Parallel Implementation and Optimization of SM4 Based on CUDA

Jun Li[1], Wenbo Xie[2], Lingchen Li[2(✉)], and Xiaonian Wu[2]

[1] China Industrial Control Systems Cyber Emergency Response Team, Beijing 100000, China
[2] Guangxi Key Laboratory of Cryptography and Information Security, Guilin University of Electronic Technology, Guilin 541004, China

Abstract. SM4 is the 128-bit block cipher used in WAPI standard in China, which has a strong security and flexibility. In this paper, the rapid implementation of SM4 is given. Based on the characteristics of CUDA (Compute Unified Device Architecture), a CPU-GPU (Central Processing Unit-Graphics Processing Unit) scheme of SM4 is proposed by exploiting the structure property. Moreover, this scheme is further improved by introducing the page-locked memory and CUDA streams. The results show that: SM4 optimized parallel implementation under GPU can obtain with a speed-up ratio of 89, and the throughput can reach up to 31.41 Gbps.

Keywords: Block cipher · SM4 · Parallel computing · CUDA · GPU

1 Introduction

With the development of Internet services such as cloud computing, big data, and ecommerce, the information security is attracted more and more attention. And cryptography is the basis method to protect the security of information. The SM4 block cipher has been applied in the WAPI (Wireless LAN Authentication and Privacy Infrastructure) wireless network standardwhich is widely used in China. During to the high computational complexity, SM4 is not suitable for occasions which required the encryption algorithm with a high speed. So, the rapid implementation of SM4 is worth to study further. At present, there are two main platforms for fast implementation of block ciphers: GPU (Graphics Processing Unit) and FPGA (Field Programmable Gate Array). However, FPGA is more difficult to develop and has a longer cycle time than GPU. But

This article is supported in part by the National Natural Science Foundation of China (61872103, 62062026), the Key Research and Development Plan of Guangxi (guike AB18281019), the Innovation Research Team Project of Guangxi (2019GXNSFGA245004), the scientific research project of young innovative talents of Guangxi (guike AD20238082), and the Science Research Foundation of Guilin University of Electronic Technology (UF19050Y).

B. Chen and X. Huang (Eds.): AC3 2021, LNICST 386, pp. 93–104, 2021.
https://doi.org/10.1007/978-3-030-80851-8_7

with the feature of the multi-thread, high-bandwidth, GPU is very suitable to do parallel computation to accelerate the speech of encryption and decryption of block ciphers. CUDA (Compute Unified Device Architecture) is a general-purpose device architecture for GPU computing launched by NVIDIA in 2007, which provides a convenient platform for GPU parallel implement of cryptographic algorithms.

As a professional graphics processing tool, GPU was originally used in the field of computer graphics, then Kedem et al. [1] used GPU to quickly crack the key of the UNIX system, which made GPU begin to apply to cryptographic problems. With the introduction of CUDA, GPU has also developed in the field of general computing. Many symmetric cryptographic algorithms have begun to achieve parallel acceleration on CUDA. Manavski et al. [2] first used CUDA to accelerate the AES in 2007, and proposed a T table to improve the performance of the cipher. Since then, most of the teams' work mostly follows their scheme. In 2008, Harrison et al. [3] completed the CTR mode of AES on CUDA, and discussed how to arrange the serial and parallel execution of the block cipher on the GPU. In 2013, Zhou et al. [4] aimed at the shortcomings of the ECB mode of AES and improved the results based on CUDA. In 2014, Mei et al. [5,6] proposed a new fine-grained benchmarking method and applied it to two popular GPU architectures: Fermi and Kepler, and discussed previously unknown features of their memory hierarchy; they also studied the impact of library conflicts on the latency of the shared memory access. In 2017, Fei et al. [7] accelerated AES under CUDA, and the acceleration obtained was about 3 times faster than that of Manavski, and for the first time he analyzed the performance from the perspective of acceleration efficiency. In the same year, Wang et al. [8] compared the performance of SM4 on two different platforms, and got a speed-up ratio of 64.7. In 2020, Li et al. [9] implemented SM4 under CUDA and discussed different thread block sizes and plaintext block sizes, and obtained a speed-up ratio of 26. In addition, the rapid implementation of other block ciphers are also been proposed based on GPU, such as SHA3 [10], IDEA, Blowfish, and Threefish [11] on GPU.

In this paper, we analyze the property of SM4 and the corresponding hardware to improve the speed of implementation on GPU. We propose a parallel scheme based on the characteristics of CUDA programming technology and further improve it. The speed-up ratio and throughput of parallel scheme of SM4 with different input block sizes are obtained. The results show that the speed-up ratio of our scheme can be improved to 89, and the throughput can reach up to 31.41Gbps.

2 Preliminaries

2.1 CUDA

CUDA is a new computing unified device architecture to applicate the GPU as a data parallel computing device, without the graphics API mapping. It is suitable for multi-threaded programming models. The multi-tasking mechanism

of the operating system can be used to call multiple CUDA's cores to run simultaneously. The thread-level parallelism is the basic idea of CUDA, so that these threads are dynamically called and executed. The CUDA uses the CPU as the host and the GPU as the device, allowing the CPU to call the GPU and the GPU to run the computationally intensive part of the program. The CUDA programming architecture is shown in Fig. 1.

There are 6 common memory structures in CUDA: register, shared memory, local memory, constant memory, texture memory and global memory. The first two are on-chip memory, and the access speed is faster than the last four [12]. Figure 2 is the memory structure of CUDA, each type of memory has its own scopes and characteristics. Register is the fastest storage unit, but the number of it is small. Variables that cannot be allocated to register will be overflowed into local memory. Register and local memory are private to each thread. Shared memory is the public memory of each block, available for each thread of the same block, but it may cause memory conflicts. Constant memory and texture memory are read-only memory. Because texture memory is mainly used for graphic processing, it is rarely used in cryptography, while constant memory is a special cache which is suitable for data processing in cryptography. Global memory is the largest memory unit in GPU, all threads can access it, but the speed of it is also the slowest [13,14]. Therefore, for the storage of data variables of different sizes, it is necessary to consider the various characteristics of the memory structure and select the appropriate storage structure.

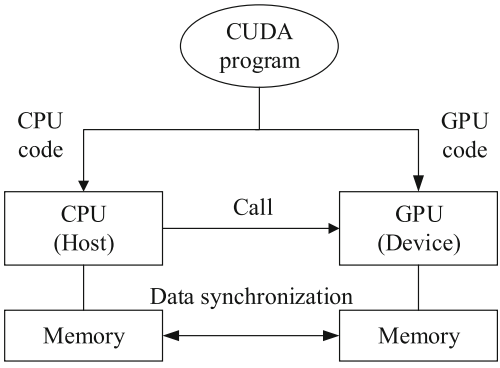

Fig. 1. The programming architecture of CUDA.

2.2 A Brief Description of SM4

SM4 was first proposed in 2006 and became China's national cryptographic industry standard in 2012 [15], which is adopted with a unbalance general feistel structure. The block size and key size are both 128-bit. And the number of iterative rounds is 32. The structure of SM4 is shown in Fig. 3.

Encryption Algorithm. Let (X_0, X_1, X_2, X_3) be the 128-bit input, where $X_i (i= 0, 1, 2, 3)$ means a 32-bit word, so the function F is

$$F(X_0, X_1, X_2, X_3, rk)=X_0 \oplus T(X_1 \oplus X_2 \oplus X_3 \oplus rk) \tag{1}$$

The round function F includes a linear transformation L and a nonlinear transformation τ, namely $T(.) = L(\tau(.))$. The nonlinear transformation τ is composed of 4 parallel S-boxes. Let $A = (a_0, a_1, a_2, a_3)$ and $B = (b_0, b_1, b_2, b_3)$ be the input and output of τ respectively, which can be defined as,

$$\tau(A) = (b_0, b_1, b_2, b_3) = (Sbox(a_0), Sbox(a_1), Sbox(a_2), Sbox(a_3)) \tag{2}$$

And the linear transformation L is defined as,

$$L(B) = B \oplus (B<<<2) \oplus (B<<<10) \oplus (B<<<18) \oplus (B<<<24) \tag{3}$$

After 32-round iteration, the output undergoes a reverse order transformation to achieve the ciphertext. The decryption algorithm is similar to the encryption algorithm, except that the decrypted round key is used in the reverse order of encryption.

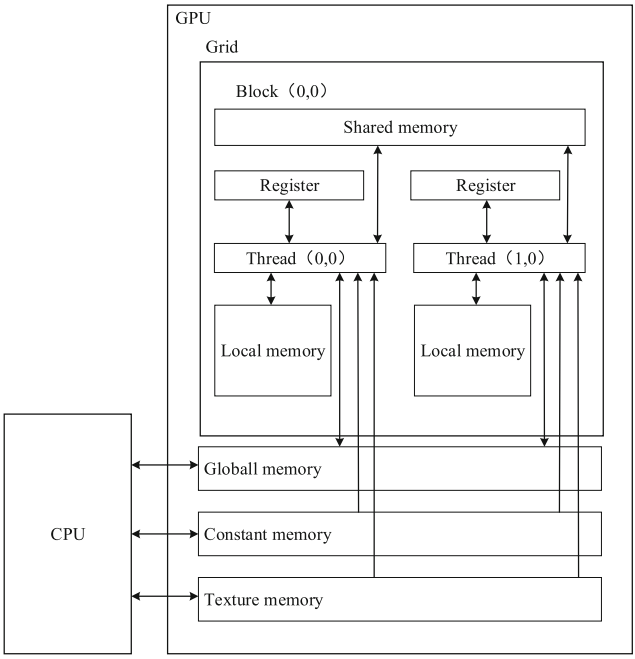

Fig. 2. The memory architecture of CUDA.

Key Schedule. Let $MK = (MK_0, MK_1, MK_2, MK_3)$ be the 128-bit master key of SM4, where $MK_i (i = 0, 1, 2, 3)$ is a 32-bit word. The round key is expressed as $rk_i (i = 0, 1, 2, ..., 31)$, which the size is 32-bit. The key schedule of SM4 can be defined as,

$$(K_0, K_1, K_2, K_3) = (MK_0 \oplus FK_0, MK_1 \oplus FK_1, MK_2 \oplus FK_2, MK_3 \oplus FK_3) \quad (4)$$

$$rk_i = K_{i+4} = K_i \oplus T'(K_{i+1} \oplus K_{i+2} \oplus K_{i+3} \oplus CK_i) \quad (5)$$

where $FK_i (i = 0, 1, 2, 3)$ and $CK_i (i = 0, 1, 2, ..., 31)$ are the constants. T' is almost the same as T with different linear transformation defined as,

$$L'(B) = B \oplus (B <<< 13) \oplus (B <<< 23) \quad (6)$$

3 Parallel Design of SM4

The basic allocation scheme of SM4 is as follow:

(1) Parallel Granularity, Thread and Grid Allocation
Through the comparison of fine-grained and coarse-grained designs, it is found that the performance of coarse-grained thread scheduling is better, that is, a single thread processes 16 bytes. The allocation of the number of threads is preferably a multiple of 32 [16]. Otherwise, the resources will be wasted. During the upper bound of threads is 1024, we test that 512 threads per block can achieve the best performance. And we set the size of grid is: the size of input/ the number of threads per block/the size of the plaintext block.

(2) Data Allocation
The simplest way is to put the plaintext, S-box, FK and CK into the global memory. However, the access speed of the global memory is very slow, while the storage capacity is large. So the global memory is suitable to store the plaintext. Through comparison tests, it is found that storing the plaintext into the global memory, and the S-box, FK and CK into the constant memory have better performance.

(3) Round Key Allocation
When using multiple-round-key, the round keys used by each GPU thread are different, and the size of keys is large. So GPU is used to generate the round keys and store them in the global memory. When using single-round-key, CPU is chosen to pre-calculate this round key to improve the speed of execution.

So the whole parallel process of SM4 shown in Fig. 4 is: First, randomly generate the input and key according to the length of the input and key that you need, and allocate memory, allocate the size of block and grid. Then, the round key will be generated by GPU or CPU according to the multi-key or single-key mode. Finally, the data is copied to GPU and the kernel function is called for encryption and decryption operations, and then the result is copied back to CPU.

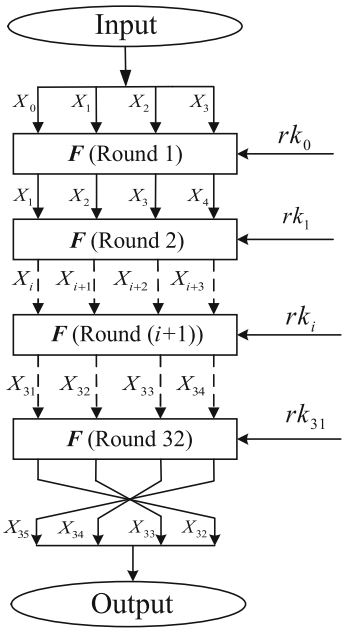

Fig. 3. The structure of SM4.

4 Experiment and Discussion

The environment of experiment in this paper is: GPU: GeForce GTX TITAN X. CPU: Inter(R) Xeon(R) E5-2643 v4 @3.40 GHz. OS: Ubuntu18.04.2 LTS, 64 bits. Version of GCC: 7.5. Version of CUDA: 10.1.

4.1 Basic Experiment

The performance of SM4 based on CPU and GPU is tested under the same size of input. Performing 5 tests, then the average value is computed to reduce the error. All tests consider a single-round-key as the round key. And we use speed-up ratio S to express the improvement of parallel performance of SM4. The calculation equation is defined as,

$$S = T_S/T_p \tag{7}$$

where T_S is the encryption time of CPU, T_p is the encryption time of GPU. The preliminary acceleration of GPU vs. CPU is shown in Fig. 5, and the speed-up ratio can reach up to 53.93.

It can be seen from Fig. 5 that when the size of input is small, the acceleration ratio of the GPU is not very obvious, and the increase is slower. The reason is that the data transmission between the GPU and CPU wastes time when the data is not so big. At the same time, GPU cannot call enough computing units

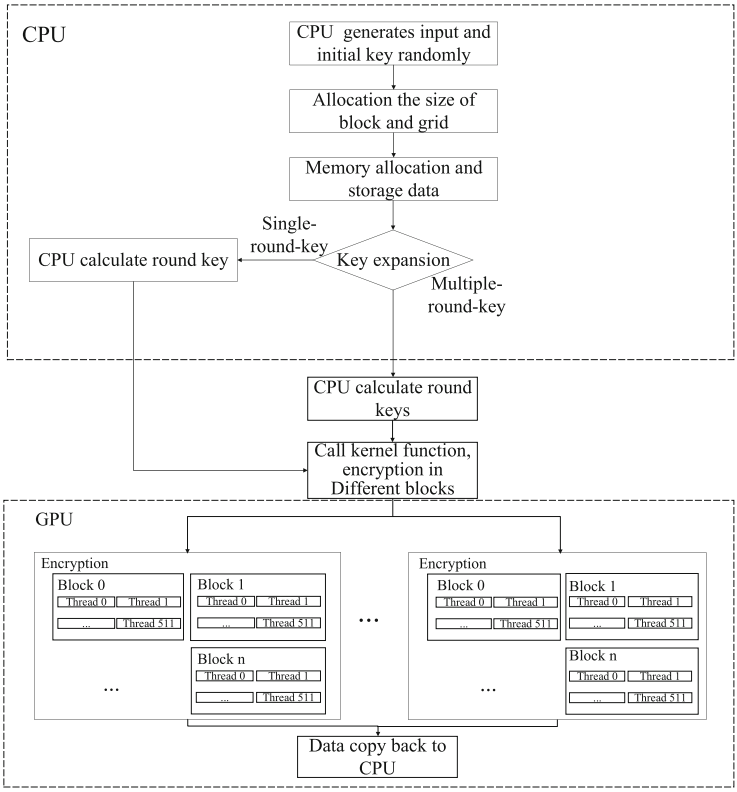

Fig. 4. The parallel process of SM4.

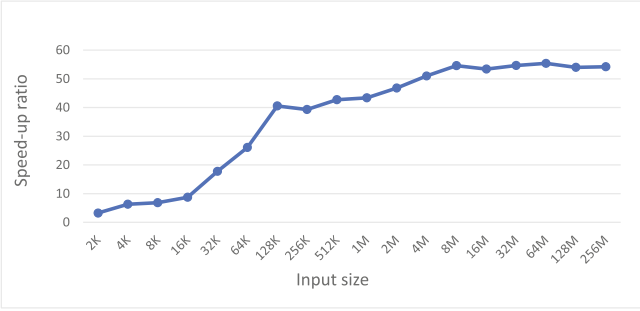

Fig. 5. The structure of SM4.

to perform parallel operations. So the improvement of performance based on GPU is not obvious. However, as the size of input increases, the speed-up ratio increases rapidly. But when the size of input reaches a certain value, that's 256 KB, the growth of the speed-up ratio begins to level off again and finally stabilizes at about 50. This shows that the acceleration of GPU is not unlimited growth, it will also be limited by its own hardware.

In order to make better use of GPU performance, the next section we will take measures to optimize operations

4.2 Performance Optimization

Data Transmission Optimization. For data transmission optimization, the measures we adopted are: use page-locked memory, also known as pinned memory, instead of pageable memory. By default, the CPU allocates pageable host memory. The GPU cannot directly access data in the pageable memory because the memory data may be shifted or destroyed, it cannot safely use the physical address of the host memory. The GPU needs to perform two-part copy operation to obtain data from the pageable memory, first copy the data to the temporary page-locked memory, and then copy the data from the page-locked memory to the GPU. But by using page-locked memory to allow GPU to access the memory directly, the time of data transmission can be reduced. Figure 6 shows the process of page-locked memory transfer and pageable memory transfer.

Although the allocation and release cost of page-locked memory is higher than that of pageable memory, it can provide higher transmission throughput for large-scale data transmission. Figure 7 shows the performance comparison between pageable memory and page-locked memory. It can be seen that when the size of input is small, the acceleration effect is not obvious. The reason is that the initial cost of page-locked memory is basically the same as the benefit from processing data. However, when transmitting more than 64 KB input, as the size of input increases, the cost of the initial overhead becomes smaller and smaller, and the page-locked memory gains obvious. The final optimized performance increased by 40.73%.

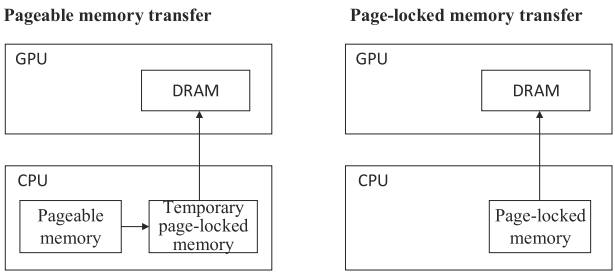

Fig. 6. Two kinds of memory data transmission.

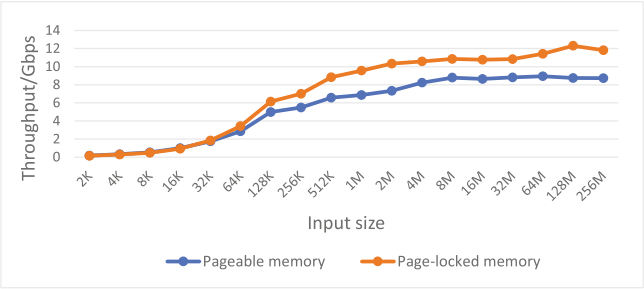

Fig. 7. Performance comparison between pageable memory and page-locked memory.

Parallel Streams Overlapped Execution Optimization. For the optimization of the sequential execution of data transmission and kernel operations, the solution we proposed is: using CUDA streams to overlap data transmission and kernel operations. The basic structure of the SM4 encryption program consists of 3 steps: ① copy the plaintext and key from the host to the device. ② Perform the encryption operation. ③ Return the ciphertext from the device to the host. In order to achieve over-lapping operations, instead of copying the input data to the GPU all at once, the idea of "divide and conquer" is adopted to divide the input data into multiple subsets. Then each sub-problem is independent and can be separately arranged in the CUDA stream for calculation. The output transmission of one stream overlaps the core calculation of another stream. The sequential execution and the overlapping streams execution are shown in Fig. 8.

In order to make better use of the two replication engines of the GPU, we use 8 streams and one queue for each stream for overlapping operations. Figure 9 is a performance comparison between sequential execution and overlapping streams execution. As a result, the optimized performance increased by 33.3% after two rounds of optimization operations. The performance of the original GPU parallel algorithm has been improved by twice. In the end, compared to SM4's CPU serial algorithm, GPU parallel algorithm has nearly 90 times the performance improvement, as shown in Fig. 10.

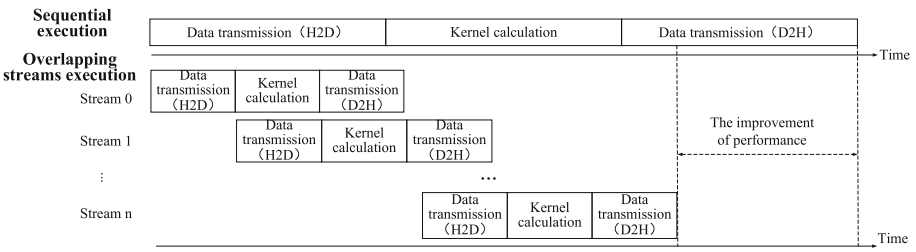

Fig. 8. Sequential execution and overlapping streams execution.

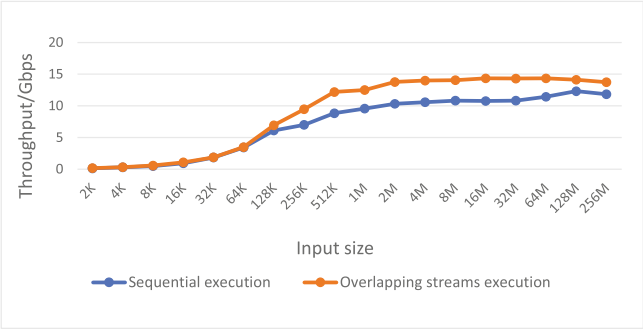

Fig. 9. Performance comparison between sequential execution and overlapping streams execution.

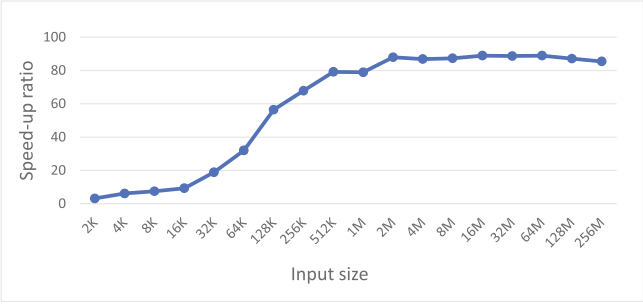

Fig. 10. Optimized speed-up ratio.

4.3 Analysis of Results

The speed-up ration in our scheme is 89 which is higher than reference [8] and [9] respectively as shown in Table 1. However, considering release time, core parameters, operating frequency, bandwidth and other factors of the GPU, we sort the performance of the GPU of the three articles from high to low: our article, [8,9]. In other words, the high speed-up ratio in our article may be caused by the better performance of the GPU itself. Therefore, we compare our result of throughput with [17,18], we can see that in Table 2. The GPU used in [17,18] are Nvidia GeForce RTX 2080 and NVIDIA GTX 1080 respectively, GPUs' performances are better than us. We find that the throughput of [17] can reach up to 27.64 Gbps, and the throughput of our result can reach 31.41 Gbps, which has a performance improvement of 1.13 times. It shows that our work does effectively improve the performance of SM4. Although the throughput is up to 76.8 Gbps in [18], our scheme is still competitive when the price cost is considered.

After analyzing the experiment, it is found that the main factors affecting GPU performance are as follows: ① The data transmission between the host and

the device takes a long time. ② The sequential execution of data transmission and kernel operations causes time wasted.

Table 1. Compare with the speed-up ratio.

Input size	GeForce GTX TI TAN X(Us)	GeForce GT 240M([8])	Quadro 600 ([9])
32K	18.96	6.34	4.14
1M	78.94	29.81	23.38
8M	87.31	39.71	25.79
32M	88.69	40.67	25.80

Table 2. Compare with the speed-up ratio of previous works.

	GeForce GTX TI TAN X(Us)	GeForce RTX 2080 ([17])	NVIDIA GTX 1080 ([18])
Throughput (Gbps)	31.41	27.64	76.80
GPU's performance	Good	Better	Best
GPU's price	Medium	High	Higher

5 Conclusion

Based on the CUDA under the Linux OS, we implement the SM4 in parallel and explore the comparison of the performance of the CPU and GPU under the same plaintext in the range of 2 KB to 256 MB under the CPU and GPU. The results show that the GPU performance of parallel implement of SM4 is 88–89 times faster than CPU. Moreover, through introducing the page-locked memory and CUDA streams, the performance of our parallel algorithm can be further improved with the throughput 31.41 Gbps.

References

1. Kedem, G.-Y.: Brute force attack on UNIX passwords with SIMD computer. In: Proceedings of the 8th Conference on USENIX Security Symposium, pp. 93–98. USENIX Association, Washington, D.C. (1999)
2. Manavski, S.A.: CUDA compatible GPU as an efficient hardware accelerator for AES cryptography. In: 2007 IEEE International Conference on Signal Processing and Communications, pp. 65–68. Springer, Dubai (2007)
3. Harrison, O., Waldron, J.: Practical symmetric key cryptography on modern graphics hardware. In: Proceedings of the 17th USENIX Security Symposium, pp. 195–210. USENIX Association, San Jose (2008)

4. Xia, C., Zhou, D., Zhang, K., Liu, C., Chu, X.: CUDA based high-efficiency implementation of AES algorithm. **30**(6), 1907–1909 (2013). (in Chinese)
5. Mei, X., Zhao, K., Liu, C., Chu, X.: Benchmarking the memory hierarchy of modern GPUs. In: Hsu, C.-H., Shi, X., Salapura, V. (eds.) NPC 2014. LNCS, vol. 8707, pp. 144–156. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44917-2_13
6. Mei, X., Chu, X.: Dissecting GPU memory hierarchy through microbenchmarking. IEEE Trans. Parallel Distrib. Syst. **28**(1), 72–86 (2017)
7. Fei, X., Chu, X., Yang, W.: Research and implementation of GPU parallel AES algorithm based on CTR model. J. Chin. Comput. Syst. **36**(3), 529–533 (2015). (in Chinese)
8. Wang, D., Chen, D.: High speed implementation of SM4 encryption algorithm based on CUDA. J. Shijiazhuang Inst. Railway Technol. **16**(1), 59–63 (2017). (in Chinese)
9. Wang, D., Chen, D.: Parallel implementation of SM4 algorithm on GPU. Inf. Netw. Secur. **20**(6), 36–43 (2020). (in Chinese)
10. Yang, J., Wang, W., Xie Z., Han, J., Zeng X.: Parallel implementations of SHA-3 on a 24-core processor with software and hardware co-design. In: IEEE 12th International Conference on ASIC, pp. 953–956. IEEE Computer Society, Guiyang (2017)
11. Cheong, H., Lee, W.: Fast implementation of block ciphers and PRNGs for Kepler GPU architecture. In: International Conference on It Convergence and Security, pp. 1–5. IEEE Computer Society, Los Alamitos (2015)
12. NVIDIA Corporation, https://docs.nvidia.com/cuda/cuda-c-programming-guide. Accessed 23 Sep 2020
13. Jason, S., Edward, K.: CUDA by Example: An Introduction to General-Purpose GPU Programming, 1st edn., Machinery Industry Press, Beijing (2011). (in Chinese)
14. Cheng, J.: Professional CUDA C Programming, 2nd edn., Machinery Industry Press, Beijing (2017). (in Chinese)
15. China's National Cryptography Administration. http://www.sca.gov.cn/sca/c100061/201611/1002423/files/330480f731f64e1ea75138211ea0dc27.pdf. Accessed 18 Nov 2016. (in Chinese)
16. Ma, J., Chen, X., Xu, R., Shi, J.: Implementation and evaluation of different parallel designs of AES using CUDA. In: IEEE Second International Conference on Data Science in Cyberspace, pp. 606–614. IEEE Computer Society, Shenzhen (2017)
17. Zhang, C.: Design and implementation of parallel SM4-GCM based on CUDA. Xi'an University of Electronic Ccience and Technology **2**(1), 104 (2019).(in Chinese)
18. Cheng, W., Zheng, F., Pan, W., Lin, J., Li, H., Li, B.: High-performance symmetric cryptography server with GPU acceleration. In: Qing, S., Mitchell, C., Chen, L., Liu, D. (eds.) ICICS 2017. LNCS, vol. 10631, pp. 529–540. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89500-0_46