Speed Record of AES-CTR and AES-ECB Bit-sliced Implementation on GPUs

Wai-Kong Lee, *Member, IEEE,* Seog Chung Seo, *Member, IEEE,* Hwajeong Seo, *Member, IEEE,* Dong Cheon Kim, Seong Oun Hwang, *Senior Member, IEEE.*

***Abstract*—The Advanced Encryption Standard (AES) has been widely used to protect digital data in various applications, such as secure IoT communication, files encryption, and pseudo-random number generation. The efficient implementation of AES on parallel architecture such as GPU, has attracted considerable interest over the past decade. These prior studies mainly implemented the AES electronics code book (ECB) and counter (CTR) mode using the table-based approach. In this brief, we set a new speed record of AES-ECB and AES-CTR on Graphics Processing Unit (GPU) based on the proposed bit-sliced implementation techniques. Our implementation achieved 2.6% (ECB) and 9% (CTR) faster than the state-of-the-art table-based implementation on a RTX3080 GPU. Our work evaluated on an embedded GPU (Jetson Orin Nano) can also achieve high throughput at 60 Gbps, which is 1.9% (ECB) and 7% (CTR) faster than state-of-the-art.**

***Index Terms*—Advanced Encryption Standard, Graphics Processing Unit, Bit-sliced Implementation.**

I. Introduction

MANY attempts have been made to produce high AES [1] encryption throughput in counter (CTR) and electronic code book (ECB) modes using a graphics processing unit (GPU) [2]–[5]. Tezcan et al. [3] proposed to store the AES states on shared memory and partially mitigate the associated bank conflict issues. Lee et al. [5] improved the performance of AES by removing all the bank conflicts on shared memory, setting up the highest throughput record to date. On the other hand, Nishikawa et al. explored the bit-sliced approach to implement AES on a GPU, which is slightly faster than the T-box approach. Hajihassani et al. [4] continue this line of research and reported a bit-sliced AES implementation superior to previous works, but was slightly slower than Lee et al. [5]. The main drawback of [4] was that each thread encrypts 32 AES blocks, thus consuming a lot of registers.

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (Ministry of Science and ICT) (RS-2024-00340882). This work was also supported by funding from The Circle Foundation (Republic of Korea) for 1 year since December 2023 as Quantum Security Research Center selected as the 2023 The Circle Foundation Innovative Science Technology Center under Grant 2023 TCF Innovative Science Project-05. This work was also 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.)

Wai-Kong Lee is with Universiti Tunku Abdul Rahman, Kampar 31900, Malaysia.

Seong Oun Hwang (Corresponding Author) is with Gachon University, Seongnam 13120, South Korea (e-mail: sohwang@gachon.ac.kr).

Hwa Jeong Seo is with Hansung University, Seoul 02876, South Korea.

Seog Chung Seo and Dong Cheon Kim are with Kookmin University, Seoul 02707, South Korea.

Side channel analysis is an important issue that was also being actively researched [6]. Recently, researchers also found that AES is still post-quantum resistance [7], which motivate us to continue improving its performance on a GPU platform. In this brief, we proposes a high throughput bit-sliced AES implementation on GPUs that surpasses the state-of-the-art results [5]. The contributions are summarized below:

1) This study proposed to utilize the barrel-shiftrows representation [8] for bit-sliced AES implementation to achieve efficient shiftrow operation. We also proposed to compute eight AES blocks per thread to avoid using excessive registers for storing the AES state [4].
2) Bit-slice implementation requires pack/unpack operations to convert between ordinary and bit-sliced representation, which is expensive for the GPU. To optimize this, a technique to generate the counter values on-the-fly was proposed, which completely avoid the pack operation. This study also replaced some expensive steps in unpacking with the built-in `PRMT` instruction.
3) The experimental results show that the proposed implementation could achieve up to 1620 Gbps and 60 Gbps on RTX3080 (desktop GPU) and Jetson Orin Nano (embedded GPU) respectively. Our AES-CTR implementation was 9% and 7% faster than [5] on a RTX3080 and Jetson Orin Nano GPU, respectively. The source code for AES-128-CTR implementation is shared on the public domain: https://github.com/benlwk/AES-BS-GPU/

II. Background

A. Bit-sliced AES Implementation

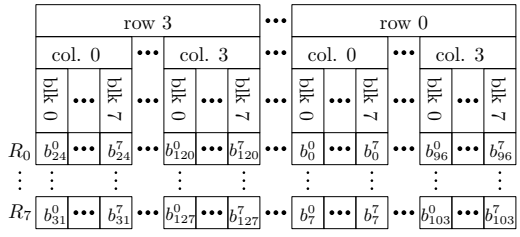

Fig. 1. Bit-sliced representation used in [9]. Eight AES blocks (128-bit each) are stored in eight 128-bit registers $R_0 - R_7$, wherein b^i_j represents the j-th bit of the i-th block.

Since AES is based on byte-wise operations, there are only 256 possible results for each operation, so we can pre-compute them into four look-up tables [1] (T-box). In this way, the sub-bytes, shift-row and mix-column are reduced down to table

look-up, significantly improved the performance of AES on 32-bit and 64-bit architectures. However, many shift/rotate and mask operations are still required to extract the byte-level data, which are slow in many processor architectures. Bit-sliced implementation technique [9] makes use of the composite field [10], allowing the AES linear layer and table look-up (S-box or T-box) to be replaced with logical instructions of bit-level. Figure 1 shows the 128-bit AES state transposed and represented in the bit-sliced form, so that the the 128-bit registers can handle entire state. Hence, all operations in AES encryption can be performed at the bit-level and encrypted in parallel. This avoids many shift/rotate/mask operations found in the byte-level approach, but the pack/unpack process to achieve this representation is non-trivial.

B. State-of-the-art AES Implementation on GPUs

Hajihassani et al. [4] proposed a new bit-sliced representation that can compute 32 AES blocks per thread, achieving ≈33% higher throughput than another bit-sliced implementation on P100 GPU [2]. This bit-sliced representation [4] directly transposed the plaintext from row-major to column-major, requiring 128×32-bit registers to hold the AES states. This approach achieved high throughput, but suffered from high register consumption. Note that the GPU can use up to maximum 255 registers per thread and 128 of them are already used to store the AES state, leaving not many for other operations, thus limiting the number of parallel warps execution. Recently, two notable methods that rely on the T-box approach [3], [5] were reported. Tezcan et al. [3] proposed to duplicate 32 copies of the T-box to avoid bank conflicts in shared memory read in T-box. Lee et al. [5] proposed a warp-based storage technique to mitigate the bank conflict issue when T-box is written onto the shared memory. This is the best performance for AES-CTR to date, achieving 1598 Gbps and 1489 Gbps on V100 and RTX3080, respectively.

III. Proposed Bit-sliced AES Implementation on GPU

A. Adapting Barrel-shiftrows Representation to GPU

Barrel-shiftrows is a bit-slice representation first introduced by [8]. Referring to Figure 2, Barrel-shiftrows were similar to [9], except that 32 32-bit registers were use instead of eight 128-bit registers. This representation is suitable for shift-row operation because the rows are separated in distinct registers, so that word-wise rotations can replace the byte-wise rotations. This also avoided rotations on the mix-column, which is particularly suitable for architectures like GPU, which does not support native rotation. By using barrel-shiftrows representation, we can process eight AES blocks per thread, wherein only 32 registers are used to store the AES state. This avoids the issue of high registers usage as found in [4].

B. On-the-fly Bit-sliced Counter Generation

The counter values for AES-CTR could be generated on the GPU using the unique thread ID with an incremental step of one. This technique was also previously used in other AES implementations on GPUs [3], [5]. However, bit-sliced implementation requires a costly pack operation to convert the counter values into the bit-sliced form. To avoid this pack operation, a technique was proposed to generate the counter values directly in the barrel-shiftrows bit-sliced form.

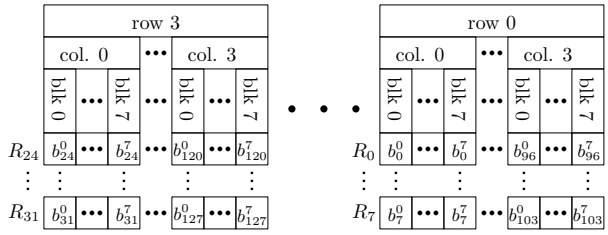

Fig. 2. Adapted from [8]. Eight AES blocks are distributed into 32 32-bit registers $R_0 - R_{31}$, wherein b^i_j represents the *j*-th bit of the *i*-th block.

Algorithm 1 Generate counter values on-the-fly.

1: $tid = blockDim \times blockIdx + threadIdx$
2: State[31] = 0x55000000 XOR nonce[31]
3: State[30] = 0x33000000 XOR nonce[30]
4: State[29] = 0x0F000000 XOR nonce[29]
5: State[28] = (tid%2)×0xFF000000 XOR nonce[28]
6: State[27] = ((tid≫1)%2)×0xFF000000 XOR nonce[27]
7: State[26] = ((tid≫2)%2)×0xFF000000 XOR nonce[26]
8: State[25] = ((tid≫3)%2)×0xFF000000 XOR nonce[25]
9: State[24] = ((tid≫4)%2)×0xFF000000 XOR nonce[24]
10: State[23] = ((tid≫5)%2)×0xFF000000 XOR nonce[23]
11: Skip the remaining lines for brevity.

Referring to Figure 3, the 64-bit nonce is a pseudo-random number, generated on a CPU and used by all GPU threads. Each thread encrypts eight counter blocks and each AES encryption has four 32-bit states, making total $4 \times 8 = 32$ AES states on each thread. After converting to a bit-sliced form, the nonce is stored on the lower 16-bit of all AES states, which exhibits a binary pattern. For instance, a value '8' is converted to 'FF', '00', '00', and '00' in bit-sliced form, which representing the binary value '1000' for '8'. On the other hand, the counter values are stored on the upper 16-bit. Since each thread encrypts a different set of counter values, the values after bit-sliced transformation is also different. For instance, state[27] and state[28] of thread 0 shows '00' and '00', which is different from thread 1 that shows '00' and 'FF' and thread 2 that shows 'FF' and '00', respectively, following a binary pattern similar to the nonce. In contrast, state[29], state[30] and state[31] remain unchanged throughout entire AES encryption. Algorithm 1 was proposed based on this specific pattern, which can be used to generate the counter values in bit-sliced form on-the-fly. It first compute the unique counter value from the thread ID in line 1. State[29], state[30] and state[31] have fixed values; they are combined with nonce values in lines 2 – 10. The nonce values are generated on the CPU and copied to the GPU global memory. Other AES states (lines 5 onwards) were computed following the binary patterns observed in Figure 3 and combined with their respective nonce values, thus completely avoided the expensive pack operation.

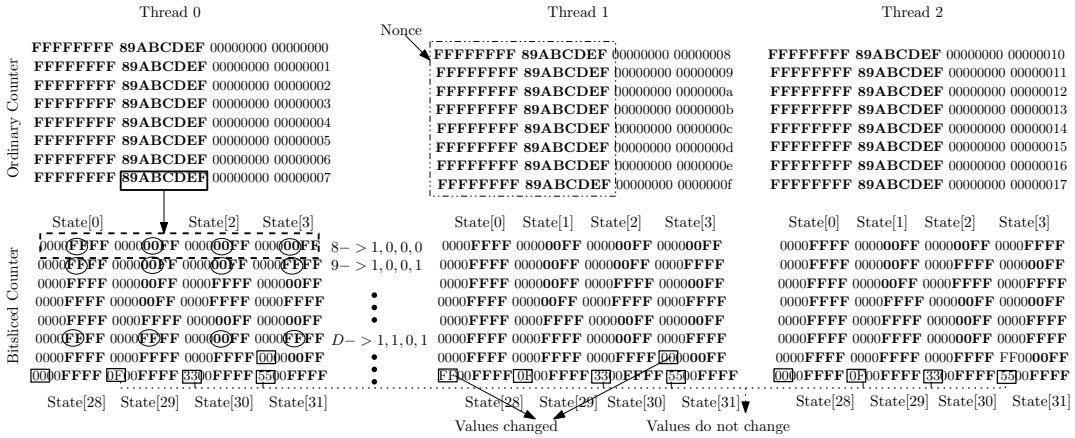

Fig. 3. Converting counter values to bit-sliced form. For example, '8' is converted to binary '1, 0, 0, 0', and 32 counter values makes up "FF, 00, 00, 00". Nonce (bold font) is stored at the lower 16-bit of AES states, while the upper 16-bit represents the counter values.

C. Optimizing the SWAPMOVE Operations for Unpack

After encryption, the bit-sliced ciphertext needs to be unpacked into ordinary form. The unpack process relies heavily on SWAPMOVE operation, which is widely used to exchange/shift bits between two variables as defined below:

$$\begin{aligned} tmp &= (b\ XOR\ (a \gg n))\ AND\ mask \\ b &= B\ XOR\ tmp \\ a &= b\ XOR\ (tmp \ll n) \end{aligned} \tag{1}$$

where *a* and *b* are the variables to be swapped, *n* is the shift amount and *mask* is the logical mask to be used. When $n = 8, 16$, the movement is byte-wise, which are also found in the unpack operation. This process is time-consuming because each SWAPMOVE takes six logical instructions to complete, and unpack operation requires a lot of SWAPMOVE. A careful investigation reveals that the `PRMT` instruction can perform byte-wise data shuffling. Therefore, we proposed to replace the expensive SWAPMOVE with a faster routine for byte-wise data movement found in unpack operation as follows:

$$\begin{aligned} &PRMT.b32\ tmp,\ a,\ b,\ sel\_1 \\ &PRMT.b32\ b,\ a,\ tmp, sel\_2 \\ &MOV.b32\ a,\ tmp \end{aligned} \tag{2}$$

where *a* and *b* are the two variables to be swapped, and *tmp* is a temporary register. The values sel_1 and sel_2 are the selection values used to determine the swap patterns. The byte-wise swap can be performed with only three instructions, which is much faster than SWAPMOVE (six instructions).

After the unpacking process, the ciphertext adopts byte-wise big-endian form. For a successful decryption, the ciphertext needs to be arranged back to little-endian form, which can be achieved through the shift and XOR operations show below:

$$\begin{aligned} tmp = &((y >> 24)\ AND\ 0xFF)| \\ &((y << 8)\ AND\ 0xFF0000)| \\ &((y >> 8)\ AND\ 0xFF00)| \\ &((y << 24)\ AND\ 0xFF000000) \end{aligned} \tag{3}$$

where y is a 32-bit input and *tmp* is the 32-bit output in little-endian form. Similar to unpacking, we found that these operations can be replaced by using one PRMT instruction:

$$PRMT.b32\ tmp,\ y,\ y, 0x00004567 \tag{4}$$

where '0x00004567' is the selection value required to arrange the input y into its little-endian form.

D. Other Techniques

Loop unrolling is applied to reduce the conditional check statements on ***for*** loops and each thread performs encryption on several CTR blocks [5]. The ciphertext is also stored in the global memory in a coalesced manner to allow burst mode write and achieve a 100% efficiency on global memory access.

IV. Experimental Results and Discussions

The experiments were carried out on a desktop GPU (RTX 3080) and an embedded GPU (Jetson Orin) released by NVIDIA, implemented on the CUDA 12.1 SDK.

A. Ablation Study

In this experiment, 2B of data is encrypted with parallel blocks, each consisting of 128 threads. Within each thread, AES encryption was performed on eight different counter/plaintext. The encryption key was expanded on the CPU copied to the shared memory on the GPU; it is used throughout the AES-CTR and AES-ECB encryption. The ECB mode was similar to CTR mode, except that each thread encrypts the plaintext stored on the global memory, instead of the CTR values. Referring to Table I, the throughput achieved by adopting barrel-shiftrows implementation is 1440 Gbps on an RTX 3080. Using the proposed CTR generation technique, the throughput was improved 4.5% to 1505 Gbps. Replacing the byte-wise SWAPMOVE with `PRMT` instruction provides a further 4.9% improvement, recording a throughput of 1579 Gbps. Finally, the best throughput of 1623 Gbps (CTR) and 1406 Gbps (ECB) were achieved after incorporating the other supplementary techniques. Note that the bit-sliced counter

TABLE I
PERFORMANCE FROM AES-128 CTR AND ECB ENCRYPTION ON RTX 3080 GPU.

GPU Device	RTX 3080	
Proposed Implementation	Throughput (Gbps) CTR	ECB
Barrel-shiftrows (Section III-A)	1440	1249
Bit-sliced CTR (Section III-B)	1505	N/A
Optimize SWAPMOVE (Section III-C)	1579	1351
Other techniques (Section III-D)	**1623**	**1406**

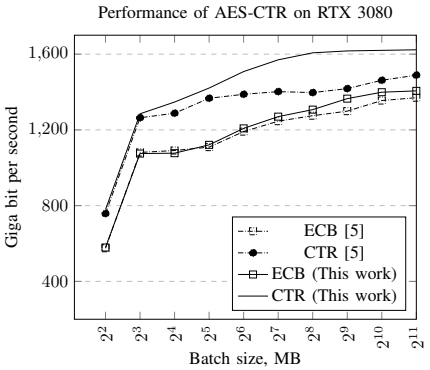

Fig. 4. Performance of AES-CTR and AES-ECB on RTX 3080 with different batch sizes.

generation does not apply to ECB mode, because the plaintext stored on the global memory was encrypted instead of being generated on the GPU. Hence, AES-ECB is slightly slower than AES-CTR due to additional global memory access.

B. Performance of AES-CTR on RTX 3080 and Jetson Orin

Figure 4 and 5 show the results of AES-CTR and AES-ECB implementation with various batch sizes. The throughput increases proportionally to the batch size because more workload is assigned to the GPU when the batch size increases, and the throughput saturates at some point when the GPU is fully utilized. The highest throughput achieved on a RTX 3080 GPU are 1623 Gbps (CTR) and 1406 Gbps (ECB). On a Jetson Orin Nano, the performance peak at 60 Gbps (CTR) and 54 Gbps (ECB).

C. Comparison with state-of-the-art

Lee et al. [5] reported the fastest AES implementation to date, achieving 1489 Gbps AES-CTR encryption on RTX 3080. On the same GPU, our results is 9% (CTR) and 2.6% (ECB) faster than the previous record [5]. We also evaluate the implementation of [5], which was open-sourced, on a Jetson Orin Nano, and achieve 7.1% (CTR) and 1.9% (ECB) speed-up. Note that a table-based implementation of AES could be vulnerable to side-channel attacks [11], which is not recommended for highly sensitive applications. In contrast, bit-sliced implementation can achieve a higher throughput on GPUs and also more resistant to side-channel attacks.

V. CONCLUSIONS

This paper presents several techniques to optimized the barrel-shiftrow bit-sliced representation and achieves a new

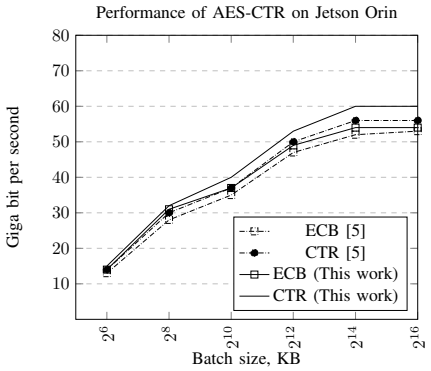

Fig. 5. Performance of AES-CTR and AES-ECB on Jetson Orin Nano with different batch sizes.

speed record for AES-CTR and AES-ECB implementation on GPU platforms. This proposed solution is very useful in applications that need to encrypt/decrypt a large amount of data in high speed. In addition, it can be used to accelerate the encryption of counter values in AES-GCM mode to achieve a high performance authenticated encryption. In future, we intend to utilize the findings in this work to develop high throughput implementation of ASCON [12], the newly standardized lightweight cryptography algorithm, on a GPU.

REFERENCES

[1] "Advanced Encryption Standard (AES)," *Federal Information Processing, Standards Publication FIPS-197*, vol. 197, no. 441, p. p.0311, 2001.
[2] N. Nishikawa, H. Amano, and K. Iwai, "Implementation of bitsliced AES encryption on CUDA-Enabled GPU," in *Network and System Security*, 2017, pp. 273–287.
[3] C. Tezcan, "Optimization of advanced encryption standard on graphics processing units," *IEEE Access*, vol. 9, pp. 67 315–67 326, 2021.
[4] O. Hajihassani, S. K. Monfared, S. H. Khasteh, and S. Gorgin, "Fast AES implementation: A high-throughput bitsliced approach," *IEEE Transactions on parallel and distributed systems*, vol. 30, no. 10, pp. 2211–2222, 2019.
[5] 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.
[6] M. Mozaffari-Kermani and A. Reyhani-Masoleh, "A low-power high-performance concurrent fault detection approach for the composite field s-box and inverse s-box," *IEEE Transactions on computers*, vol. 60, no. 9, pp. 1327–1340, 2011.
[7] K. Jang, A. Baksi, H. Kim, G. Song, H. Seo, and A. Chattopadhyay, "Quantum analysis of aes," *Cryptology ePrint Archive*, 2022.
[8] A. Adomnicai and T. Peyrin, "Fixslicing AES-like ciphers: New bitsliced AES speed records on ARM-Cortex M and RISC-V," *Cryptology ePrint Archive*, 2020.
[9] E. Käsper and P. Schwabe, "Faster and timing-attack resistant AES-GCM," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2009, pp. 1–17.
[10] D. Canright and D. A. Osvik, "A more compact aes," in *Selected Areas in Cryptography: 16th Annual International Workshop, SAC 2009, Calgary, Alberta, Canada, August 13-14, 2009, Revised Selected Papers 16*. Springer, 2009, pp. 157–169.
[11] Y. Gao and Y. Zhou, "Side-channel attacks with multi-thread mixed leakage," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 770–785, 2020.
[12] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, "Ascon v1.2: Lightweight authenticated encryption and hashing," *J. Cryptol.*, vol. 34, no. 3, p. 33, 2021. [Online]. Available: https://doi.org/10.1007/s00145-021-09398-9