IOSR Journal of Electronics and Communication Engineering (IOSR-JECE)
e-ISSN: 2278-2834, p-ISSN: 2278-8735
PP 38-46
www.iosrjournals.org

A Review on An Encryption Engines For Multi Core Processor Systems

Bodake Vijay[1] & Gawande R.M.[2]
[1,2](Comp. Engg. Dept.,MCERC Nasik. SPP Univ., Pune(MS), India)

Abstract : *With the development of the GPGPU (General-purpose computing on graphics processing units) more and more computing problems are solved by using the parallel property of GPU (Graphics Processing Unit). CUDA (Compute Unified Device Architecture) is a framework which makes the GPGPU more accessible and easier to learn for the general population of programmers. This is because it builds on C and hides many of the complicated details of how the GPU works from a CUDA developer. Using the unique properties of the GPU through CUDA has greatly increased the efficiency of many computational problems. The target in this project is to study and analyze the majority of algorithms related to the cryptography and then to design and make an implementation of an algorithm in CUDA. Finally, this reach will compare the performance between the GPU implementation and the CPU implementation in order to look into the possibility of improving the performance of algorithms. The survey done in three cryptography algorithm AES, RSA, MD5.*

Keywords: *Advanced Encryption Standard, Graphics Processing Unit, Parallel Computing, CUDA, cryptography.*

I. Introduction

This section introduces to the entire background of the proposed work. It highlights the overall description of entire work. It also highlights the concept of cryptography and GPU.

In current years, with the speedy expansion of microelectronics technology, the computing ability of many general-purpose processors has gone distant beyond CPU. Amongst them, the Graphics Processing Unit (GPU) is a typical example. The enhancement of GPU technology has greatly improved the computer graphics processing speed and encouraged the improvement of computer graphics related applications. At the same time, the techniques of parallel computing, streaming processor and programmability of GPU provide a running platform for general-purpose computing beside graphics processing. In this project, normally focus on the application of general-purpose computation on GPU. It propose a new approach of fast AES parallelizing algorithm according to the architecture of GPU. This approach can improve the throughput and speed of AES encryption by optimizing round function and large-scale parallel computing technology. According to this approach, design a fast data encryption system based on parallel AES algorithm that is implemented on GPU. This system can take full advantage of the great concert computing capability of GPU and performing large-scale parallel computation of AES blocks, thus reach fast AES encryption of data. It has the vital significance in the practical application of computer information security. Cryptography is the learning of scientific methods concentrated on data security, authentication, including privacy and data reliability. When encrypting, decrypting, and hashing data an implementation of cryptography is typically comprised of computationally intensive algorithms which are used by applications.

Information security is the hot topic of research in the field of computer science and technology, and the data encryption is one of the most significant methods for information security. Since a new kind of encryption algorithm, i.e. Advanced Encryption Standard (AES), has been projected for substituting the previous encryption of Data Encryption Standard (DES) in 2001, most applications are starting to use AES instead of DES to protect their information security in the former ten years. Presently, the implementations of AES are based on CPU because CPU is observed as the computing component in the computer system from the traditional point of view. With the speedy development of information data, most applications require encrypting data with the performance of high speed. The traditional CPU-based AES implementation shows the poor performance and cannot meet the demands of fast data encryption. Therefore, how to improve a new technique for high performance is an interesting subject of research, which are interesting most researchers in developing new approaches for fast AES encryption.

II. Literature Survey

In this chapter compare the traditional AES encryption with the GPU encryption and try to find the optimize method for the encryption.

In disparity, CPU implementations have lagged behind as a direct result of their inherently serial nature. CPUs are best at performing sequential operations and are thus limited in their ability to take advantage of any parallelism. Multi-core CPUs have improved the CPUs ability to handle multiple threads of execution in parallel but are still restricted to small numbers of threads. Consequently, typical computer systems have traditionally relied on expensive expansion cards for high-speed encryption and hashing. However, now all computers are equipped with some sort of GPU. The parallel processing limitations inherent in CPUs can now be compensated by utilizing a GPU capable of general purpose stream processing [4] [5]. Currently, two such GPU technologies exist NVIDIA's CUDA [6] and ATI's Stream [7]. Of these two, CUDA is somewhat more developed and has had wider industry use [6] [7].

There have been several tries exploiting GPU, particularly AES cryptography in NVIDIA CUDA framework and then displayed the performance improvement over a traditional CPU. Most of them have applied a traditional AES CUDA-GPU implementation, i.e. dividing plaintext into an equal chunk size, then each individual chunk will be encrypted in each AES GPU chunk with multiple threads with/without operational modes [10-13]. Notice that all of them investigated on different GPU models with the comparative performance illustration.

Some proposals have also considered other performance factors; for example, S. A. Manavski et al. [17] proposed AES implementation on CUDA-GPU utilizing an offchip constant (slow) memory without operational mode consideration [16].

In 2009, A. D. Biagio et al. [18] evaluated the performance of NVIDIA 8800 GT GPU of parallel AES implementation using on-chip shared memory. Similarly, P. Maistri et al. [16] also investigated on AES parallelism with NVIDIA 9600 GT and 260 GTX comparatively. D. Lee et al. [21] investigated on parallel AES implementation without operational mode providing the pseudo-code leading to reachable code, but they do not deliberate each separate AES state for optimization purpose. Thus, without any optimization, there is not much benefit for AES on parallel architecture. In addition, C. Mei et al. [19] evaluated parallel AES design operation of NVIDIA GeForce 9200M given the conversation of several memory usages. Note that there is a difficulty to acquire the actual CUDA implementation. In addition, they do not practically discuss the actual parallel algorithm for performance evaluation among several parallel AES implementations. Consider the closely related work, In addition, M. Kipper et al. [22] discussed the AES implementation detail and provide the actual code; and when investigating into its detail, the optimization was occurred during the SubByte and MixColumn states. Although block ciphers operate at a computational complexity of (n) where n is the number of blocks the share size of the data to be encrypted has caused many to look for improvements in the encryption algorithms. As the processing power of such GPUs increases so does the options for other, non-graphics associated applications to be executed on them. With the increase over the years of complex data that must be stored securely, the use of encryption methods has become well-known.

Here NVidias CUDA language is used to implement both AES and DES algorithms to encrypt documents composed of random bytes in approximately one third and one fourth of time required by traditional CPUs [13].

Lately, the research community has started to accelerate cryptographic algorithms using the GPU. For example, various authors looked at the feasibility of the current industry standard for symmetric cryptography, the Advanced Encryption Standard (AES) [21, 18, and 9]. Only two groups, namely Fleissner and Moss et al, have intended for the efficient implementation of modular exponentiation on the GPU [20, 14]. Their results were not hopeful, as they were restricted by the legacy GPU architecture and interface (cf. the next section). To the best of my knowledge there are neither publications about the implementation of these systems on modern, GPGPU-capable hardware nor on the implementation of elliptic curve based systems. It aim to fill this gap by implementing the core operations for both systems efficiently on modern graphics hardware, creating the basis for the use of GPUs as accelerators for public key cryptography. It will use Nvidias current graphic GPU series, the G80 generation, together with its new GPGPU interface CUDA [14].

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.[15]

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 that 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.

In addition, M. Kipper et al. [19] discussed the AES implementation detail and provide the actual code and when investigating into its detail, the optimization was occurred during the SubByte and MixColumn states.

III. The Cuda Architecture

CUDA (Compute Unified Device Architecture) was first proposed by NVidia in 2007, offering versions for both the Windows and Linux operating systems, while version 2.0 contained support for MacOS X also. CUDA was designed to be of use in both professional and casual graphics cards, having similarities with the OpenCL framework and with Microsoft's Direct Compute alter- native[7][22]. Figure 1 presents the overall CUDA architecture [7].Kepler GTX 680 is the example of great performance/watt Streaming Multiprocessor, also known as "SMX." In GTX680 total no of CUDA core is 1536 .GTX Core work on Base Clock 1006 and Boost Clock 1058[10].Each CUDA core in GTX 680 Support 1024 number of active thread. The GTX 680 has 2048MB memory which work on 6 MHz Speed [7]. Because of this eForce GTX 680 to deliver revolutionary performance/watt when compared to GeForce GTX 580.

3.1 The Programming Paradigm

CUDA SDK uses an extended C language that allows the user to program using the CUDA architecture. A user defined C function that is executed in the GPU is called a kernel [7][22]. A set of parallel threads, which are organized into thread blocks and grids of thread blocks, execute the kernel concurrently .The programmer specifies the number oftimes the kernel has to be executed by specifying the number of threads in the program. Each thread executes one instance of the kernel. So, if the user specifies the number of threads as N, the kernel will be executed N times by N different threads [22]. CUDA follows a Single Instruction Multiple Thread (SIMT) programming model. The Kepler architecture also supports concurrent global kernel execution by allowing up to 16 kernels to execute simultaneously. The limitation with executing multiple kernels is that all kernels must belong to the same program, as CUDA cannot manage application level parallelism.

3.2 Gpu Thread Architecture

The massive parallelism in the CUDA programming model is achieved through its multi-threaded architecture. This thread parallelism allows the programmer to partition the problem into coarse sub problems that can be processed in parallel by blocks of threads, and each sub problem is further divided into finer pieces that can be solved cooperatively in parallel by all threads within a block [22]. The CUDA threads are organized into a two-

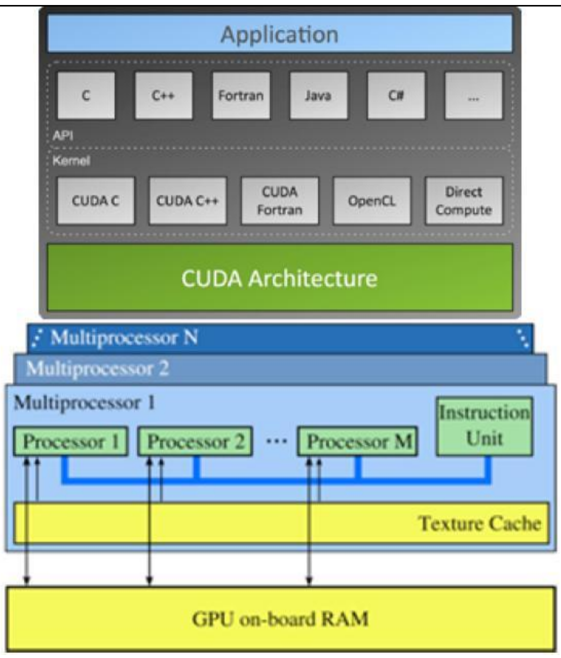

Fig 1: CUDA Architecture

level hierarchy using unique coordinates called block JD and thread JD. Each of these threads can be independently identified within the kernel using its unique identifier represented by the built-in variable blockldx and threadJdx [22]. The programmer can configure the number of threads required in a thread block, with a maximum of 1024 threads per block. An instance of the kernel is executed by each of these threads. Figure 2 shows The CUDA thread block structure.

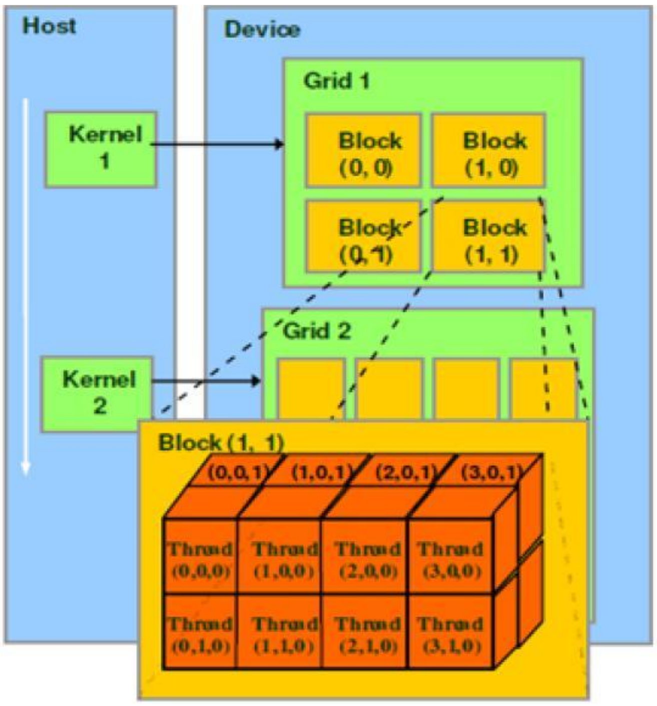

Fig2: CUDA Thread Block Structure

3.3. Cuda Memory Organization

A large portion of computing time on the device is spent on data movement, especially the reading of data into the individual threads. Since there are hundreds of arithmetic units on the GPU, the memory bandwidth of the computer chip is often the bottleneck or major time consumer [22]. With four types of memory available on the GPU, picking the right memory type and utilizing it correctly is crucial for maximum speed. The four types of device memory are:

3.3.1 Global Memory:

Global Memory By far the largest amount of memory on the device is read-and-write global memory: 500 megabytes. Although far slower than other memory types, it is relatively constraint free and the easiest to use.

3.3.2 Shared Memory:

The shared memory is read-and-write memory resides physically on the GPU. It is placed as opposed to off-chip DRAM, so it is much faster than global memory. Only threads is one block is allow to access shared memory. Thread in one block can access that block share memory but threads in other block don’t have access to share memory in different block. This type of memory provides an excellent speedup because threads in one block communicate with each other and use share memory. But synchronization is necessary between threads. For example, each thread in a block computes a correlation value for a specific target to reference comparison. It is imperative that each thread has completed its process before thread correlation values are compared to each other, or the results may be invalid. Synchronization forces each thread to wait until all threads in a block have finished computing their correlation value before proceeding with the process of finding the maximum value of the block. In addition to the synchronization constraint, shared memory is limited to 16 kilobytes per block.

3.3.3 Constant Memory:

Constant memory is read-only memory that does not change over the course of kernel execution. A single read can be broadcast to "nearby" threads in a half-warp saving up to 15 reads [22]. Because constant memory is cached consecutive reads of the same address take relatively little time. Constant memory is most effective when all 16 members of the half-warp are reading the same address. However, when each half- warp member is reading a different address, the read requests are serialized and can take longer. In GTX 680 constant memory size is 65536KB.

3.3.4 Texture Memory:

Texture memory is global memory [22]. The difference between global memory and texture memory is that, the textures memory is accessed through a dedicated read-only cache, and this cache performs filtering which can perform linear floating point operation during read process. The cache, however, is different to a normal cache, in that it is optimized for three-dimensional locality and not locality in memory. For some applications, this is ideal and will give a performance improvement in both because of the cache and the free FLOPs you can get from the filtering hardware, but for others, it won’t and textures can be slower because access include a cache miss penalty in addition to the global memory read, and it is very costly operation.

3.4 Cuda Processing Flow

Following Figure 3 shows CUDA Programe communication with different Device parts .It starts with, Copy data from main memory to GPU memory [7]i.e. Global Memory then CPU instructs the process to GPU by executing Kernel Command after that GPU execute parallel in each core and Copy the result from GPU memory to main memory.

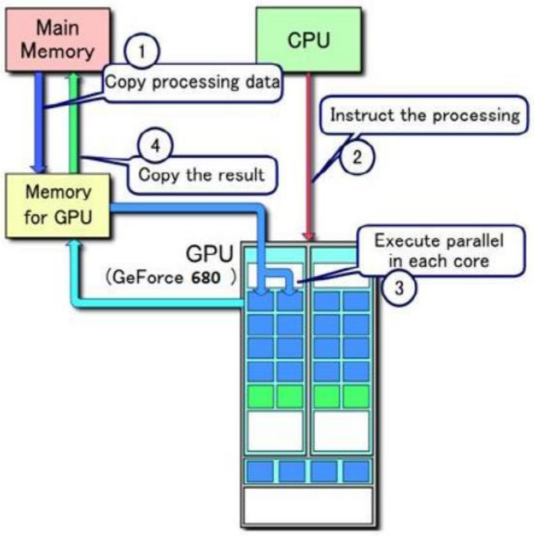

Fig 3: CUDA Processing Flow.

IV. Serial Aes Algorithm

AES, i.e. Rijndael algorithm is a symmetric key cryptography. The AES standard contains three i.e. AES-128, AES-192 and AES-256 block ciphers. The encryption of AES is supported with a 128 bits fixed block size of each. The AES cipher calculation is specified as a number of repetitions of transformation rounds that convert the input plaintext into the final output of ciphertext. Each round contains the several processing steps, which include by encryption key. A set of inverse rounds are applied to convert the cipher text back into the original plaintext using the same encryption key. Figure 1.1 shows the flowchart of the AES-128 algorithm.

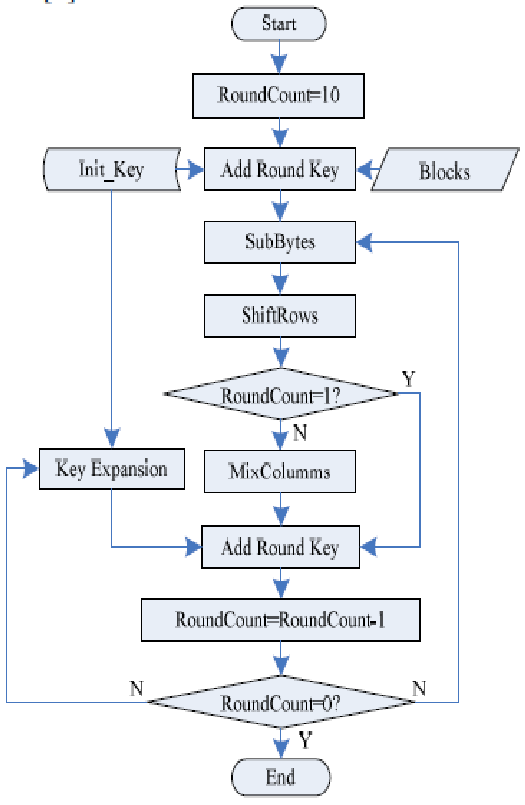

Fig 4: AES 128 bit Flowchart

From this figure, I can see that the AES-128 algorithm is iterative and consists of 10 rounds. The input is a chunk of data and the initial key. Each round operates on the middle result of the preceding round and is a order of the four transformations, namely SubBytes, ShiftRows, MixColumns and AddRound-Key. The middle result of any step is called the state. The final round is slightly different and the output after 10 rounds is the block of encrypted data.

These four basic transformations are applied as follows:

1. SubBytes: The SubBytes operation is a nonlinear byte substitution. Each byte from the input state is replaced by another byte according to the substitution box (called the S-box). The S-box is computed based on a multiplicative inverse in the finite field GF (2^8) and a bitwise affine transformation.

2. ShiftRows: In the ShiftRows transformation, the first row of the state array remains unchanged. The bytes in the second, third, and forth rows are cyclically shifted by one, two, and three bytes to the left, respectively.

3. MixColumns: During the MixColumns process, each column of the state array is considered as a polynomial over GF (28). After multiplying modulo x4 + 1 with a fixed polynomial a(x), given by

$$A(x) = 03x^3 + 01x^2 + 01x + 02 \qquad (1)$$

The result is the corresponding column of the output state.

4. AddRoundKey: A round key is added to the state array using a bitwise exclusive-or (XOR) operation. Round keys are calculated in the key expansion process. If Round keys are calculated on the byte for each data block, it is called AES with online key expansion. On the other hand, for most applications, the encryption keys do not change as frequently as data. As a result, round keys can be calculated before the encryption process, and kept constant for a period of time in local memory or registers. This is called AES with offline key expansion.

Similarly, there are three steps in each key expansion round.

1. KeySubWord: The KeySubWord operation takes a four byte input word and produce an output word by substituting each byte in the input to another byte according to the S-box.

2. KeyRotWord: The function KeyRotWord takes a word $[a_{3,}\ a_{2,}\ a_{1,}\ a_0]$ performs a cyclic permutation and returns the word $[a_{2,}\ a_{1,}\ a_{0,}\ a_3]$ as output.

3. KeyXOR: Every word w[i] is equal to the XOR of the previous word, w [i-1] and the word Nk positions earlier, w[i-Nk]. Nk equals 4, 6 or 8 for the key lengths of 128, 192 or 256 bits, respectively.

V. Parallel Aes With Gpu Implementation

When implementing AES on GPU, and more specifically on CUDA, I 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. My expectations would be that the global memory of the GPU is much slower to access than to compute the values. In tests done and 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.

Many instances of parallelism are inherent in the AES algorithm. A coarse-grained level of parallelism is provided by the independence of each 16-byte block, allowing all bytes of a block to be encrypted independently of each other. This independence is reduced for certain modes of operation, but those modes are not considered in this paper. Within each block encryption, each function within a round can be performed independently of other instances of that function: an s-box substitution can be performed on each byte independent of any other byte, each row can be shifted independent of any other row, each column can be mixed independent of any other column, and each word of the round key can be added independent of any other round word. To achieve the desired maximum occupancy, it is necessary to consider only the coarse grained parallelism provided by individual data blocks. Each CUDA thread completes the AES encryption on a single 16-byte block of data. All 16-byte blocks are computed in parallel. Implementing the aforementioned block level parallelism would be beneficial only for smaller datasets, which do not generate full occupancy of the GPU. Below fig. 5 shows the AES parallel structure.

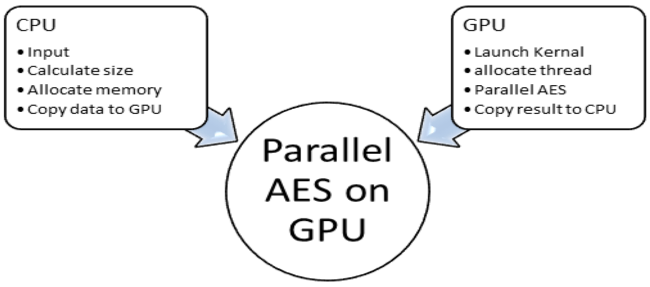

Fig 5: AES Parallel Structure

The proposed system architecture in which CPU (Host) is take the input in 128 bit data for encryption from user .After that this data is divided into 64 Kb chunk which are used to allocate memory on the GPU using malloc function of CUDA language. These data then copy on the GPU for processing then CUDA launch the kernel for the encryption .Kernel calculate the thread index with this it perform the parallel AES encryption .In the parallel encryption the functions sub Bytes ,Shift row, Mix columns and AddroundKey are work as parallel for encryption. Then these data is collect on GPU and transfer the data on CPU to user. This way my technique speeding the AES encryption with the help of GPU.

VI. Conclusion

To overcome the issue of low efficiency over the traditional CPU-based implementation of AES, I proposed a new algorithm for AES parallelism in this project. According to this proposal, designing and implementing the parallel AES algorithm based on GPU. This implementation will achieves up to 7x speedup over the implementation of AES on a comparable CPU. This implementation can be applied for the computer forensics which requires high speed of data encryption. In the future work will include GPU implementations of hashing and public key algorithms (e.g. MD5, SHA-1 and RSA) in order to create a complete cryptographic framework accelerated by GPU.

References

[1] Bin Liu, Student Member, IEEE, and Bevan M. Baas, Senior Member, IEEE, Parallel AES Encryption Engines for Many-Core Processor Arrays, ieee transactions on computers, vol.62, no. 3, march 2013
[2] NIST, Advanced Encryption Standard (AES), http://csrc.nist.gov/publications/_ps/_ps197/_ps-197.pdf, Nov. 2001.
[3] NIST, Data Encryption Standard (DES), http://csrc.nist.gov/ publications/_ps/_ps46-3/_ps46-3.pdf, Oct. 1999.
[4] NVIDIA Co., "NVIDIA CUDA Best Practices Guide Version 2.3," Santa Clara,California, 2009.
[5] NVIDIA Co., "NVIDIA CUDA Programming Guide Version 2.3.1," Santa Clara,California, 2009.
[6] NVIDIA Co. CUDA Zone. [Online]. http://www.nvidia.com/cuda
[7] AMD. (2010, August) ATI Stream Technology. [Online]. http://www.amd.com/US/PRODUCTS/TECHNOLOGIES/STREAMTECHNOLOGY/ Pages/stream-technology.aspx
[8] J. L. D. Comba, C. A. Dietrich, C. A. Pagot and C. E. Scheidegger. Computation on GPUs: From a Programmable Pipeline to an Efficient Stream Processor. Revista de Informtica Terica e Aplicada, vol. X, no. 1, 2003, pp. 41-70.
[9] D. Luebke. CUDA: Scalable Parallel Programming for High- Performance Scientific Computing. In Proceedings of 5th IEEE International Symposium on Biomedical Imaging, From Nano to Macro (ISBI 2008), Paris France, May 2008, pp. 836-838.
[10] M. Garland, S. Le Grand and J. Nickolls et al. Parallel Computing Experiences with CUDA. IEEE Micro, vol. 28, no. 4, pp. 13-27, 2008.
[11] NVIDIA. CUDA [EB/OL]. (2010-01-09) http://www.nvidia.cn/object/cudahomecn.html.
[12] C. J. Thompson, S. Hahn, and M. Oskin. Using Modern Graphics Architectures for General-Purpose Computing: a Framework and Analysis. In Proceedings of the 35th Annual ACM/IEEE international Symposium on Microarchitecture (MICRO-35). Istanbul, Turkey. November 2002, pp.306-317.
[13] Brandon P. Luken, Ming Ouyang, and Ahmed H. Desoky"AES and DES Encryption with GPU", Computer Engineering and Computer Science Department University of Louisville Louisville, KY 40292
[14] Robert Szerwinski and Tim Guneysu"Exploiting the Power of GPUs for Asymmetric Cryptography" Horst Gortz Institute for IT Security, Ruhr University Bochum, Germany szerwinski,gueneysu@crypto.rub.de.
[15] Svetlin A. Manavski "cuda compatible gpu as an efficient hardware accelerator for AES cryptography"2007 IEEE International Conference on Signal Processing and Communications (ICSPC 2007), 24-27 November 2007, Dubai, United Arab Emirates.

[16] Chakchai So-In, Sarayut Poolsanguan, Chartchai Poonriboon, Kanokmon Rujirakul, Comdet Phudphut "Performance Evaluation of Parallel AES Implementations over CUDAGPU Framework "Department of Computer Science, Faculty of Science, Khon Kaen University Maung, Khon Kaen, Thailand, 40002.
[17] S. A. Manavski "CUDA compatible GPU as an efficient hardware accelerator for AES cryptography" In Proceeding(s) of the IEEE International Conference on Signal Processing and Communication, pp. 65-68, 2007.
[18] P. Maistri, F. Masson, and R. Leveugle, Implementation of the Advanced Encryption Standard on GPUs with the NVIDIA CUDA framework, In Proceeding(s) of the IEEE Symposium On Industrial Electronics and Applications, pp. 213-217, 2011.
[19] A. D. Biagio, A. Barenghi, G. Agosta, and G. Pelosi, "Design of a Parallel AES for Graphics Hardware using the CUDA framework," In Proceeding(s) of the IEEE International Conference on Parallel and Distributed Processing, pp. 1-8, 2009.
[20] C. Mei, H. Jiang, and J. Jenness, "CUDA-based AES parallelization with _ne tuned GPU memory utilization ", In Proceeding(s) of the IEEE International Symposium on Parallel and Distributed Processing, Workshops and Ph.D. Forum, pp. 1-7, 2010.
[21] D. Le, J. Chang, X. Gou, A. Zhang, and C. Lu, "Parallel AES Algorithm for Fast Data Encryption on GPU ", In Proceeding(s) of the International Conference on Computer Engineering and Technology, pp. V6-1-6-6, 2010.
[22] N Wilt. The CUDA Handbook: A Comprehensive Guide to GPU Programming.Addison-Wesley Professional, 2013.