Journal of Information Security and Applications 47 (2019) 235–245

Contents lists available at ScienceDirect

Journal of Information Security and Applications

journal homepage: www.elsevier.com/locate/jisa

Efficient implementation of lightweight block ciphers on volta and pascal architecture

Pei Li, Shihao Zhou, Bingqing Ren, Shuman Tang, Ting Li, Chang Xu, Jiageng Chen*

Central China Normal University, Wuhan 430079, China

A R T I C L E I N F O

Article history:
Available online 24 May 2019

Keywords:
Lightweight blockcipher
LED
PRESENT
Piccolo
GPU

A B S T R A C T

Lightweight block ciphers based on symmetric cipher are tailored to addressing security issues for highly constrained Internet of things devices. In this article, we explore general software implementations of lightweight ciphers on GPU architectures, with a special focus on LED, Piccolo and PRESENT. First, we implement the lightweight block ciphers using lookup table based technique. Then we analyze the effect of different factors on the encryption performance, such as the size of lookup table, the parallelism level, the use of different type of memory and the idle state caused by data transmission. We evaluated the three lightweight block ciphers on Nvidia Tesla V100 and Nvidia GTX1060 GPUs. Finally, the experimental result shows a great improvement on all the investigated ciphers compared to the unoptimized implementations.

1. Introduction

The connected Internet of things (IoT) devices have become ubiquitous in our society. Since IoT is always related to user's daily life or work, the proper encryption methods are required to ensure the security of the data generated by IoT devices. The lightweight cryptography is tailored to IoT devices which are highly constrained by the limitations of memory, storage, connectivity, processing power and energy consumption [1]. And it can potentially have wide applications on various areas such as vehicular ad hoc networks, authentication schemes and so on [2,3]. Since the lightweight cryptography usually consists of a large amount of homogeneous computation, the use of hardware accelerators such as GPUs has been considered to be one of the omost important methods to achieve high-speed data encryption.

GPU (Graphics Processing Unit), originally designed for processing graphics, has recently evolved into general purpose massive parallel computing device. Compared to CPU, GPU has more ALUs, simpler controller functions and fewer caches. A large number of ALUs of GPU are arranged in matrix, which can deal with a large number of parallel but relatively simple processing tasks. New programming frameworks, such as OpenCL and CUDA, enable software developers to implement general applications on GPU architecture. Many studies claim that GPUs deliver at least 10x speedups over multi-core CPUs on data parallel applications [4,5].

This article exploit multiple optimization techniques to achieve high performance for three lightweight block ciphers(LED, PRESENT and Piccolo) on GPU architectures. Different types of memory where the lookup tables are stored are evaluated and analysed for their effect on encryption performance. The optimal computational parallel granularity have also been experimented on different GPU. The use of overlapping technique, which hides the time of data transfers in calculations, further improve the throughput of each block cypher.

The rest of the article is organized as follows. Section 2 briefly introduces the back ground of lightweight block cipher, GPU Architectures and related work. Different optimization techniques are presented in Section 3. The experimental results are presented and analysed in Section 4. Finally Section 5 concludes this work.

2. Background knowledge

2.1. Lightweight block ciphers

In the Internet of Things, the computing power of sensors, smart cards and other micro-devices is often limited. As a kind of special block cipher algorithms, therefore, the lightweight block cipher algorithm has the characteristics of relatively short packet length and key length, relatively simple algorithm structure and encryption method, and strives to provide sufficient security guarantee while effectively run quickly in a resource-constrained environment. Up to now, a large number of lightweight block cipher algorithms have been proposed. In the case of limited resources, they each have different efficiency advantages in software

* Corresponding author.
E-mail address: jiageng.chen@mail.ccnu.edu.cn (J. Chen).

https://doi.org/10.1016/j.jisa.2019.04.006
2214-2126/© 2019 Published by Elsevier Ltd.

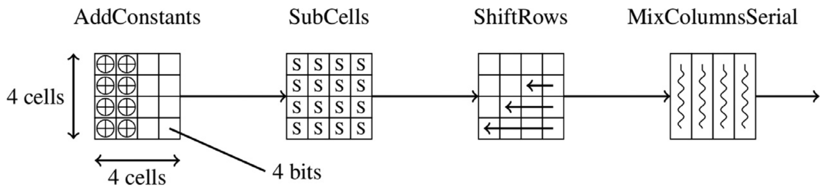


Fig. 1. Four steps of a single round of LED.

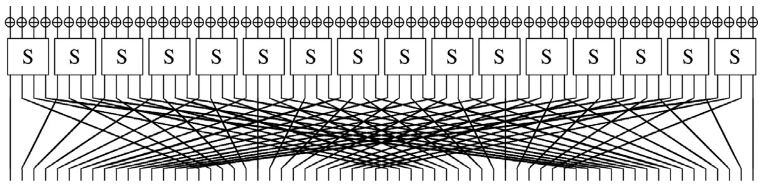


Fig. 2. Three steps of a single round of PRESENT.

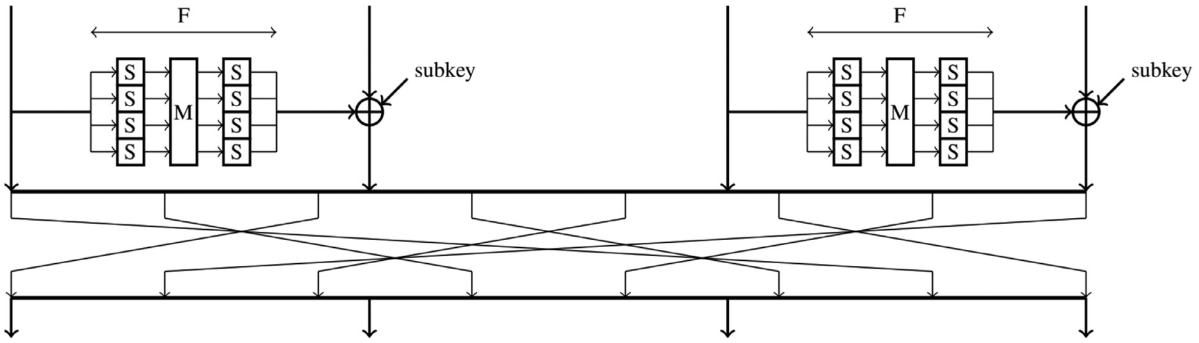


Fig. 3. Three steps of a single round of Piccolo.

and hardware. These algorithms include LED [6], PRESENT [7] and Piccolo [8].

LED. LED was proposed at Workshop on Cryptographic Hardware and Embedded Systems (CHES2011), and it is a kind of 64-bit cipher with 64–128 bit key and 32 or 48 rounds. The round function of LED uses an AES-like design and LED cipher re-use the PRESENT Sbox. LED cipher state is arrannged in a (4×4) grid where each nibble represents an element from GF(2^4) with the underlying polynomial for field multiplication given by $X^4 + X + 1$. As shown in Fig. 1, one round is composed of four steps: AddConstants, SubCells, ShiftRows and MixColumn. Firstly, AddConstants function just XORs a fixed constant to the first column and a round-dependent constant to the second column of the internal state. The SubCells function applies the 4-bit Sbox to every nibble with S[X] = 0xC, 0x5, 0x6, 0xB, 0x9, 0x0, 0xA, 0xD, 0x3, 0xE, 0xF, 0x8, 0x4, 0x7, 0x1, 0x2. The ShiftRows function shifts left by i position all the nibbles located in row i. In the end, the linear function MixColumns applies a MDS diffusion matrix M with

$$\begin{bmatrix} 4 & 1 & 2 & 2 \\ 8 & 6 & 5 & 6 \\ B & E & A & 9 \\ 2 & 2 & F & B \end{bmatrix}$$

to every column of the state independently.

PRESENT. At CHES2007, PRESENT was proposed as a 64-bit block cipher that applies 31 rounds for both its 80 and 128-bit key versions. As shown in Fig. 2, One round is composed of three steps: addRoundKey, sBoxLayer and pLayer. The first step just XORs the incoming subkey to the internal state and the second step applies the 4-bit Sbox to all nibbles with S[X] = 0xC, 0x5, 0x6, 0xB, 0x9, 0x0, 0xA, 0xD, 0x3, 0xE, 0xF, 0x8, 0x4, 0x7, 0x1, 0x2. The last step is pLayer function, it is a bit permutation where a bit located at position i is moved to position $j = i \cdot 16 \bmod 63$ when $i \in \{0, \ldots, 62\}$ and $j = i$ if $i = 63$. After the last round, a last addRoundKey layer is performed.

Piccolo. Piccolo is a 64-bit block cipher that applies respectively 25 and 31 rounds for the 80 and 128-bit key versions. The round function is a 4-line type-II generalized Feistel network variant. Therefore, the internal state can be considered as four 16-bit branches. As shown in Fig. 3, One round is composed of three steps. The first function applies a transformation F to the first branch and third branch respectively and XORs the result to the second branch and fourth branch, respectively. Next step is that two incoming 16- bit subkeys are XORed to the second and fourth branches respectively. The last step is that a permutation on the nibble position is performed, where a nibble at position i is moved to position T[i] with T = [4, 5, 14, 15, 8, 9, 2, 3, 12, 13, 6, 7, 0, 1, 10, 11]. The 16-bit function F itself applies a 4-bit Sbox to every nibble with S[X] = 0xE, 0x4, 0xB, 0x2, 0x3, 0x8, 0x0, 0x9, 0x1, 0xA, 0x7, 0xF, 0x6, 0xC, 0x5, 0xD, then multiplies the current vector by an MDS diffusion matrix M with

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}.$$

After this, function F itself applies the 4-bit Sbox to every nibble as well. In the end, two 16-bit whitening keys are incorporated to the first and third branches respectively, at the beginning and at the end of the ciphering process.

2.2. GPU Programming

Computing Unified Device Architecture (CUDA) is a parallel computing platform and programming model released by NVIDIA. The basic idea of CUDA is to do SIMD on a grand scale. Each CUDA

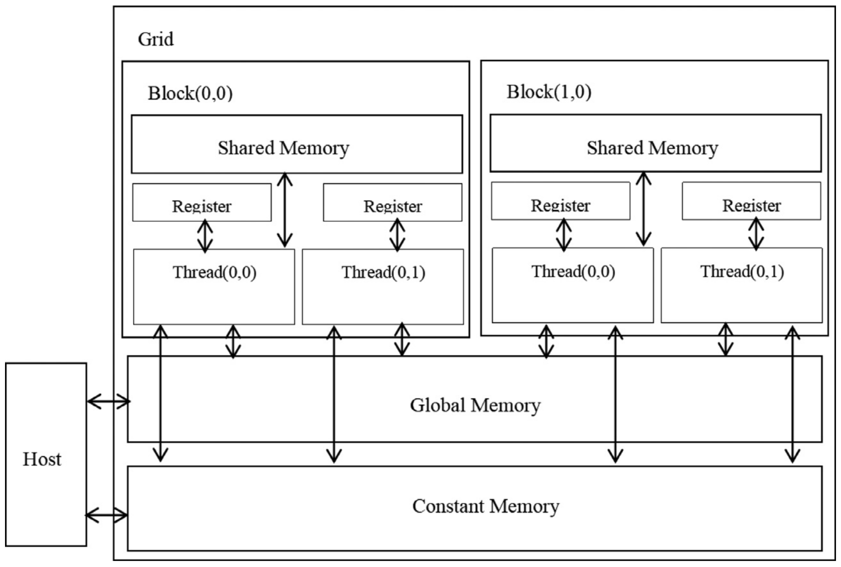


Fig. 4. The memory model of CUDA.

thread executes the same code (CUDA Kernel). Multiple threads can be grouped into a block. The block of threads will be assigned into a Streaming Multiprocessors(SM) for execution. The maximum number of threads per block depends on the architecture of GPU.

Fig. 4 shows the hierarchy of CUDA memory. CUDA supports a variety of memory types, including registers, shared memory, global memory, and constant memory. Developers can use, therefore, different memory combination to speed up the execution of the kernel.Shared memory and registers on CUDA devices vary widely in their functionality and access costs. Although, the processor still needs to load memory when accessing data stored in shared memory. However, shared memory has lower latency and higher bandwidth than accessing the global memory because shared memory is located on the chip. The variable __shared__ is used before the variable declaration to indicate that the shared variable residing in the kernel function is saved in shared memory. Global memory and constant memory can exchange data directly with Host. Constant memory provides the device read-only access with short-latency and high-bandwidth.

2.3. Related work

There are several research work focusing on implementation techniques used for optimizing AES using GPU that are mentioned above.

Di Biagio [9] proposed an implementation of CTR mode of operation using Geforce 8800GT. Di Biagio et al. indicated that using on-chip shared memory rather than constant memory to store T-boxes would bring a considerable performance improvement. In addition, they carried out a tentative discuss on parallel processing granularity. Also Chonglei Mei et al. implemented AES encryption on Geforce 9200M in a similar way.

Guo et al. [10] tested the fast implementation of AES algorithm and the performance has been improved by about 50 times when compared to the standard AES algorithm. In addition, using the Intel AES-NI extended instruction sets, the performance has been improved by about 50 times compared with the fast implementation of AES algorithm. In the end, using CUDA and GPU to execute the AES in parallel, and it can improve the performance by about 18 times compared with the fast implementation of AES algorithm.

Lee et al. [11] presented implementation of block ciphers in NVIDIA GTX 980 with Maxwell architecture, they used 16 bytes/thread granularity and proposed a novel method to store the encryption keys in high speed registers and exchange it across threads in same warp using warp shuffle operation.

Dai et al. [12] presented a large polynomial arithmetic library optimized for Nvidia GPUs to support fully homomorphic encryption schemes. Authors putted the library to use to evaluate homomorphic evaluation of two block ciphers: Prince and AES, which show 2.57 times and 7.6 times speedup, respectively, over an Intel Xeon software implementation.

Abdelrahman et al. [13] implemented CUDA AES with high performance. They explored multiple optimization techniques exploited to achieve higher throughput and speedup for the AES algorithm on three different GPU architectures and found that encryption speeds with 207 Gbps on the NVIDIA GTX TITAN X (Maxwell) and 280 Gbps on the NVIDIA GTX 1080 (Pascal) have been achieved by performing new optimization techniques using 32 bytes/thread granularity.

In research of lightweight block ciphers, Benadjila et al. [14] provide three main contributions for lightweight ciphers software implementations on x86 architectures with a special focus on LED, PRESENT and Piccolo. In addition, they obtained the best known table-based implementations for the studied lightweight ciphers. But they have not explored to encrypt the lightweight password on non-x86 platforms. Most of the previous works on GPU platform have been focused on block AES, which also include works such as [15,16] and so on. But the optimization works on the lightweight ciphers in the IoT environment has not been investigated thoroughly.

3. Optimization techniques

This section discusses the optimization strategy of lightweight block encryption.

3.1. Table-based implementations

Tabulating operations is an old but practical method well known by programmers for efficiency purpose which has been applied in many previous researches such as [14]. In order to improve the performance of block cypher, most of the logical operations (such as S-box, ShiftRows, MixColumnsSerial) in one round will be tabulated. Thus, in each round of encryption, only shift, mask operations and table lookup return operations are needed for each

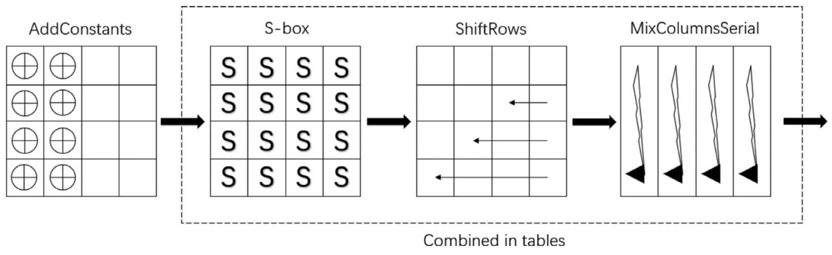


Fig. 5. Table-based implementations in LED.

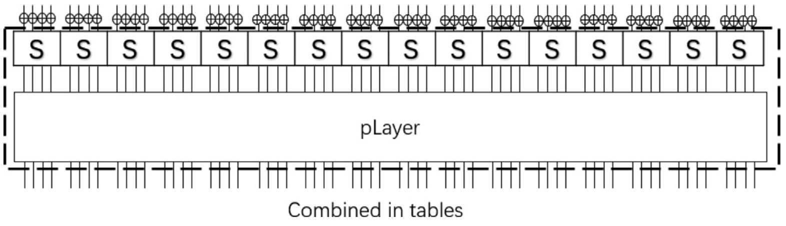


Fig. 6. Table-based implementations in PRESENT.

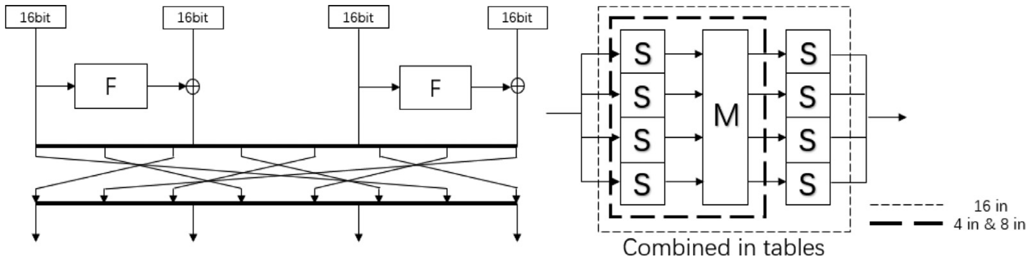


Fig. 7. Table-based implementations in Piccolo.

block cypher. Therefore, table lookup encryption can be performed on almost all blocks at the same time.

In LED, there are two tables, one includes AddConstants and the other includes S-box, ShiftRows and MixColumnsSerial (Fig. 5).

In PRESENT, the table includes logical operations S-box and pLayer (Fig. 6).

In Piccolo, it includes 4-line type-II Feistel network variant, and the F function contains two non-linear S-boxes and one MixColumnsSerial. So, in tables with 4-bit and 8-bit input, there are two tables, one includes S-box and MixColumnsSerial, the other includes S-box (expanded output 64 bit). In the 16 bit input table, the two tables can be combined to form a separate table (Fig. 7).

The key ideas of table construction are follows:

1. Most logical operations should be included in each set of tables.
2. The bigger table will reduce the number of logical operations in the program.
3. Each n-bit table input represents the possibility of 2^n entries, and the actual size of each of the tables should be controlled to fit into the cache of the device for better efficiency.

In the next GPU optimization, in order to avoid redundant computation, we use a fixed round keys for all the light block cipher encryption, which are generated from correct key scheme.

3.2. GPU platform specific optimization

In order to achieve the best performance, all the computing unit on GPU should be fully occupied during the encryption process. Many factors will affect the GPU occupation, such as the use of different memory type, the parallel granularity and the idle state caused by data transfer. Each GPU usually has its own hardware architectures and the optimization strategy could be slightly different from other GPU. Our goals is to find the most suitable parameters for lightweight algorithms in current GPU environment to achieve the fastest encryption speed.

In GPU, the speed of thread invocation is related to the memory type. In theory, shared memory can be quickly invoked in blocks, but the storage space is smaller, while constant memory and global memory are relatively slow, but they have larger storage space. Through quantitative data, fixed tables can test the best choice of these three kinds of memory.

Each GPU device has its own architecture. For example, Nvidia Tesla v100 supports up to 1024 threads per block, but the supported block number is much greater than 1024. The specific time of encryption involves data transmission, data call, data return and so on. Therefore, we guess that different data size should be different in the balance of different threads and blocks, and we can get a general trend. In the test model we designed, threads are used as independent variables, blocks are used as dependent variables, and appropriate values are used as step size to test to find the optimal balance when the amount of data is fixed.

CUDA application can overlap data transfer and computation by using multiple stream. Fig. 8 demonstrates that a memory copy and kernel execution can perform at the same time when two independent streams are used. Considering the actual application time, stream is used to encrypt a long section of plain text which needs to be encrypted, and hide the transmission time in the encrypted time. Firstly, we extend the stream to 2, 4, 8, and test different data quantities to get the whole encryption time. The space allocation time, i.e. the actual transmission time, is calculated as additional parameters. Then, by adjusting the order of code, that is to say, adjusting the order of module usage theoretically, the optimal flow usage scheme is obtained.

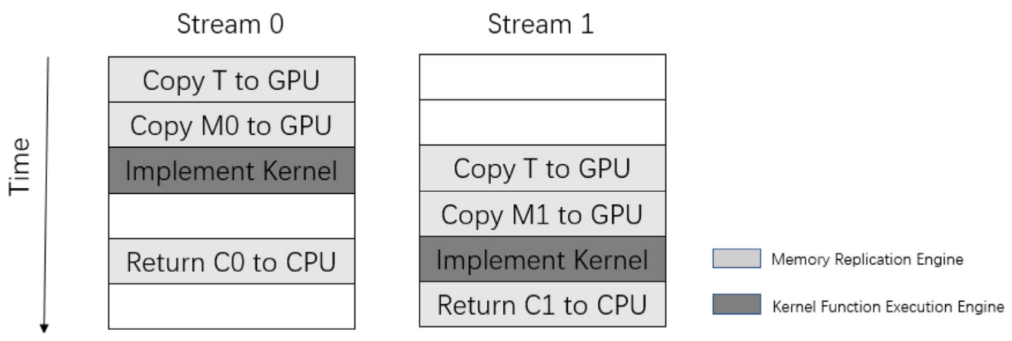


Fig. 8. The time line when there are two streams.

Table 1
The general plan table for the experiment.

Variable factors	Detailed parameters
Cipher type	LED; PRESENT; Piccolo
Table type (input size)	4-bit; 8-bit; 16-bit
Data Size	1MB; 64 MB; 512 MB
Memory type	Global Memory; Constant Memory; Shared Memory
Parallel Granularity	Number of threads per block; Block number
Number of streams	2; 4; 8

Table 2
Configurations of experimental platform.

Name	Device a	Device b
CPU model	Intel Core i5-8300H	
CPU cores	4	
CPU threads	8	
CPU frequency	2300 MHz	
Accelerator Type	NVIDIA GPU	NVIDIA GPU
GPU Architecture	Volta	Pascal
Models	Tesla V100	GeForce GTX 1060
Cores	5120 CUDA Cores	1280 CUDA Cores
GPU Max Clock rate	1671MHz(1.67GHz)	1380 MHz (1.38 GHz)
Memory	16,130 MBytes	6,144 Mbytes
CUDA Driver Version	10.0	10.0

3.3. Practical optimization scheme

In this article, the actual optimization scheme is to combine the technique of tabulate implementation and GPU implementation, and get the optimal conclusion based on the lightweight algorithm. Since there are many variables involved in the experiment, including the size of data, different tables, the choice of GPU parameters, Table 1 is the general plan table which contains variable factors in the experiment.

4. Experiments and performance evaluation

4.1. Experimental platform

Table 2 demonstrates the configuration of experimental platform. Two different GPUs are used to evaluate the optimization strategy for specific GPU architecture. The impact of the use of memory type is evaluated on Nvidia GTX1060 GPU. The impacts of parallel granularity and number of stream are evaluated on Nvidia Tesla V100 GPU. Midrange CPU is used to demonstrate the speedup that can achieve through the implementation of encryption algorithm on GPU architectures.

4.2. GPU optimization ideas

4.2.1. Effect of using different memory

As analyzed in the previous section, Shared memory is Shared by threads in the same block, so the data transmission delay is small and the space is relatively small, which is equivalent to the cache in the CPU. Therefore, the first step of our encryption algorithm focuses on the optimization using Shared memory

We tested all three Memory types with 1 MB data. In the experiment, number of threads per block and number of blocks in the GPU are controlled to be 1024 and 128. In order to reduce the error of the experiment, the encryption process of each experiment is timed for 30 times (including the process of data transmission and encryption, in which the time of the transmission table is not included in the Shared memory), and the average value of time is calculated after the maximum and minimum values are cleared. Finally, the throughput of cryptographic operations under various memory conditions was compared, and the memory with the highest throughput was selected as the result of our first step of lightweight encryption optimization.

In the experiment, we explored the appropriate storage mode for the tables with 4-bit,8-bit and 16-bit input in the three algorithms. Since the tables with 16-bit input exceeded the size of Shared memory, the query of such tables could only rely on Global memory.

The Fig. 9 shows the broken line graph of throughput changes of LED, PRESENT and Piccolo algorithms in Global memory of three tables. LED and Piccolo both had the highest throughput with 16-bit input, followed by 8-bit input and 4-bit input. This shows that for these two algorithms, the larger the table, the faster the encryption speed. The increase of the table makes the read and write instructions become less, and the amount of data to be processed by Kernel becomes smaller. Therefore, as the table with 16-bit input is longer than 8-bit input and 4-bit input, the computation time will be greatly reduced accordingly.

The maximum throughput of PRESENT is 8-bit input, which is related to the Cache size used in the test. When the Cache size is fixed, the Cache hit ratio of the algorithm with a relatively concentrated range of look-ups will be higher, and the performance of the whole algorithm will be improved.The address range of PRESENT look-up table is relatively scattered, and the size of the table is too large when 16-bit input and the amount of read and write is low, so the cache miss is high. Although there are many 4-bit input instructions, the table itself is very small, and the range of read is concentrated, so the cache miss is low when encrypting. That's why PRESENT is different from the other two algorithms.

The following Fig. 10 shows the throughput of the three input modes of the three algorithms in different storage modes. It can be clearly seen that the operation throughput with Shared memory is greater than that with Global memory and Constant memory. The experimental result shows that low latency (Shared memory) does improve encryption performance. This is because Shared memory is equivalent to putting a table into the Cache ahead of time, making it more efficient to read the table directly from the Cache when looking it up. Global memory relies on look-ups and Cache replacement algorithms to fill the contents of the Cache, which have a high probability of miss.

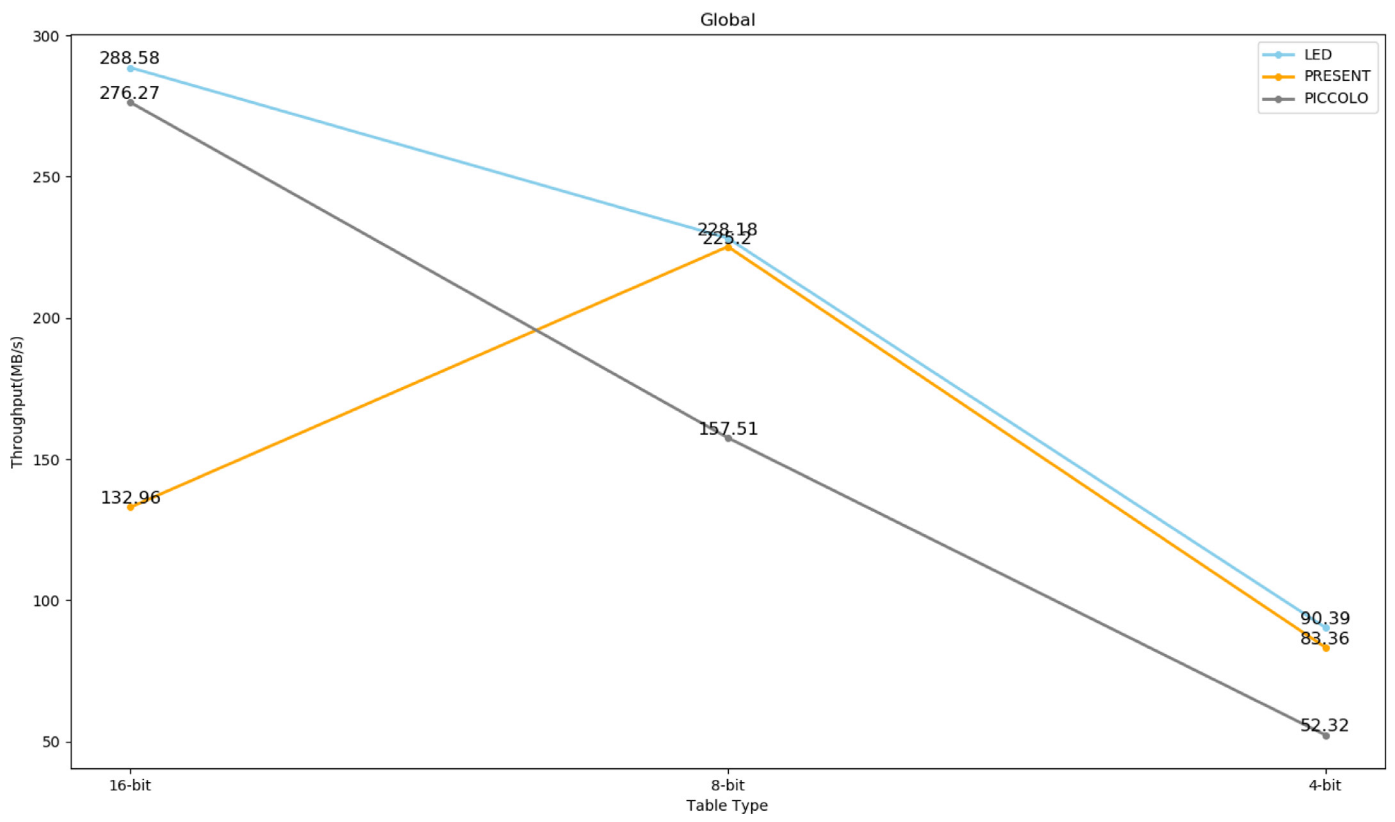
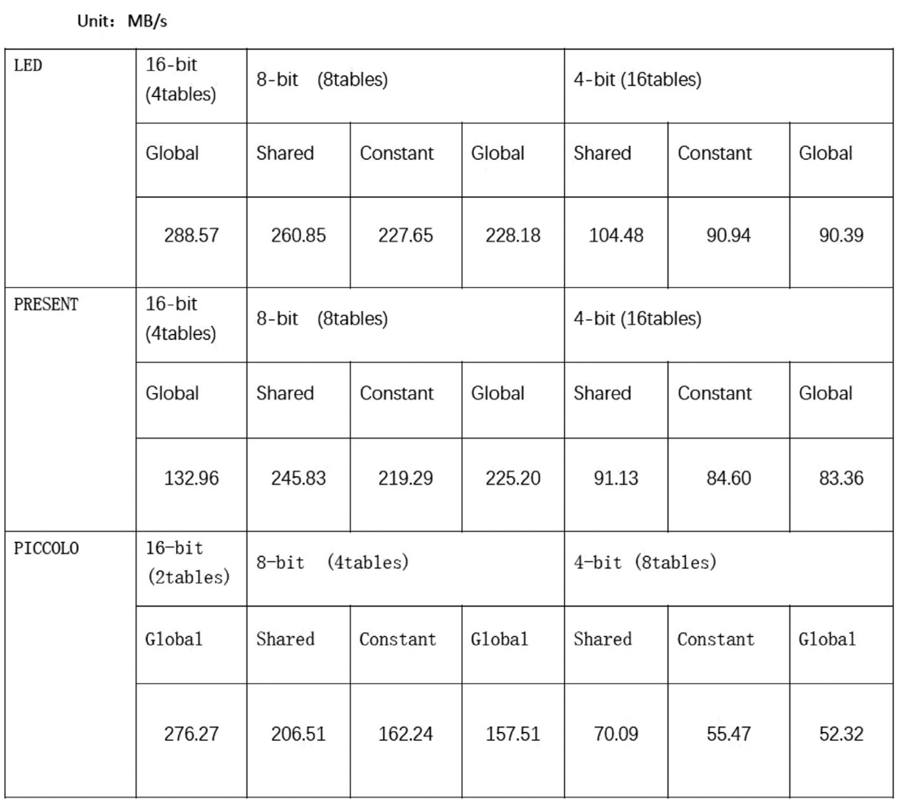


Fig. 9. Performance using Global memory.


Unit: MB/s

LED	16-bit (4tables)	8-bit (8tables)			4-bit (16tables)		
	Global	Shared	Constant	Global	Shared	Constant	Global
	288.57	260.85	227.65	228.18	104.48	90.94	90.39
PRESENT	16-bit (4tables)	8-bit (8tables)			4-bit (16tables)		
	Global	Shared	Constant	Global	Shared	Constant	Global
	132.96	245.83	219.29	225.20	91.13	84.60	83.36
PICCOLO	16-bit (2tables)	8-bit (4tables)			4-bit (8tables)		
	Global	Shared	Constant	Global	Shared	Constant	Global
	276.27	206.51	162.24	157.51	70.09	55.47	52.32

Fig. 10. LED,PRESENT,Piccolo throughput when using different memory.

Table 3
The lower bound of number of threads per block.

LED		
16-bit(4 tables)	8-bit(8 tables)	4-bit(16 tables)
32	256	32
PRESENT		
16-bit(4 tables)	8-bit(8 tables)	4-bit(16 tables)
32	256	32
Piccolo		
16-bit(2 tables)	8-bit(4 tables)	4-bit(8 tables)
1024	256	32

Therefore, we can conclude that Shared memory can be used to optimize the encryption speed of the lightweight algorithm in an appropriate range.

4.2.2. Effect of parallel granularity

In Tesla V100, the maximum number of threads per block is 1024. For lightweight encryption algorithms, such as LED, PRESENT and Piccolo, the block size and the number of blocks satisfy the following formula:

$$number\ of\ blocks = \frac{DataSize}{(BlockSize \times 8)}$$

Since the number of threads running concurrently on a multi-stream processor is 32, the best number of threads/block depends on the program, but should be a multiple of 32. In our experiment, block size will be set at 32, 64, 128, 256, 512 and 1024, respectively. However, in the actual test experiment, the number of threads is also limited by the influence of different size of tables.The lower bound of number of threads per block is shown in Table 3.

In order to study the effect of parallel granularity on encryption throughput, we evaluated the performance of 16-bit-LED block cypher with different block size. The experimental results are shown in Fig. 11. The independent variables in the graph are the number of threads per block and the size of the data set (expressed as the size of the plaintext). The experimental results show that when the test data are 1MB, 64 MB and 512 MB, threads reach the maximum throughput at 32, 1024 and 1024, respectively. It can be seen that the size of the test data set also affects the value of the optimal threads. However, for different size of data set, the throughput of different threads does not follow a specific rule, but the figure of experimental results shows that for the small data set, the smaller block size has the better throughput. For larger data set, the throughput has a gradual upward trend with the increase of the number of threads per block. So for larger data throughput, the number of threads can choose the largest Value which is 1024.

According to the same method, throughput of other algorithms with different threads can be measured, and the optimal block size can be found. In this step, three lightweight encryption algorithms, LED, press and piccolo, are used to construct tables in three ways. The results of threads optimization test for nine algorithms are shown in Tables 4, 5 and 6.

The optimal block size is very difficult to determine without experimentation. It depends on the program and the GPU architecture. According to our experimental result, the performances of using small block size (32,64,128,256) are generally better when the problem size is small. When the problem size is greater than 64 MB, using large block size (256,512,1024) would be a better choice. We can also calculate the theoretically optimal block size by dividing the maximum number of resident threads per multiprocessor by the maximum number of resident blocks per multiprocessor.

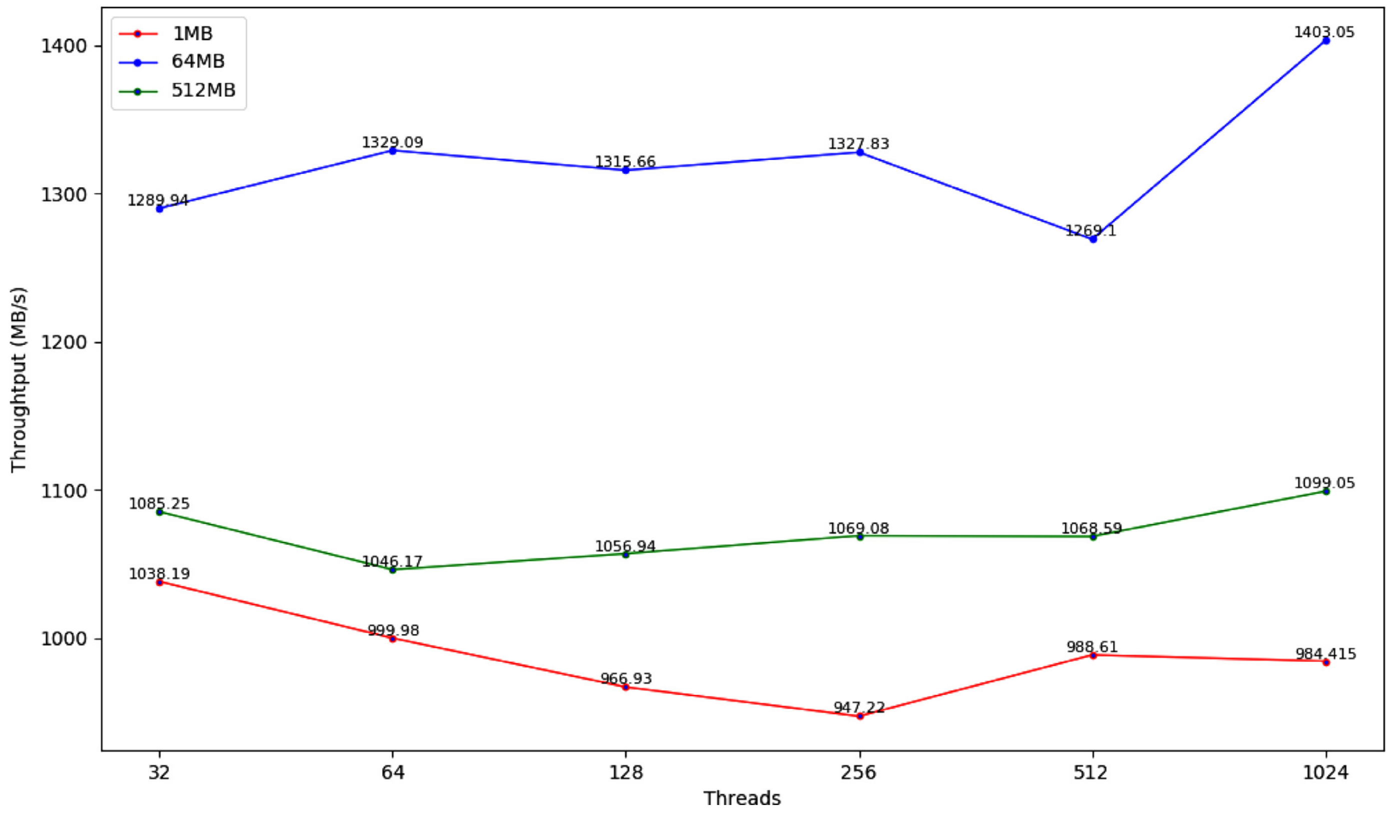


Fig. 11. Impact of the thread number per block in 16-bit LED.

Table 4
Optimal threads and blocks in LED.

LED									
Types		16-bit (4tables)			8-bit (8tables)			4-bit (16tables)	
Data Size	1M	64M	512M	1M	64M	512M	1M	64M	512M
Threads	32	1024	1024	256	256	256	256	512	128
Blocks	4096	8192	65,536	512	32,768	262,144	512	26,384	524288

Table 5
Optimal threads and blocks in PRESENT.

PRESENT									
Types		16-bit (4tables)			8-bit (8tables)			4-bit (16tables)	
Data Size	1M	64M	512M	1M	64M	512M	1M	64M	512M
Threads	256	1024	32	256	256	1024	256	128	512
Blocks	512	8192	2,097,152	512	32,768	65,536	512	655,364	131,072

Table 6
Optimal threads and blocks in Piccolo.

Piccolo								
Types		16-bit (2tables)		8-bit (4tables)			4-bit (8tables)	
Data Size	1M	64M	1M	64M	512M	1M	64M	512M
Threads	1024	1024	32	512	512	64	128	256
Blocks	128	8192	4096	16,384	131,072	2048	655,364	262,144

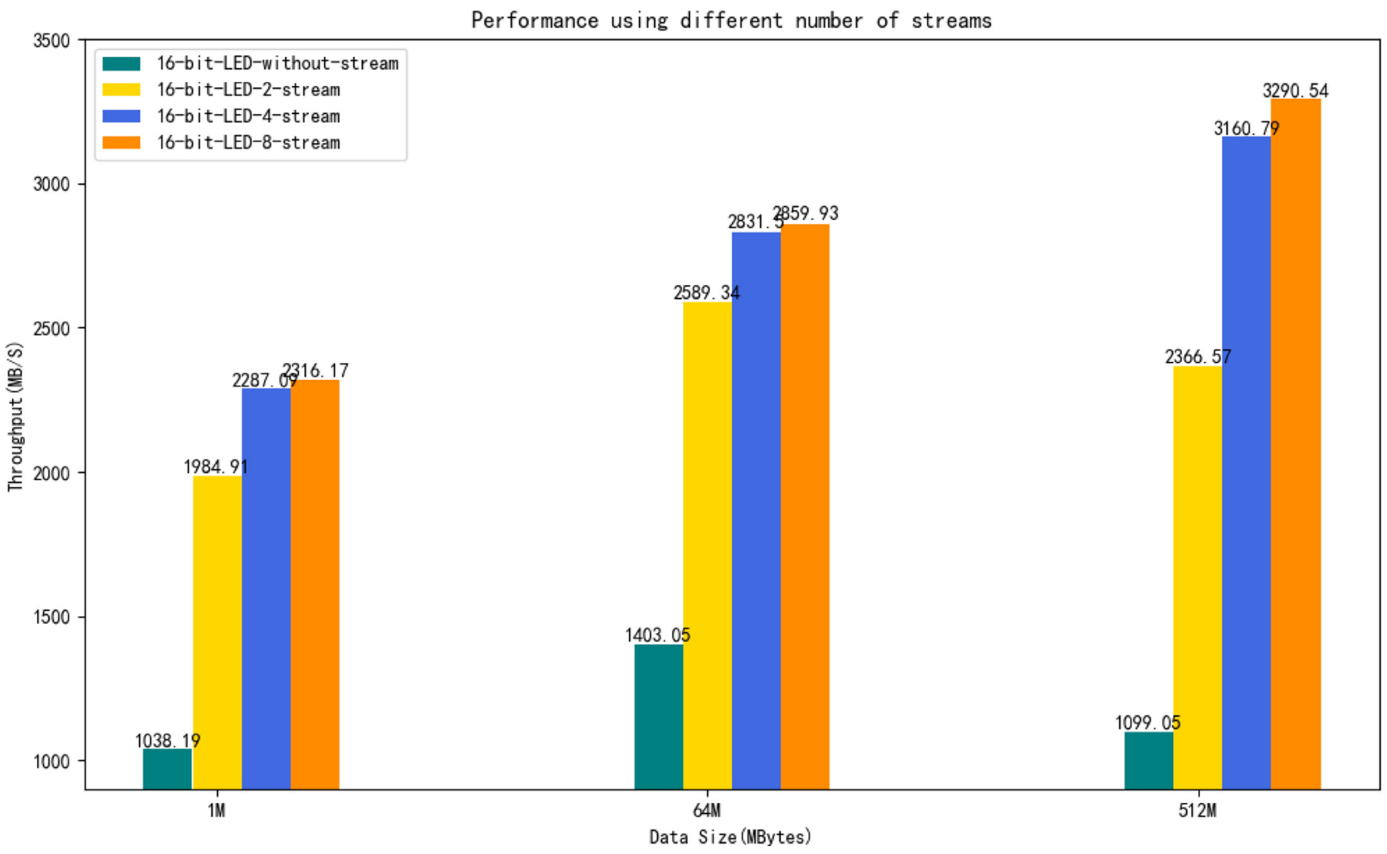


Fig. 12. Improving the throughput of LED algorithm by using multiple streams.

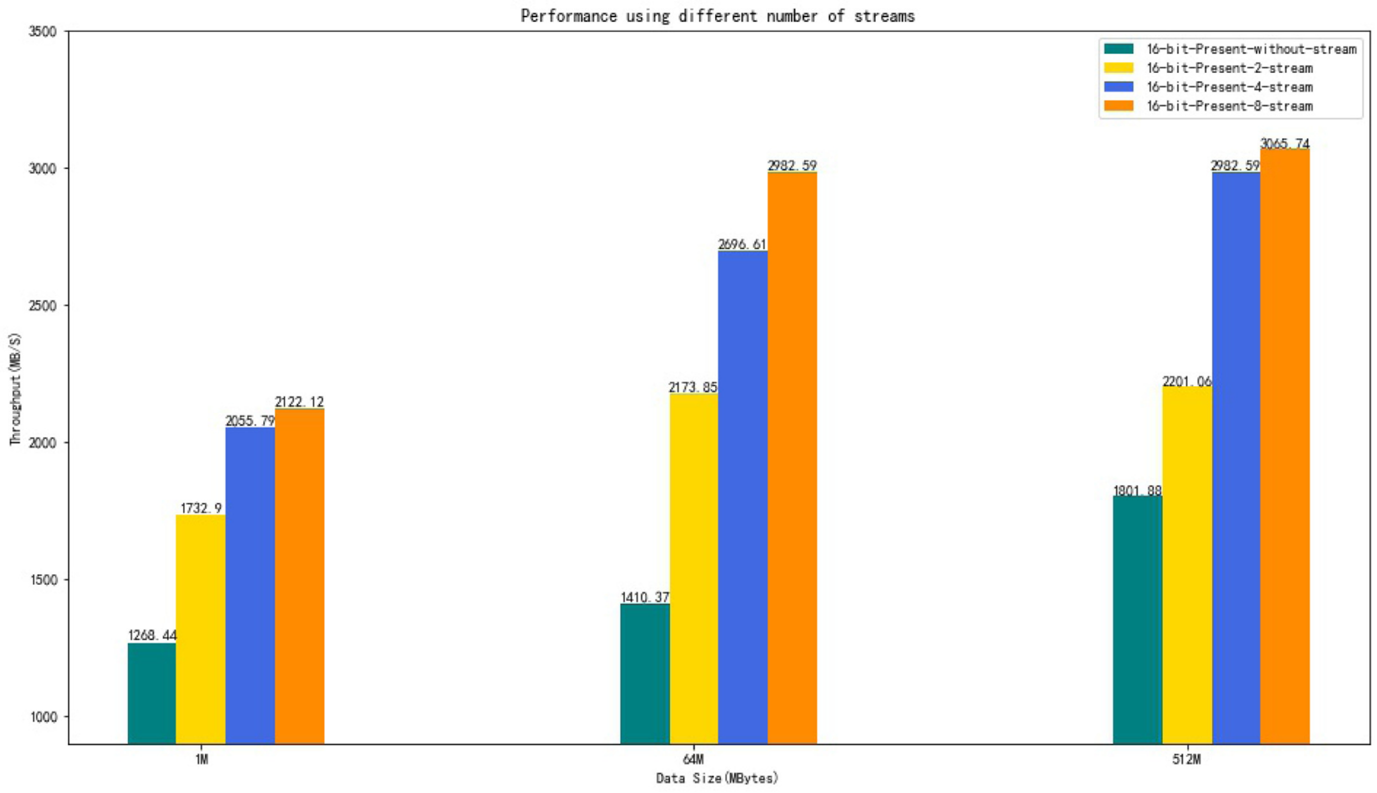


Fig. 13. Improving the throughput of PRESENT algorithm by using multiple streams.

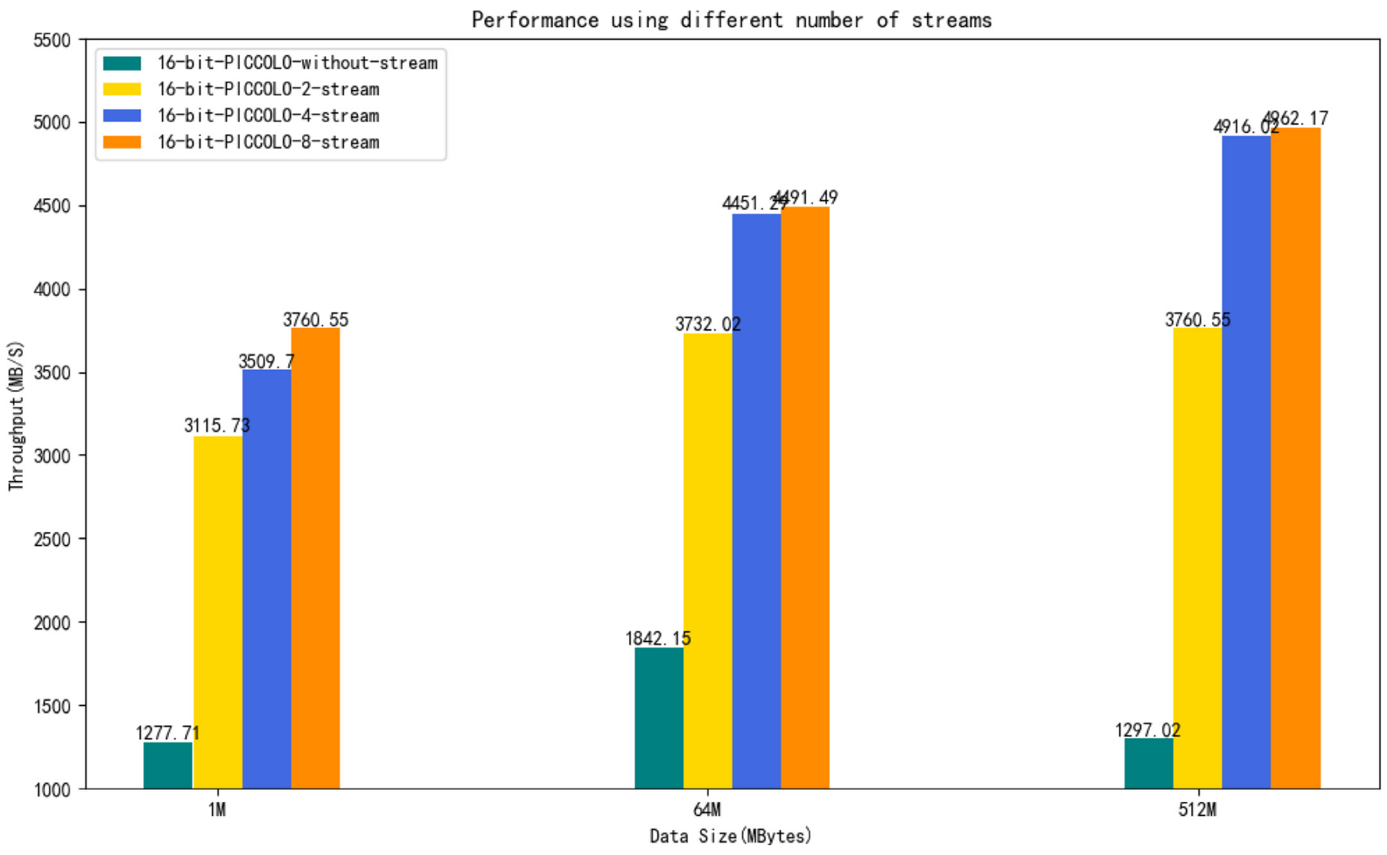


Fig. 14. Improving the throughput of Piccolo algorithm by using multiple streams.

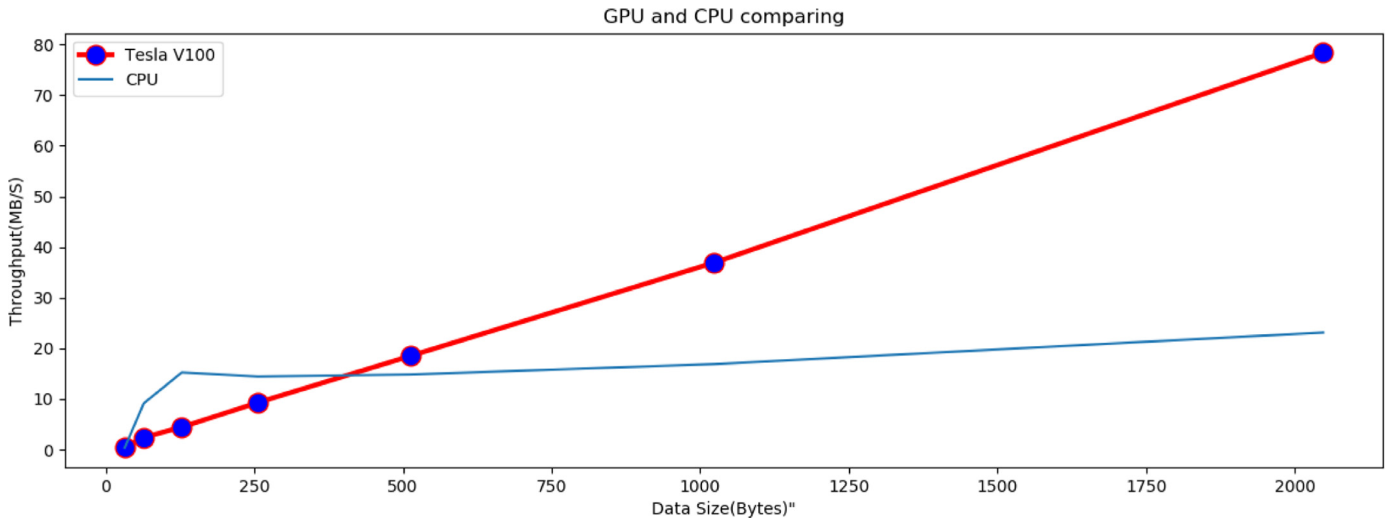


Fig. 15. Comparison of encryption performance on GPU and CPU under 8-bit input LED.

4.3. Overlapping data transfer and processing

The third optimization step is to use the stream method. In theory, when the stream is used in the encryption algorithm, the main purpose is to segment plaintext, encrypt data separately, and hide the transmission time of data in order to speed up the algorithm. In this article, streams are used to optimize lightweight encryption algorithms, and the number of streams is set to 2, 4 and 8 respectively. In the experiment, the throughput is compared, and the target is to verify the acceleration effect of streams in practical applications and find the optimal number of streams. Here, we also take the 16-bit input LED algorithm as an example, using 2,4,8 streams to test the corresponding throughput.

Fig. 12 demonstrates the effect of using multiple streams on encryption performance of LED algorithm. According to the experimental results of 16-bit input LED encryption algorithm, the use of stream speeds up the encryption process to a great extent. For example, as can be shown in the figure, for 1MB data set, 1.91x speed-up can be achieved by using two streams. When using eight streams, 2.23x speed-up can be achieved compared to no-stream implementation.

Similar improvement can be achieved when applying multiple stream on the other two encryption algorithm. According to Figs. 13 and 14, both PRESENT and Piccolo achieve maximum encryption throughput when 8 streams are used. In general, the improvement of throughput is more obvious for the encryption of large data size using multiple streams. This is because the cost of data transition is very expensive. The larger data is, the longer it takes for data transmission. When multiple streams are used, most data transmission time can be hidden, thus the algorithm can achieve a more obvious acceleration.

4.4. GPU and CPU encryption speed comparison

In this experiment, we evaluate the performance of encryption on CPU and GPU. We used 8-bit input LED algorithm to encrypt data. Our GPU operation was carried out on tesla v100. Meanwhile, we fixed the thread of GPU operation as 1024 and the block as 128. CPU is Intel(R) Core(TM) i5-8300h processor.

According to the experimental result in the Fig. 15, CPU achieve higher throughput with encryption of a small amount of data, however the GPU is clearly more advantageous when dealing with large data.

5. Conclusion

This article has studied four optimization techniques of lightweight block cipher implementation on GPU architectures: lookup table based implementation, the use of memory type where the lookup tables are pre-stored, parallel granularity and overlapping between data transfer and processing. The larger the size of lookup tables is, the fewer operations are required in an encryption round, thus the higher throughput can be achieved. In order to reduce the latency of reading element from lookup tables, the tables can be pre-stored in shared memory of GPU. However the size of shared memory is usually less than 1MB, large lookup table can only be stored in global memory. The highest throughput can be achieved when the computing resources on GPU are fully saturated, inappropriate parallel granularity and the idle state cause by the data transmission will greatly reduce the encryption performance. After carefully adjusting the thread block size and using the technique of stream, we managed to greatly improve the performance for all the three lightweight block ciphers on various GPU platform. Also, it shows that for large amount of data, our implementation has significant advantage over the CPU platform.

Conflict of Interest

In this part, we explored the common technique of using different GPU memory. We declare that we do not have any commercial or associative interest that represents a conflict of interest in connection with the work submitted.

Acknowledgments

This work has been partly supported by the National Natural Science Foundation of China under Grant No. 61702212 and the Fundamental Research Funds for the Central Universities under Grand No. CCNU19TS017.

References

[1] Hatzivasilis G, Fysarakis K, Papaefstathiou I, Manifavas H. A review of lightweight block ciphers. J Cryptogr Eng 2017;8:1–44. doi:10.1007/s13389-017-0160-y.
[2] He D, Zeadally S, Xu B, Huang X. An efficient identity-based conditional privacy-preserving authentication scheme for vehicular ad hoc networks. IEEE Trans Inf Forensics Secur 2015;10(12):2681–91.

[3] He D, Wang D. Robust biometrics-based authentication scheme for multiserver environment. IEEE Syst J 2015;9(3):816–23.

[4] Lee VW, Kim C, Chhugani J, Deisher M, Kim D, Nguyen AD, et al. Debunking the 100x GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU. In: Proceedings of the 37th annual international symposium on computer architecture. New York, NY, USA: ACM; 2010. p. 451–60. 10

[5] Asano S, Maruyama T, Yamaguchi Y. Performance comparison of FPGA, GPU and CPU in image processing. In: Proceedings of the international conference on field programmable logic and applications; 2009. p. 126–31. doi:10.1109/FPL.2009.5272532.

[6] Guo J, Peyrin T, Poschmann A, Robshaw M. The LED Block Cipher. In: Preneel B, Takagi T, editors. Cryptographic Hardware and Embedded Systems – CHES 2011. CHES 2011. Lecture Notes in Computer Science, 6917. Berlin, Heidelberg: Springer; 2011.

[7] Bogdanov A, Knudsen LR, Leander G, Paar C, Poschmann A, Robshaw MJB, et al. Present: an ultra-lightweight block cipher. Ches 2007;4727:450–66.

[8] Shibutani K, Isobe T, Hiwatari H, Mitsuda A, Akishita T, Shirai T. Piccolo: an ultra-lightweight blockcipher. In: Proceedings of the international conference on cryptographic hardware & embedded systems; 2011.

[9] Biagio AD, Barenghi A, Agosta G, Pelosi G. Design of a parallel AES for graphics hardware using the CUDA framework. In: Proceedings of the international parallel & distributed processing symposium; 2009.

[10] Guo GL, Qian Q, Zhang R. Different implementations of AES cryptographic algorithm. In: Proceedings of the IEEE international symposium on ieee international conference on high performance computing & communications; 2015.

[11] Lee W-K, Cheong H-S, Phan RC-W, Goi B-M. Fast implementation of block ciphers and PRNGS in maxwell GPU architecture. Cluster Comput 2016;19(1):335–47.

[12] Dai W, Doroz Y, Sunar B. Accelerating NTRU based homomorphic encryption using GPUS. In: Proceedings of the IEEE high performance extreme computing conference; 2015.

[13] Abdelrahman AA, Fouad MM, Dahshan H, Mousa AM. High performance CUDA AES implementation: a quantitative performance analysis approach. In: Proceedings of the computing conference. IEEE; 2017. p. 1077–85.

[14] Benadjila R, Guo J, Lomné V, Peyrin T. Implementing lightweight block ciphers on x86 architectures. In: Proceedings of the international conference on selected areas in cryptography. Springer; 2013. p. 324–51.

[15] Manavski SA. Cuda compatible GPU as an efficient hardware accelerator for AES cryptography. In: Proceedings of the IEEE international conference on signal processing and communications. IEEE; 2007. p. 65–8.

[16] Li Q, Zhong C, Zhao K, Mei X, Chu X. Implementation and analysis of AES encryption on GPU. In: Proceedings of the IEEE 14th international conference on high performance computing and communication and IEEE 9th international conference on embedded software and systems. IEEE; 2012. p. 843–8.