Parallel AES Algorithm for Fast Data Encryption on GPU

Deguang Le, Jinyi Chang, Xingdou Gou, Ankang Zhang, Conglan Lu
School of Computer Science & Engineering
Changshu Institute of Technology
Changshu 215500, China
ledeguang@gmail.com

***Abstract*—With the improvement of cryptanalysis, More and more applications are starting to use Advanced Encryption Standard (AES) instead of Data Encryption Standard (DES) to protect their information security. However, current implementations of AES algorithm suffer from huge CPU resource consumption and low throughput. In this paper, we studied the technologies of GPU parallel computing and its optimized design for cryptography. Then, we proposed a new algorithm for AES parallel encryption, and designed and implemented a fast data encryption system based on GPU. The test proves that our approach can accelerate the speed of AES encryption significantly.**

Keywords- Advanced Encryption Standard (AES); Graphics Processing Unit (GPU); Parallel Computing

I. INTRODUCTION

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 important methods for information security. Since a new kind of encryption algorithm, i.e. Advanced Encryption Standard (AES), has been proposed for replacing the previous encryption of Data Encryption Standard (DES) in 2001, More and more applications are starting to use AES instead of DES to protect their information security in the past ten years. Currently, the implementations of AES are based on CPU because CPU is regarded as the computing component in the computer system from the traditional point of view. With the rapid growth of information data, more and more applications require encrypting data with the performance of more and more high speed. The traditional CPU-based AES implementation shows the poor performance and can not meet the demands of fast data encryption. Therefore, how to develop a new method for high performance is a challenging topic of research, which are interesting more and more researchers in developing new approaches for fast AES encryption.

In recent years, with the rapid development of microelectronics technology, the computing capability of many general-purpose processors has gone far beyond CPU. Among them, the Graphics Processing Unit (GPU) is a typical example. The improvement of GPU technology has greatly enhanced the computer graphics processing speed and image quality, and promoted the development of computer graphics-related applications. At the same time, the techniques of streaming processor, parallel computing and programmability of GPU provide a running platform for general-purpose computing beside graphics processing. Therefore, the GPU-based general-purpose computing is a hot topic of research [1]. In this paper, we focus on the application of general-purpose computation on GPU. We propose a new approach of fast AES parallelizing algorithm according to the architecture of GPU. Our approach can improve the throughput and speed of AES encryption by optimizing round function and large-scale parallel computing technology. According to our approach, we design a fast data encryption system based on parallel AES algorithm that is implemented on GPU. Our system can take full advantage of the high-performance computing capability of GPU and performing large-scale parallel computation of AES blocks, thus achieve fast AES encryption of information. Our system has the important significance in the practical application of computer information security and forensics [2].

The rest of this paper is organized as follows. Section II introduces the traditional algorithm of AES encryption. In section III, we study the technologies of general-purpose computation of GPU. In section IV, we propose the parallelized algorithm of AES encryption and analyze its principle of AES parallelism. Then, we detail the implementation of a fast encryption system for accelerating the AES encryption on GPU. Furthermore, we test and analyze the performance of our approach by comparing with the traditional approach, which shows the advantage and higher performance of our approach. Finally, we make an conclusion and propose our future work in section V.

II. TRADITIONAL AES ALGORITHM

AES, i.e. Rijndael algorithm[3]，is a symmetric key cryptography. The AES standard comprises three block ciphers, i.e. AES-128, AES-192 and AES-256. The encryption of AES is carried out in blocks with a fixed block size of 128 bits 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 consists of several processing steps, including one that depends on the encryption key. A set of reverse rounds are applied to transform the ciphertext back into the original plaintext using the same encryption key. Figure 1 shows the flowchart of the AES-128 algorithm.

From this figure, we can see that the AES-128 algorithm is iterative and consists of 10 rounds. The input is a block of data and the initial key. Each round operates on the intermediate result of the previous round and is a sequence of the four transformations, namely SubBytes, ShiftRows, MixColumns and AddRound-Key. The intermediate 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. The detailed description of the algorithm can be referred in [3].

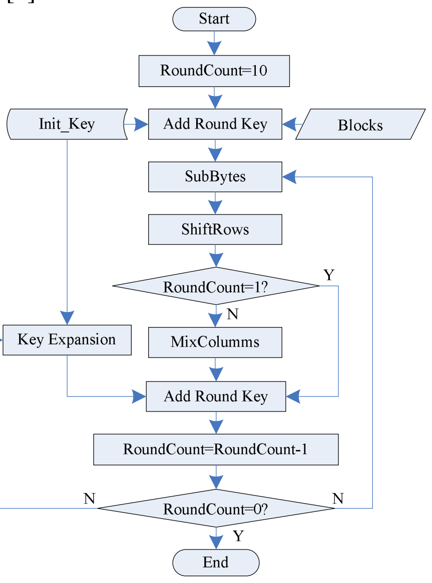

Figure 1. Flowchart of AES-128 alogrithm.

III. GENERAL PURPOSE COMPUTATION ON GPU

Driven by the insatiable market demand for general-purpose computing as well as real-time, high-definition graphics, GPU introduce a lot of hardware and software techniques, such as unified shader architecture, streaming processors, scalable parallel computing and programmability [4-5]. In this section, we will present these technical principles for general-purpose GPU computing.

A. *Unified Shader Architecture*

In a traditional GPU, it includes discrete pixel and vertex shader units. There exists a real possibility that for any given scene either the pixel shader engine or the vertex shader engine could go underutilized, leaving hardware idle and affecting potential performance [6]. Therefore, it's very difficult to make a GPU work as efficiently as possible with discrete pixel and vertex shader units. In order to take full advantage of both pixel and vertex shader units and increase the performance of multi-task shaders, the shader unification is proposed in the new GPU architecture, called unified shader architecture [7].

With the unified shader architecture, GPU can perform vertex or pixel calculations in the same units. Therefore, the benefit of unified versus discrete pixel and vertex shader units is that the unified shader can be allocated to either pixel or vertex shader as needed by the application. If a scene has a lot of pixel shader effects and few vertex shader effects, most of the units would be used for the pixel shader effects while only a few would be used for the vertex shader effects. If there are a lot of vertex effects, then the opposite is true. It is the flexibility here that makes a unified architecture efficient, and improvements to efficiency lead to improvements in overall performance.

B. *Streaming Processors*

The unified shader architecture has prompted GPU to be equipped with multiple individual scalar Streaming Processors (SPs) instead of 32 4-component vector processors used on its previous GPU [8]. Each streaming processor of GPU is capable of being used for pixel or vertex operations. Besides, these SPs are organized in group, called Streaming Multiprocessor (SM). For example, NVIDIA G80 GPU includes 128 streaming processors, a SM consists of eight streaming processors. Therefore, G80 GPU contains 16 SMs. In GPU, the SM creates, manages and executes concurrent threads in hardware with zero scheduling overhead. It implements the multiple data computations through barrier synchronization intrinsic with a single instruction. Fast barrier synchronization together with lightweight thread creation and zero-overhead thread scheduling efficiently support very fine-grained parallelism, allowing, for example, a low granularity decomposition of problems by assigning one thread to each data element.

To manage hundreds of threads running several different programs, the SM employs a new architecture we call Single Instruction Multiple Thread (SIMT). In SIMT, the SM maps each thread to one scalar streaming processor, and each scalar thread executes independently with its own instruction address and register state. The SIMT of SM unit creates, manages, schedules and executes threads in groups of 32 parallel threads called warps. Individual threads composing a SIMT warp start together at the same program address but are otherwise free to branch and execute independently. Therefore, the technique of streaming processors can achieve high performance in scientific computing that shares stream characteristics of large parallelism and intensive computation.

C. *Parallel Computing*

As an kind of computing device, GPU is featured of parallel computing compared with traditional CPU that is serial computing. The reason behind the discrepancy in computing mode between CPU and GPU is that GPU is specialized for compute-intensive, highly parallel computation – exactly what graphics rendering is about – and therefore designed such that more transistors are devoted to data processing rather than data caching and flow control [9]. Thus, the massively parallel device of GPU is capable of processing the large amounts of data required to perform the classification and detection tasks very quickly.

For parallel computing, the user can define threads which run on GPU in parallel using standard instructions we are familiar with within the field of general purpose programming [10]. The user declares the number of threads which must be run on a single SM by specifying a block size. The user also defines multiple blocks of threads by declaring a grid size. A grid of threads makes up a single kernel of work which can be sent to GPU and when finished, in its entirety, is sent back to the host and made available to the application.

D. *Programmability*

GPU was originally designed for operating on large continuous streams of vertex and fragment data. In order to use GPU for general purpose computing, the problem has to be mapped onto the GPU's programming architecture. For this work, the cross platform OpenGL API was used for interfacing with the graphics hardware. The other main graphics API is Microsoft Direct X. Programming GPU involves writing custom vertex and fragment programs to be executed by the vertex and fragment shaders. Therefore, it requires the programmers are familiar with the hardware architecture of GPU and graphics API before they can develop a general-purpose application on GPU. Besides, mapping general-purpose program into graphics API would not only complicate the program but also occupy massive extra resources, and thus reduce the efficiency significantly.

In order to improve the general-purpose computing capability of GPU as well as decrease the difficulty of GPU programming, NVIDIA introduce a new software programming platform, namely Compute Unified Device Architecture (CUDA) [11]. CUDA gives up the traditional graphics API. Its core is the C compatible complier. Compared with standard C complier, the CUDA C compiler can manage the hardware resources of GPU efficiently. Since C language is the worldwide programming language, anyone who knows C language can develop the general-purpose application based on GPU easily. Besides, CUDA provides a lot of function libraries, which hide the hardware resources of GPU. Therefore, programmers can call these functions directly without understanding the lower-level hardware architecture of GPU, which make GPU programming more simply.

IV. Parallel AES Algorithm and Its Implementation on GPU

In this section, we firstly describe the considerations for choosing a suitable GPU. Then, we present the principle for AES parallelism and propose a parallel AES algorithm. Besides, the implementation of the parallel AES algorithm on GPU is presented using the CUDA programming model. Finally, we analyze and evaluate our approach.

A. *Considerations of GPU*

In traditional GPU architecture, the programming is based on OpenGL environment. One disadvantage of OpenGL is that the data are stored in floating-point format in the memory model of OpenGL. Therefore, it is not suitable for AES computation, which requires massive numerical calculation. Another major disadvantage in implementing AES algorithm on traditional GPU architectures is the unavailability of the bitwise logical operations in the programmable shaders. This would concludes CPU keeps loaded during all the execution [12]. Therefore, we would adopt the NVIDIA GPU which supports the CUDA platform.

NVIDIA is the leader who promotes the general-purpose computing on GPU [13]. In our implementation, we use the latest GPU with NVIDIA G92 core. The software environment is based on the C compatible CUDA development platform. We assume that readers are familiar with C language and CUDA programming model. In the next sub-sections, we will details the principle of AES parallelism and its implementation.

B. *Principle of AES Parallelism*

In the traditional implementation of AES, the computation of data blocks is performed serially in GPU. Therefore, the efficiency and speed is poor. Figure 2 illustrates the principle of AES parallelism.

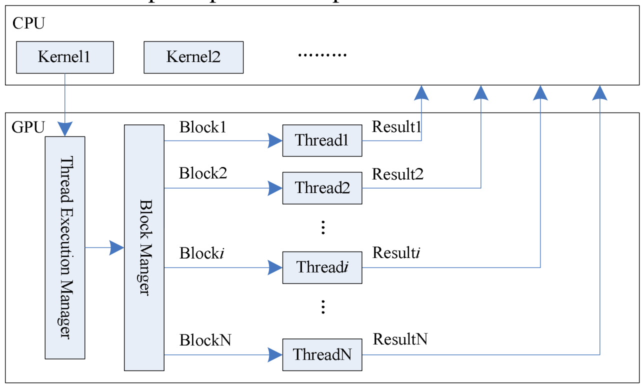

Figure 2. Principle of AES parallelism.

From this figure, we can see that in GPU, it includes three components: the thread execution manager, the block manager and multiple threads. After CPU calls the kernel function executed in GPU, GPU will enable the block manger active through the thread execution manager. The block manager will then divide plaintext into multiple blocks. Then, each block will be computed in individual thread. Finally, the encrypted block will be outputted to CPU, which will be assembled into the ciphertext. Since GPU allows the number of thread in parallel to be the magnitude of one hundred thousand. Therefore, the AES encryption on GPU has high efficiency of parallel computing.

According to the principle of AES parallelism, we propose the parallelized AES algorithm illustrated as follows.

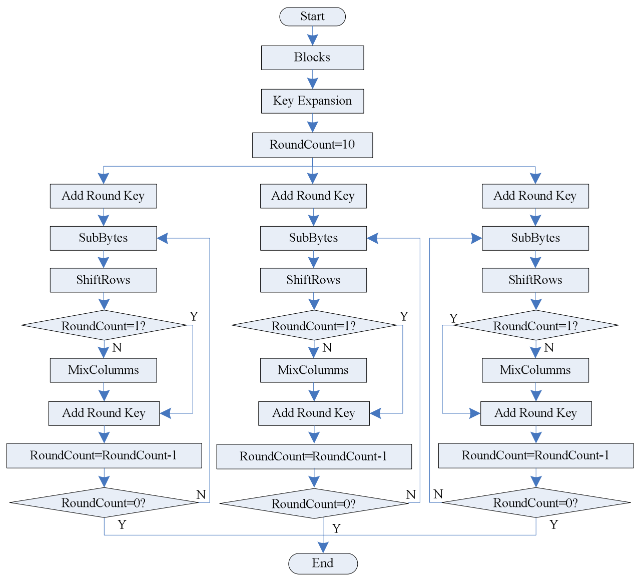

Figure 3. Parallel AES algorithm.

C. *Implementation of AES Parallelism*

According to the proposed parallel AES algorithm, we firstly store the plaintext and expanded key in the global memory space of GPU. The plaintext is then divided in blocks of 16 bytes which are encrypted completely in parallel. One CUDA thread is responsible for computing one block of plaintext. These operations can be executed as follows:

```
unsigned int* d_plaintext;
cudaMalloc((void**)&d_plaintext, sizeof(unsigned int)*
(mem_length ));
cudaMemcpy( d_plaintext, plaintext, sizeof(unsigned int)
* mem_length, cudaMemcpyHostToDevice) ;
unsigned int* d_roundkey;
cudaMalloc( (void**)&d_roundkey, sizeof(unsigned int) *
44 ));
cudaMemcpy( d_roundkey, roundkey, sizeof(unsigned int)
* 44 , cudaMemcpyHostToDevice);
```

Where, d_palintext refers to the variable of plaintext defined in the global memory of GPU. d_roundkey denotes the variable of expanded key allocated in the global memory of GPU.

Threads from the same CUDA block need to share and frequently access the information of AES expanded key. For this reason, it is loaded in shared memory together with the blocks of plaintext processed by that CUDA block through the following operations.

```
__shared__ unsigned int s_roundkey[44];
s_rourndkey[tid] = d_roundkey[tid];
__shared__ unsigned int s_plaintext[BLOCK_SIZE];
s_plaintext[tid] = d_plaintext[tid];
```

Where, s_plaintext and s_roundkey are the variables of plaintext and expanded key respectively, which reside in the shared memory space of a thread block. tid refers to the thread ID.

There are two general methods of AES computation [14]. One is the matrix computation, namely the 128-bit block is mapped into a 4×4 matrix. Then, the matrix is computed in a sequence of four stages: AddRoundKey, SubBytes, ShiftRows and MixColumns in each round. The other is the table lookup. Formula (1) shows the expression of AES transformation through the table lookup [15].

$$e_j = T_0\left[a_{0,j}\right] \oplus T_1\left[a_{1,j-c1}\right] \oplus T_2\left[a_{2,j-c2}\right] \oplus T_3\left[a_{3,j-c3}\right] \oplus k_j \qquad (1)$$

Where, a refers to the input matrix variable. e refers to the output matrix in each round transformation. T_0~T_3 refer to the lookup tables which are obtained through performing the transformation of S-box table $S[a]$ expressed as follows.

$$T_0[a]=\begin{bmatrix} S[a]\bullet 02 \\ S[a] \\ S[a] \\ S[a]\bullet 03 \end{bmatrix} T_1[a]=\begin{bmatrix} S[a]\bullet 03 \\ S[a]\bullet 02 \\ S[a] \\ S[a] \end{bmatrix} T_2[a]=\begin{bmatrix} S[a] \\ S[a]\bullet 03 \\ S[a]\bullet 02 \\ S[a] \end{bmatrix} T_3[a]=\begin{bmatrix} S[a] \\ S[a] \\ S[a]\bullet 03 \\ S[a]\bullet 02 \end{bmatrix}$$

Variable k refers to the round-key. In the last round transformation, because it lacks of the stage of MixColumns, the table T in formula (1) should be replaced using S-box table S [a]. Thus, a single AES round on a state can be done by performing 4 table lookups and 4 32-bit exclusive-or operations.

In the table-lookup, Each T table contains 256 4-byte word entries and these 4 T tables make up for 4KByte of total space. In CUDA-enabled GPU, it includes the global memory, shared memory, texture memory and constant memory. Among them, the global memory can be configured up to 4 GBytes. Given the flexibility of the memory model, it is possible to efficiently use the four T-lookup tables. Besides, the core of CUDA-enabled GPU is a scalar processor. Therefore, there is no need to combine instructions in vector operations in order to get the full processing power. Furthermore, the 32 bit logical exclusive-or operation can be executed natively on CUDA-enabled GPU. Therefore, we adopt table lookup for AES parallelism in this paper.

Figure 4 illustrates the pseudo-code of kernel function of our parallelized AES algorithm implemented on GPU.

```
DWORD tid = ( threadIdx.x + blockIdx.x*THREADS );
uint4 state=data[tid];
input[tid]=state;
state=state^key0;
for round = 1 step 1 to 9
 $state = T_0[state] \oplus T_1[state] \oplus T_2[state] \oplus T_3[state] \oplus k_j$
end for
$state = S_0[state] \oplus S_1[state] \oplus S_2[state] \oplus S_3[state] \oplus k_{10}$
out[tid]=state;
```

Figure 4. Pseudo-code of kernel function of parallelized AES algorithm

From figure 4, we can see that every GPU thread of CUDA block performs 4 table lookups and 4 32-bit exclusive-or operations in each AES round on a state. Besides, 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 block 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 memory and then returned to the CPU.

D. Performance Analysis and Evalulation

In this sub-section, we evaluate the performance of parallel AES algorithm that is used in a cipher system for data encryption. In our cipher system, the hardware are equipped with CPU of Intel Core 2 Duo E8200, the memory of 1GB and a GPU graphics card of NVIDIA GeForce GTS250. The software is our implementation of parallel AES algorithm which runs in Windows XP.

Figure 5 illustrates the comparisons of AES encryption in terms of encryption time and speedup. In this figure, the horizontal axis indicates the size of plaintext in terms of Byte, the left vertical axis indicates the time in which the plaintext are encrypted into the ciphertext through AES algorithm in terms of Second (Sec) and the right vertical axis indicates the metric of speedup which is defined as follows:

$$Speedup = \frac{AES\_GPU\_Time}{AES\_CPU\_Time} \qquad (2)$$

Besides, as illustrated in the figure, the line labeled with Serial_AES represents the AES encryption time spent during the computation by traditional CPU. the line labeled with Parallel_AES represents the AES encryption time spent during the computation by modern GPU. The line labeled with Speedup represents the AES encryption time ratio according to formula (2).

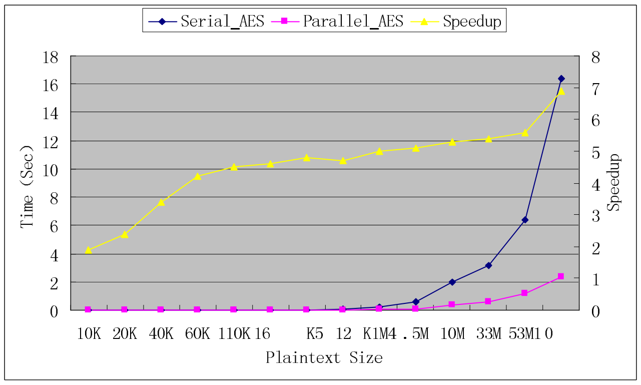

Figure 5. Comparisons of AES algorithms.

It also can be observed from above figure that although the total time required for AES encryption increases with the increment of plaintext size, but the speedup increases as the CPU time increases faster than the GPU time. Therefore, the parallel AES implementation on GPU achieves massive speedups over the serial AES implementation on CPU. Besides, the most effective use of GPU resources are when the plaintext size is sufficiently large to exploit the parallelism of the GPU architecture and hide the memory transfer and access latency. When the plaintext size is less than 10 Kbytes, the time of parallel AES on GPU is about half of that of serial AES on CPU, so the speedup is 2. When the plaintext size is between 10 KBytes and 1 MBytes, the time of our parallel AES on GPU is 4 times fast than that of serial AES on CPU. When the plaintext size is about 200 MBtyes, Our speedup is capped close to 7.

V. Conclusions and Future Work

To overcome the issue of low efficiency over the traditional CPU-based implementation of AES, we proposed a new algorithm for AES parallelism in this paper. According to our proposal, we designed and implemented the parallel AES algorithm based on GPU. Our implementation achieves up to 7x speedup over the implementation of AES on a comparable CPU. Our implementation can be applied for the computer forensics which requires high speed of data encryption. In the future, we will focus on efficient implementations of other common symmetric-key encryption algorithms, such as Blowfish, Serpent and Twofish. Besides, future work will also 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.

Acknowledgments

This work is supported in part by the Program for Advanced Talents within Six Industries of Jiangsu Province under grant No. 09-D-LE Deguang, the Teacher Science Foundation of Changshu Institute of Technology under grant No. KY2009024.

References

[1] J D. Owens, D. Luebke, N. Govindaraju et al. A Survey of General-Purpose Computation on Graphics Hardware. Computer Graphics Forum, Vol. 26, No. 1, pp. 80-113, 2007.

[2] L. Marziale, G. G. Richard III, and V. Roussev. Massive Threading: Using GPUs to Increase The Performance of Digital Forensics Tools. Digital Investigation, pp. S73-S81, 2007.

[3] J. Daemen, V. Rijmen. The Design of Rijndael: AES−The Advanced Encryption Standard. New York, USA: Springer-Verlag, 2002.

[4] Q. Huang, Z. Huang, P. Werstein, and M. Purvis, 2008. GPU as a General Purpose Computing Resource. In Proceedings of the 2008 Ninth international Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2008). Dunedin, New Zealand, December 2008, pp. 151-158.

[5] R. Suda, T. Aoki, S. Hirasawa, A. Nukada, H. Honda, and S. Matsuoka. Aspects of GPU for General Purpose High Performance Computing. In Proceedings of the 2009 Asia and South Pacific Design Automation Conference (ASP-DAC 2009). Yokohama, Japan, January 2009, pp.216-223.

[6] 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.

[7] S. Arul, M. Dash, M. Tue and N. Wilson. Hierarchical Agglomerative Clustering Using Graphics Processor with Compute Unified Device Architecture, In Proceedings of International Conference on

Computer Design and Applications (ICCDA 2009), Singapore, May 2009, pp. 556-561.

[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 Informática Teórica 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/cuda_home_cn.html

[12] D. L. Cook, J. Ioannidis, A. D. Keromytis, and J. Luck, CryptoGraphics: Secret Key Cryptography Using Graphics Cards. In Proceedings of Cryptographer's Track at the RSA Conference (CT-RSA 2005), San Francisco, CA, USA, February 2005. pp. 334–350.

[13] NVIDIA. High Performance Computing GPU [EB/OL].(2010-01-09). http://www.nvidia.cn/object/tesla_computing_solutions_cn.html.

[14] S. A. Manavski. CUDA Compatible GPU As an Efficient Hardware Accelerator for AES Cryptography, In Proceedings of IEEE International Conference on Signal Processing and Communication (ICSPC 2007), Dubai, United Arab Emirates, November. 2007, pp. 65–68.

[15] J. Daemen, V. Rijmen. AES Proposal： Rijndael [EB／OL]. (2010-01-09) http:// www.daimi.au.dk/~ivan/rijndael.pdf.