AES cipher on GPU achieved a maximum speed up of 1.32 when using a file size of 1 MB. At a file size of 32 MB the CPU version is even faster by fragments. The AES cipher is a fast cipher to begin with.

Table 1. AES 128 encryption performance results

Input (MB)	Timings (msec)			Throughput (Mb/s)		Speed Up
	GPU		CPU	GPU	CPU	
	*Total time	GPU time		*Total time		
1	31	26	42	251	190	1.32
4	116	107	135	274	237	1.16
16	466	425	465	274	275	1.00
32	936	858	913	273	280	0.98
64	1883	1728	2319	271	220	1.23

**Total time donates the GPU encryption time with the data transfer to and from the GPU*

The Serpent encryption achieves a maximum speed up of 3.91 which is the highest speed up between the three ciphers. This is because the Serpent cipher is the slowest when running on CPU and running it in parallel have given better results than the other two ciphers which are already fast.

Table 2. Serpent encryption performance results

Input (MB)	Timings (msec)			Throughput (Mb/s)		Speed Up
	GPU		CPU	GPU	CPU	
	*Total time	GPU time		*Total time		
1	33	30	107	240.38	74	3.22
4	117	110	460	271.76	69	3.91
16	456	431	1633	280.47	78	3.58
32	939	892	3268	272.51	78	3.48
64	1889	1784	6691	271.01	76	3.54

**Total time donates the GPU encryption time with the data transfer to and from the GPU*

The Twofish cipher achieves better results than the AES, but less than the improvement of Serpent. The maximum is achieved with a file size of 16 MB with a speed up of 2.11.

Table 3. Twofish encryption performance results

Input (MB)	Timings (msec)			Throughput (Mb/s)		Speed Up
	GPU		CPU	GPU	CPU	
	*Total time	GPU time		*Total time		
1	28	23	50	285	160	1.79
4	104	90	201	307	159	1.93
16	390	355	821	328	155	2.11
32	788	707	1655	324	154	2.10
64	1574	1445	3130	325	163	1.99

**Total time donates the GPU encryption time with the data transfer to and from the GPU*

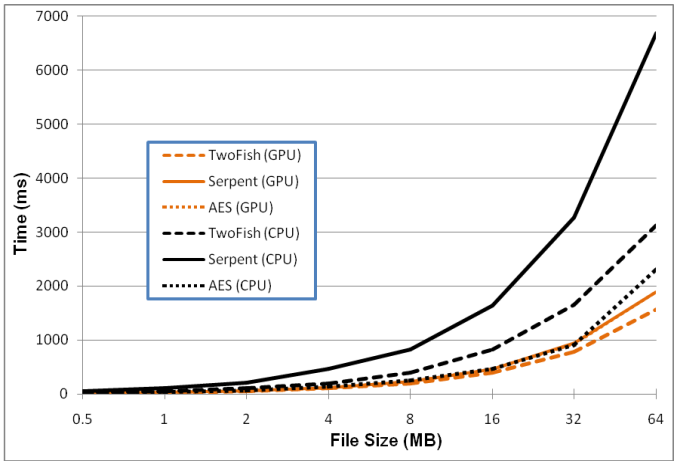

Figure 2. Timing results of AES, Serpent and Twofish on CPU and GPU

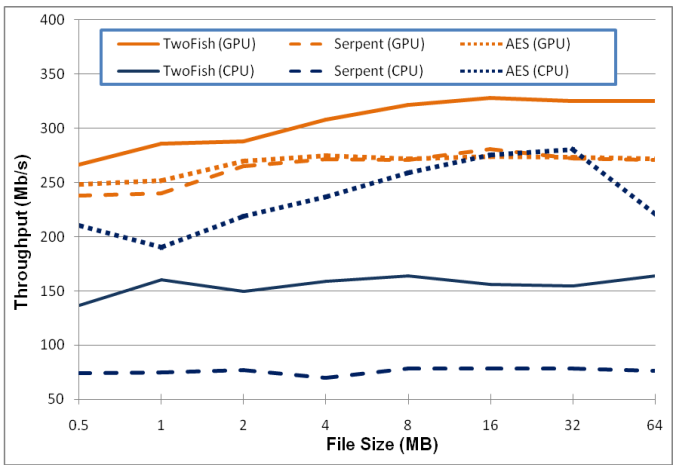

Figure 3. Throughput results of AES, Serpent and Twofish on CPU and GPU

However, the results obtained do not rise to the expectations we had. We believe this is due to several issues. First of all, the graphic card used is an average graphics card (designed for laptops) with comparison to the processor used and many more powerful models could give better results, but this was the only card available to us to test on. The code used is optimized for CPU which means it contains many memory lookups instead of calculations. This could speed up performance on a CPU but could introduce latency on the GPU, as the GPU takes less time in computations than in memory access [8]. It might give better performance to reduce the memory accesses as possible. Another issue is that the shared data between the running threads (key schedule and other look up tables) so if two threads try to access the same index in the array the access will turn into serial access. This does not affect the performance largely.

5.3 OpenSSL test

The OpenSSL toolkit provides a command to test the throughput of some optimized block ciphers. Unfortunately it only contains the test for the AES CBC mode and the Blowfish cipher which was the base cipher for the Twofish. So we run the OpenSSL speed command for AES-CBC-128 bits and for Blowfish then show the results in table 4. However, since the mode of operation used is different, it is irrelevant to compare it with our results which are based on the ECB mode of operation. The test was run on Linux Ubuntu v8.04 (32-bits) running on the same PC used for the previous tests, with OpenSSL v0.9.8g.

	AES-CBC-128 bits (block size)		Blowfish-CBC (block size)	
	16B	64B	16B	64B
Throughput MB/s	91.2	119	91.6	98.4

Table 4. AES and Blowfish OpenSSL speed test results

6. Conclusions

The work done proves that GPU programming could be used to get speed ups in the encryption and decryption of the three block ciphers chosen. We were able to get performance improvements by implementing the block ciphers on the GPU. During this work we have noticed that different ciphers give different speed ups depending on the nature of the cipher. However, more optimizations could be proposed to further achieve higher speed ups, and similar implementations could be done on different types of ciphers such as asymmetric ciphers and hash functions.

7. Future Work

There are some proposed ideas to further improve the work done in this project. A new better graphics card which is more powerful than the one used in this project would give better speed ups in terms of results, so trying this would be interesting. Testing more configurations such as trying to reduce the memory look ups in the encryption function and relying more on calculations might increase the performance. Moreover, the implementation could be extended to cover more operating modes such as the counter mode, or the decryption of the CBC mode. It may even be extended to explore the possibility of implementing asymmetric key algorithms, hashing or even files compression.

8. REFERENCES

[1] D. Robert Stinson. Cryptography: theory and practice, 3rd Edition. CRC Press, 2006. p. 102-112.

[2] Ross Anderson, Eli Biham and Lars Knudsen. Serpent home page. [Online] June 07, 2009. [Cited: June 17, 2009]. http://www.cl.cam.ac.uk/~rja14/serpent.html

[3] Bruce Schneier. Twofish. [Online] June 8, 2009. [Cited: June 07, 2009] http://www.schneier.com/twofish.html

[4] NVIDIA_CUDA_Programming_Guide_2.2. CUDA Toolkit. Version 2.2. [Cited: June 07, 2009].

[5] cbguder. cbguder's aes-on-cuda master –Github. [Online] February 02, 2009. [Cited: June 17, 2009]. http://github.com/cbguder/aes-on-cuda/tree/master

[6] Dr Brian Gladman. Index of /pub/crypt/cryptography/symmetric/serpent. [Online] 14th January 1999. [Cited: June 17, 2009]. http://www.nic.funet.fi/pub/crypt/cryptography/symmetric/serpent

[7] Dr Brian Gladman. Index of /pub/crypt/cryptography/symmetric/twofish. [Online] 14th January 1999. [Cited: June 17, 2009]. http://www.nic.funet.fi/pub/crypt/cryptography/symmetric/twofish

[8] Svetlin A. Manavski, "Cuda compatible GPU as an efficient hardware accelerator for AES cryptography", In Proc. IEEE International Conference on Signal Processing and Communication, ICSPC 2007, (Dubai, United Arab Emirates), pp.65-68, November 2007. [Online] May 28, 2009. [Cited: June 17, 2009]. http://www.manavski.com/downloads/PID505889.pdf