Implementation of SM4 block cipher on CUDA GPU and its analysis*

1st Si-Woo Eum
Division of IT Convergence Engineering
Hansung University
Seoul, South Korea
shuraatum@gmail.com

2nd Hyun-Jun Kim
Division of IT Convergence Engineering
Hansung University
Seoul, South Korea
khj930704@gmail.com

3rd Hyeok-Dong Kwon
Division of IT Convergence Engineering
Hansung University
Seoul, South Korea
korlethean@gmail.com

4th Kyung-Bae Jang
Division of IT Convergence Engineering
Hansung University
Seoul, South Korea
starj1023@gmail.com

5th Hyun-Ji Kim
Division of IT Convergence Engineering
Hansung University
Seoul, South Korea
khj1594012@gmail.com

6th Hwa-Jeong Seo
Division of IT Convergence Engineering
Hansung University
Seoul, South Korea
hwajeong84@gmail.com

***Abstract*—SM4 is a symmetric key algorithm developed by the China National Cryptographic Authority. In this paper, the parallel implementation of SM4 block cipher commonly used in China was performed on GPU. The SM4 block cipher has an implementation that uses an 8-bit Sbox table and an implementation that uses a 32-bit T-table. Measuring the performance of each of the two table implementations, the T-table implementation performed approximately 0.75× worse than the Sbox table implementation. Additionally, Implemented SM4 to use shared memory for better performance. The result is a performance improvement of approximately 1.06× ∼ 1.19× when using shared memory in the Sbox table implementation.**

***Index Terms*—SM4, Block cipher, GPU, Shared memory, Parallel implementation, CUDA programming**

I. Introduction

Graphics Processing Unit(GPU) have become an integral part of today's major computing systems. Due to the programmability and great computational power of GPU, research is underway to map complex problems that require extensive computation to GPU [1]. As interest in virtual currency grows, GPU used for graphics and images are increasingly being applied to crypto. Various studies using GPU for parallel implementation of block ciphers are also being conducted. In a recent study [2], the most commonly used AES block cipher was optimized using techniques that minimize bank conflict in the Shared memory of GPU.

SM4 is a symmetric key algorithm developed by the China National Cryptographic Authority. Its simple design can be applied to various smart devices. And SM4 block cihper used in Chinese National Standard for Wireless LAN WAPI(WLAN Authentication and Privacy Infrastructure) [3].

In this paper, We implement SM4 block cipher on GPU. SM4 uses Sbox table to perform substitution operations. Optimizations are implemented to utilize the Shared memory to minimize memory access speed. The rest of this paper is organized as follows: In Section II, reviews the SM4 block cipher and GPU. In Section III, introduces the implementation method for implementing SM4 block cipher on GPU. In Section IV, evaluates and analyzes the performance of our implementation. Finally, In Section V concludes the paper.

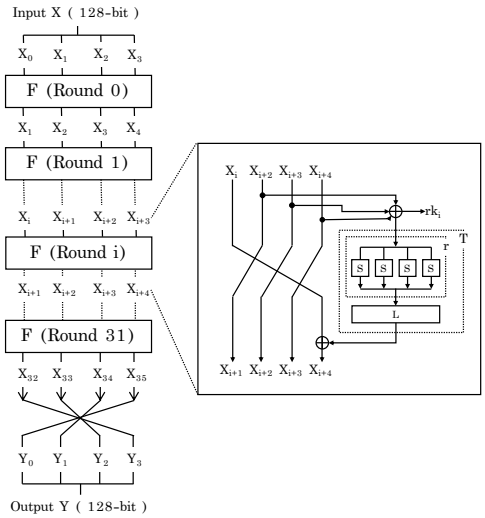

Fig. 1. Structure of SM4 block cipher.

II. Related Work

A. SM4 block cipher

In the SM4 block cipher, it has block size of 128-bit and Key size of 128-bit. and The SM4 block cipher proceeds 32 rounds. There is an `F` function as a round function, and there are `L`, `tau`, `S` and `T` in detail. `L` computation is linear transformation, `tau` computation is nonlinear transformation, `S` computation is substitution using S-box and `T` computation is Permutation [4] [5]. The Encryption process is shown in Figure 1.

- **`F` Round Function** A 128-bit input value is divided into 32-bit units(X_0, X_1, X_2, X_3). The `F` requires X_i, X_{i+1},

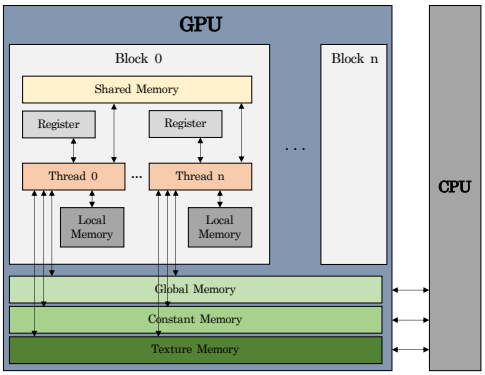

Fig. 2. Memory structure of GPU.

X_{i+2}, X_{i+3} and rk_i(Round key) arguments. The Round function F is defined as follow:
F(X_0, X_1, X_2, X_3, rk)

$$= X_0 \oplus T(X_1 \oplus X_2 \oplus X_3 \oplus rk)$$

- **T Computation** T Computation is a reversible substitution function. It consists of a tau computation and L computation. T is defined as follow:

$$T(A) = L(tau(A)).$$

- **tau Computation** tau is composed of 4 parallel S-boxes. A 32-bit input word A is divided into 8-bit units(a_0, a_1, a_2, a_3). Values divided into 8-bit are substituted through Sbox and then 32-bit output as B(b_0, b_1, b_2, b_3). tau is defined as follow:
tau(A) = (Sbox(a_0), Sbox(a_1), Sbox(a_2), Sbox(a_3))

$$= (\ b_0,\ b_1,\ b_2,\ b_3\ )$$

- **L Computation** L is a linear substitution function. the 32-bit output of tau will be input of L. L is defined as follow:
C = L(B)
$= B \oplus (B \lll 2) \oplus (B \lll 10) \oplus (B \lll 18) \oplus (B \lll 24)$

B. Graphics processing unit(GPU)

GPU are parallel programmable processors that feature arithmetic and memory bandwidths that exceed CPU. GPU provide different types of memory: global memory, shared memory, constant memory, local memory, register and texture memory [6]. Each memory is divided into on-chip and off-chip according to the location of the memory. On-chip includes registers, shared memory, and local memory, and Off-chip includes global memory, constant memory, and texture memory. Since the on-chip is inside the GPU chip, the memory access is very fast, but the memory size is small. Conversely, since the off-chip is outside the GPU chip, the memory access is slow, but the memory size is large. The Memory structure of GPU is shown in Figure 2.

CUDA (Computing Unified Device Architecture) is a parallel computing platform developed by Nvidia. CUDA comes with a software environment that allows developers to use C as a high-level programming language. The CUDA GPU architecture includes Thread, Block, Grid, Warp, and Functional kernel running on the GPU [7].

III. Implementation on GPU

SM4 block cipher implementations use Sbox table for substitution. The sbox table is a precomputed table of 256×8-bit values. The SM4 implementation also has a T-table implementation with a larger size than the Sbox. In implementations using T-table, the values stored in the table are precomputed up to L Computation. The T-table consists of 4 tables with a size of 256×32-bit. In this paper, We implement both types of tables. And It also implements optimizations using shared memory.

A. Implementation of T computation

We implement SM4 block cipher using 8-bit Sbox. First, in the typical implementation, tables are stored in global memory. At this time, the table stored there is an 8-bit Sbox(S-table) or a 32-bit T-table(T-table). The difference between S-table and T-table implementation is the difference in T Computation implementation. The T-table stores precomputed values up to the L computation involved in the T computation. The implementation of T Computation using an S-table is Algorithm 1. The implementation of T Computation using a T-table is Algorithm 2.

Algorithm 1 The implementation of T computation using an S-table.

INPUT : $X_{i+1} \oplus X_{i+2} \oplus X_{i+3} \oplus rk[i]$ //32-bit
OUTPUT : X_{i+4}

$A : X_{i+1} \oplus X_{i+2} \oplus X_{i+3} \oplus rk[i]$

uint32 Temp = 0;
Temp $|=$ ((uint32)S-Table[(uint8)(A$\gg$24)] $\ll$24;
Temp $|=$ ((uint32)S-Table[(uint8)(A$\gg$16)] $\ll$16;
Temp $|=$ ((uint32)S-Table[(uint8)(A$\gg$8)] $\ll$8;
Temp $|=$ S-Table[(uint8)(A)];

/* return L linear Transform */
return (Temp ^ RotL(Temp,2) ^ RotL(Temp,10) ^ RotL(Temp,18) ^ RotL(Temp,24));

Algorithm 2 The implementation of T computation using an T-table.

INPUT : $X_{i+1} \oplus X_{i+2} \oplus X_{i+3} \oplus rk[i]$
OUTPUT : X_{i+4}

$A : X_{i+1} \oplus X_{i+2} \oplus X_{i+3} \oplus rk[i]$

return T-table_0[$(uint8)(A \gg 24)$^
T-table_1[$(uint8)(A \gg 16)$]^
T-table_2[$(uint8)(A \gg 8)$]^
T-table_3[$(uint8)A$];

Although T-table occupies a large memory of 4KB, as in Algorithm 2, it can be seen that several operations are omitted

compared to Algorithm 1. In other words, there are many omitted operations, so the implementation is simple.

B. Storing roundkey in constant memory

SM4 block ciphers use 32×32-bit Roundkey(rk). A Roundkey is a value generated by a key expansion algorithm on the input 128-bit key. A kernel function that does encryption can also be implemented by including a key expansion function, but since all threads use the same key to perform encryption, in this implementation roundkeys are generated before the kernel function performing encryption. If all threads use the same key, it means all threads use the same roundkey at the same time. That is, all threads simultaneously access the memory in which a specific round key is stored. A good memory to use in this case is constant memory. Constant memory is off-chip, so memory access is slow. However, under certain conditions, they can be accessed as fast as register. A specific condition for fast access is when all threads access the same memory. So, we implemented by storing the roundkey in constant memory in all implementations. This implementation can be inefficient if not all threads use the same key.

C. Implementation using shared memory

Shared memory is memory that can be shared and used by all threads included in the same block. Also, It is on-chip, threads can access memory very quickly. However, shared memory has disadvantages such as a small memory size of 48KB and bank conflict. A bank conflict is a bottleneck that occurs when multiple threads access the same bank. This causes latency and is a factor that prevents fast memory access, which is an advantage of shared memory. In this paper, the problem of resolving bank conflicts is left as a future work. In fact, the memory access speed of shared memory is fast enough to ignore bank conflicts.

Shared memory is used by declaring it within a kernel function. In the implementation using shared memory, it is additionally necessary to copy the table stored in the global memory to the shared memory. Since different shared memory is used for each block, the table copy process must be performed in units of blocks. Also, since it is inefficient for all threads to copy the same memory, an optimization implementation in which different threads copy different table values is required.It is implemented so that copy processing can be performed equally for each thread according to the number of threads used in block. The copy process code can be seen in Listing 1.

```
__shared__ uint8_t S[256];

for (i = threadIdx.x; i < 256; i+=blockDim.x)
{
    S[i] = S_t[i];
}
```

Listing 1. The Table Copy Process code

Shared memory is allocated and used through $\_\_shared\_\_$ inside the kernel function. $threadIdx.x$ means the thread index within the block and $blockDim.x$ means the number of threads within the block. For example, if 32 threads are used, each will copy 8 different values (since the table size is 256). If 256 threads are used, each will copy 1 different values.

IV. Evaluation

In this section, we measure and evaluate performance by increasing the number of threads and grids each implementation. It also comparatively analyzes each implementation and draws conclusions about its performance evaluation. Nvidia GeForce GTX 3060 laptop GPU was used for measurements. Visual Studio was used for implementation, and only the kernel function performing encryption was measured in the performance measurement range. Measure the average of the calculation time repeated 10 times, and use the measurement time to calculate the number of encryptions per second and use it as a comparison unit(KB/s). For the input value, an arbitrary value using the $rand()$ function of C language was used, and the length of the input value is Gridsize×Threadsize×Blocksize(byte).

Table I shows performance of SM4 using S-table stored in global memory. Overall, it shows the best performance when using 128 threads, and the higher the number of grids, the better the performance, but the difference is insignificant. The performance of an implementation is used to compare the performance of two other implementations.

Table II shows performance of SM4 using S-table stored in shared memory. As with global memory, it shows the best performance when using 128 threads. It can see performance gains just by using shared memory, despite the addition of the process of copying tables from global to shared memory compared to implementations that use global memory. It was confirmed that the performance improved about $1.06\times \sim 1.19\times$.

Finally, Table III is the performance result of the implementation using T-table. The reason for implementing using T-table is that the operation is omitted due to the simplification of `T` computation, and the size of the register used by the GPU is 32-bit, so it is thought to be more efficient. However, as a result of the measurement, it was confirmed that the performance was lower than when using the S-table. Compared to the implementation using the S-table stored in global memory, performance degradation of about $0.75\times$ was confirmed.

A performance comparison of the three implementations is shown in Figure 3. The biggest performance difference is when using 512 threads and all implementations show worst performance when using 1024 threads.

V. Conclusion

In this paper, the parallel implementation of SM4 block cipher commonly used in China was performed on GPU. As a result of comparative analysis of implementation using S-table and T-table, it was confirmed that the performance of T-table, which simplified `T` computation using a large 4KB table, was not good. In addition, it was confirmed that performance can be improved just by using shared memory

through implementation using shared memory. This means that GPU implementations must actively utilize shared memory for tasks that require heavy memory accesses. It has been confirmed that memory access affects performance, and as a future work, we propose an optimization implementation through bitslicing implementation without memory access.

VI. Acknowledgement

This work was partly supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. NRF-2020R1F1A1048478, 25%) and this work was partly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2021-0-00540, Development of Fast Design and Implementation of Cryptographic Algorithms based on GPU/ASIC, 50%) and this work was partly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2022-0-00627, Development of Lightweight BIoT technology for Highly Constrained Devices, 25%).

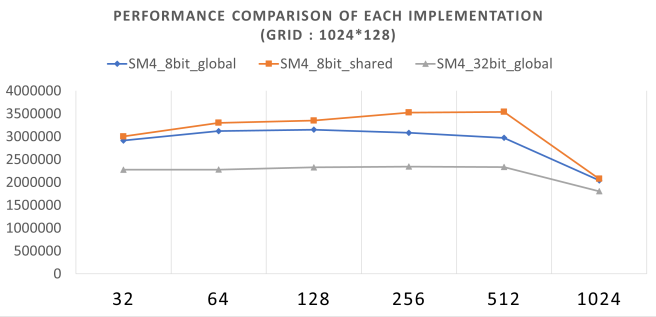

Fig. 3. Comparison of performance measures for each implementation(Grid size : 1024*128).

References

[1] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "Gpu computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, 2008.

[2] W.-K. Lee, H. J. Seo, S. C. Seo, and S. O. Hwang, "Efficient implementation of aes-ctr and aes-ecb on gpus with applications for high-speed frodokem and exhaustive key search," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 6, pp. 2962–2966, 2022.

[3] J. Zhang, M. Ma, and P. Wang, "Fast implementation for sm4 cipher algorithm based on bit-slice technology," in *International Conference on Smart Computing and Communication*, pp. 104–113, Springer, 2018.

[4] W. Diffie and G. Ledin, "Sms4 encryption algorithm for wireless networks," *Cryptology ePrint Archive*, 2008.

[5] H. Kwon, H. Kim, S. Eum, M. Sim, H. Kim, W.-K. Lee, Z. Hu, and H. Seo, "Optimized implementation of sm4 on avr microcontrollers, risc-v processors, and arm processors," *Cryptology ePrint Archive*, 2021.

[6] S. An, Y. Kim, H. Kwon, H. Seo, and S. C. Seo, "Parallel implementations of arx-based block ciphers on graphic processing units," *Mathematics*, vol. 8, no. 11, p. 1894, 2020.

[7] D. Guide, "Cuda c programming guide," *NVIDIA, July*, vol. 29, p. 31, 2013.

TABLE I
PERFORMANCE OF SM4 BLOCK CIHPER USING 8-BIT SBOX STORED IN GLOBAL MEMORY(UNIT: KB/S(KILO BLOCK ENCRYPTION PER SECOND))

SM4_Global S-table		**Threads**					
		32	64	128	256	512	1024
Girds	1024 * 8	2746781.25	3011764.75	**3097398.5**	3031379.5	2626483	1606936.25
	1024 * 16	2833425.5	3068624.5	**3119573.5**	3057629.25	2659999.5	1605959.5
	1024 * 32	2878830.5	3097867.25	**3127195.25**	3070580	2674022.25	1972573.125
	1024 * 64	2900439	**3111752.75**	3077269.75	3077269.75	2812197	2063320.25
	1024 * 128	2912607.5	3119811.25	**3151375.5**	3082538.5	2972410.25	2040644.125

TABLE II
PERFORMANCE OF SM4 BLOCK CIHPER USING 8-BIT SBOX STORED IN SHARED MEMORY(UNIT: KB/S(KILO BLOCK ENCRYPTION PER SECOND))

SM4_Shared S-table		**Threads**					
		32	64	128	256	512	1024
Girds	1024 * 8	2797814.25	3196004.75	**3295784**	3190031.25	2915095	1585476.75
	1024 * 16	2910744.75	3253892.5	**3324675.5**	3255961.75	3064606.75	1605298.75
	1024 * 32	2964678.75	3279423.5	**3339856.5**	3307627	3160860	1731146.125
	1024 * 64	2990217.5	3292604.75	**3346678.75**	3334826	3204757.25	2056502.25
	1024 * 128	3003153	3299633.5	3350374.25	3525905	**3539520.75**	2076374.875

TABLE III
PERFORMANCE OF SM4 BLOCK CIHPER USING 32-BIT T-TABLE STORED IN GLOBAL MEMORY(UNIT: KB/S(KILO BLOCK ENCRYPTION PER SECOND))

SM4_Global T-table		**Threads**					
		32	64	128	256	512	1024
Girds	1024 * 8	2267493.5	2323049	**2348623.75**	2288268.25	1910804.125	1396331.875
	1024 * 16	2321995.5	2347008.75	**2362168.25**	2312817.75	1942152.75	1379949.25
	1024 * 32	2117890.5	**2358903.25**	2060154.875	2326677.75	1934036.875	1721767.875
	1024 * 64	2189203.75	2367356.25	**2372979.5**	2286001.25	2145358.75	1804748.25
	1024 * 128	2275176.5	2278910.75	2327537.5	**2342964.75**	2333570.5	1805061.125