BitSlicing Implementation on GPU

Zhou Jiancong
2022.2.21

Outline

- 《Fast AES Implementation: A High-Throughput Bitsliced Approach》
- Other Bitslicing Implement

1-《Fast AES Implementation: A High-Throughput Bitsliced Approach》

1.1 Concentration & Method

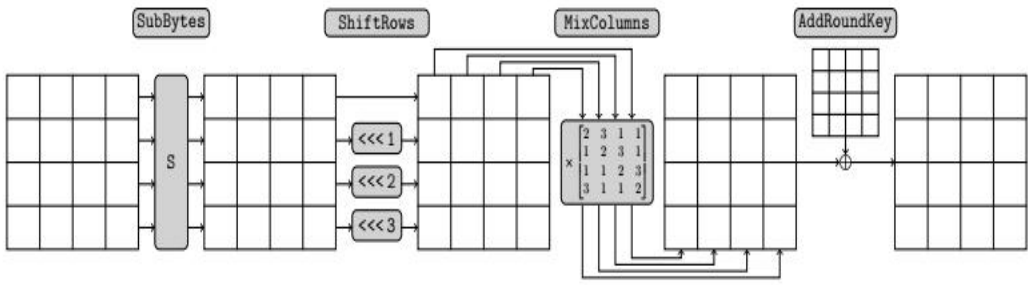
- Contribution：
- *Highest published and reported encryption throughput of ECB mode and CTR mode*

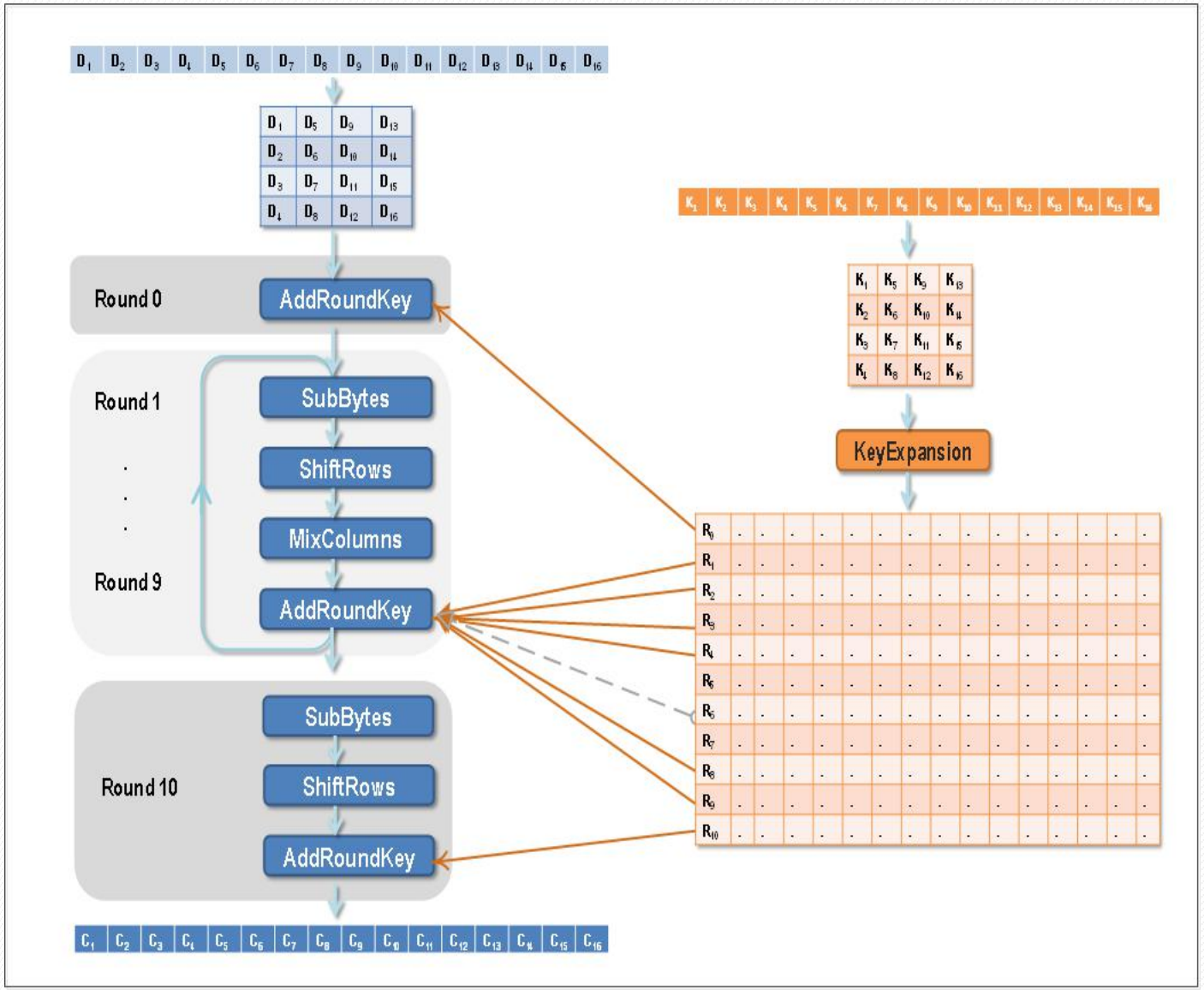
- Main Idea
- *Completely BitSliced AES Implementation*
- *Specific Optimization in AddRoundKey, SubBytes, ShiftRows, MixColums Stages*

1-《Fast AES Implementation: A High-Throughput Bitsliced Approach》

1.2 Introduction of AES

- SPN Structure
- AES-128 has **10 Rounds**
- **4 Stages** each Round
  - -AddRoundKey
  - -SubBytes
  - -ShiftRows
  - -MixColums



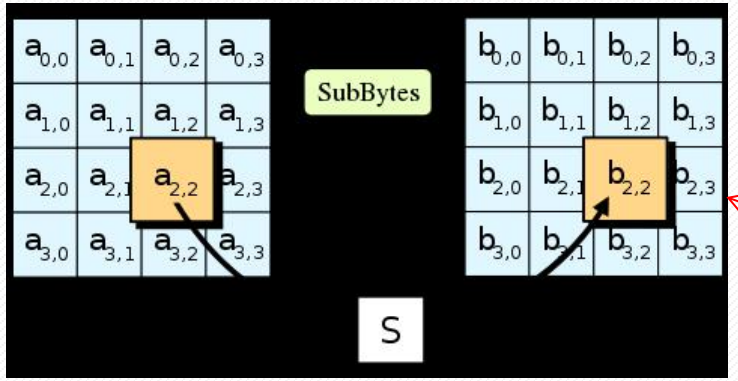


1-《Fast AES Implementation: A High-Throughput Bitsliced Approach》

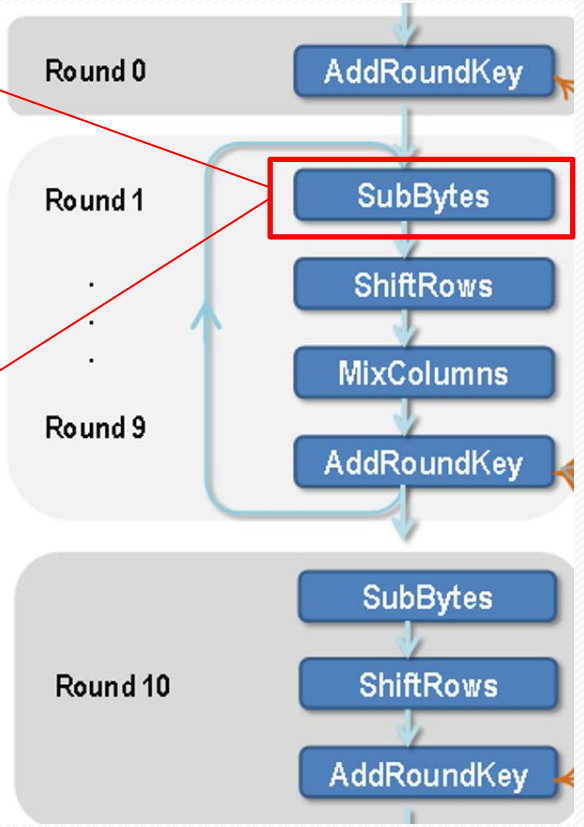
1.2 Introduction of AES

SubBytes

- **Sub(x,y)**
- e.g. Sub(5,3)=ed



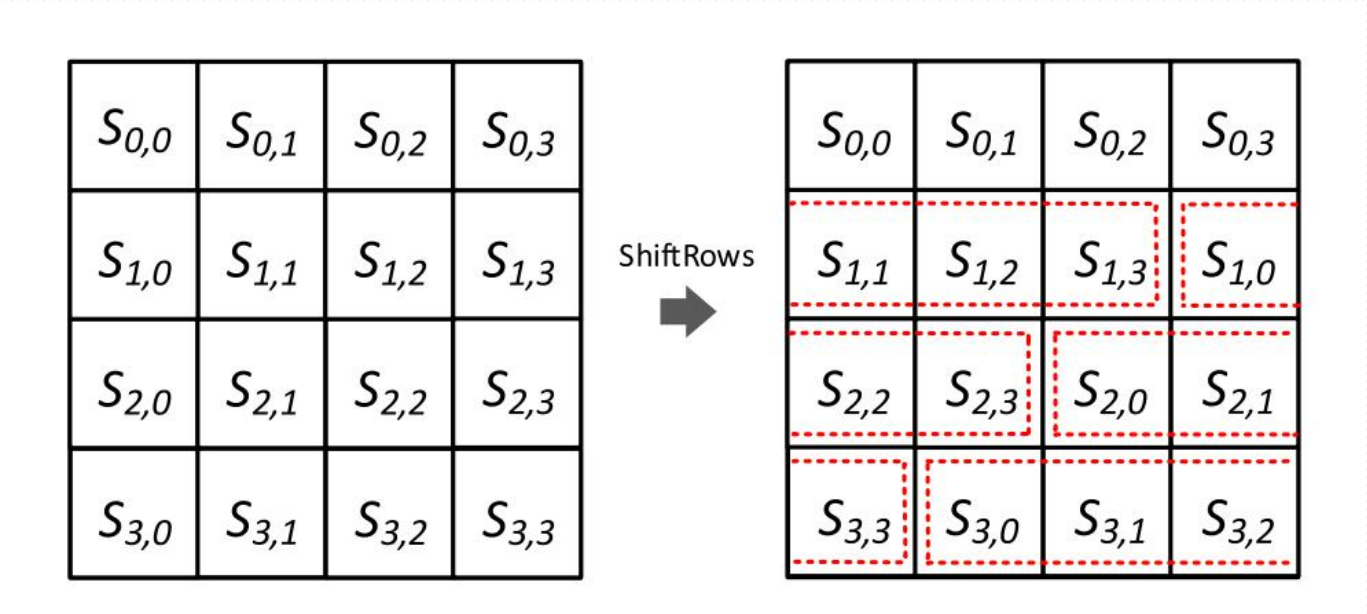
x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

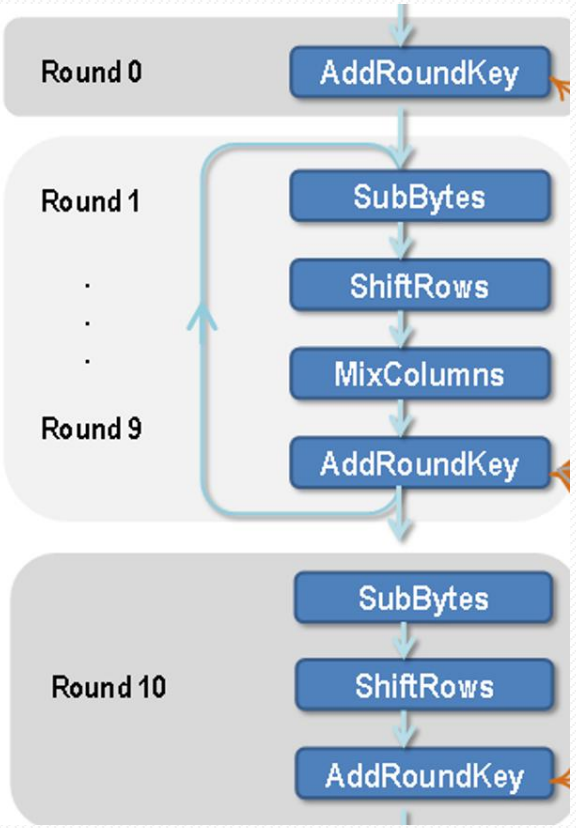


1-《Fast AES Implementation: A High-Throughput Bitsliced Approach》

1.2 Introduction of AES

ShiftRows





1-《Fast AES Implementation: A High-Throughput Bitsliced Approach》

1.2 Introduction of AES

MixColumns

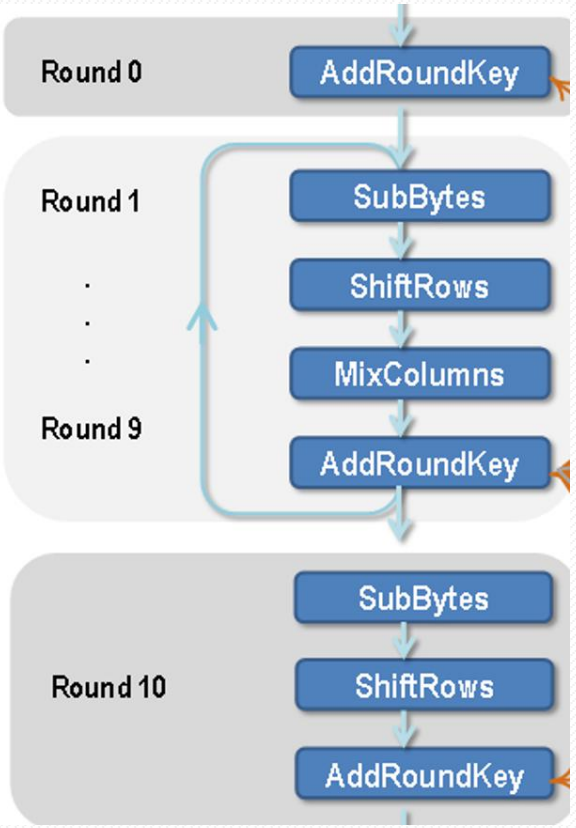
$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$

$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$

$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}).$$

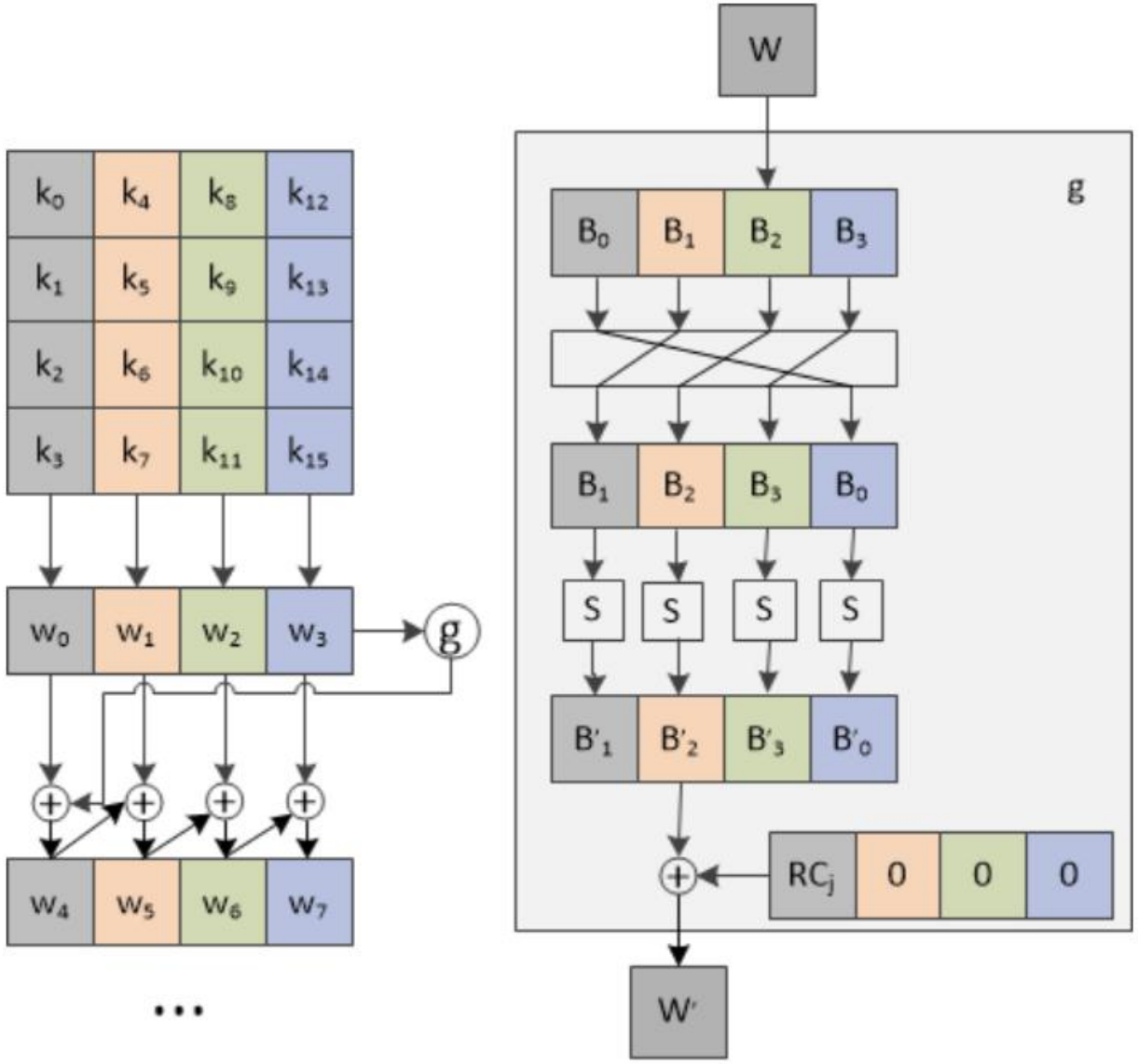


1-《Fast AES Implementation: A High-Throughput Bitsliced Approach》

1.2 Introduction of AES

AddRoundKey

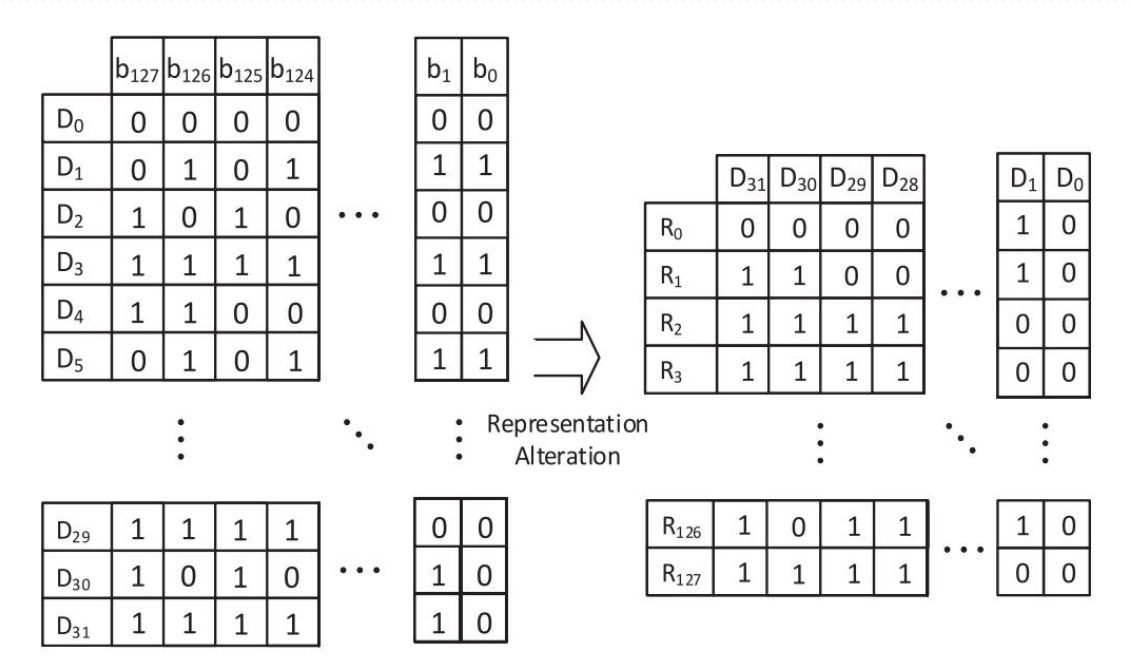
- 10 Round SubKeys
- 128 bits Each SubKey
- Could be Pre-Calculated on CPU

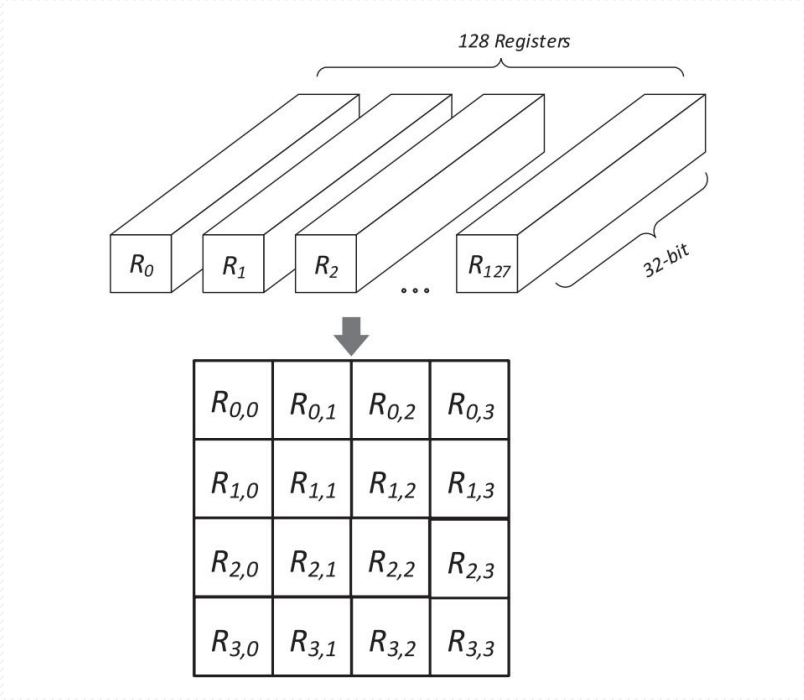


1-《Fast AES Implementation: A High-Throughput Bitsliced Approach》

1.3 BitSliced Improvement

row-major to column-major representation





FASTER

- eliminates t shift and mask operations

SAFER

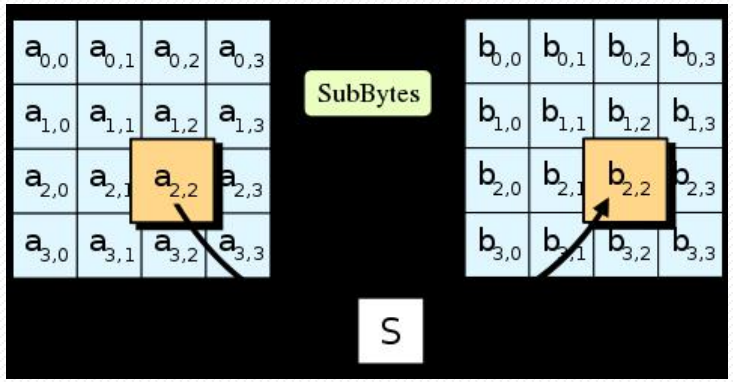
- S-box calculation immunes our AES implementation to timing attacks

1-《Fast AES Implementation: A High-Throughput Bitsliced Approach》

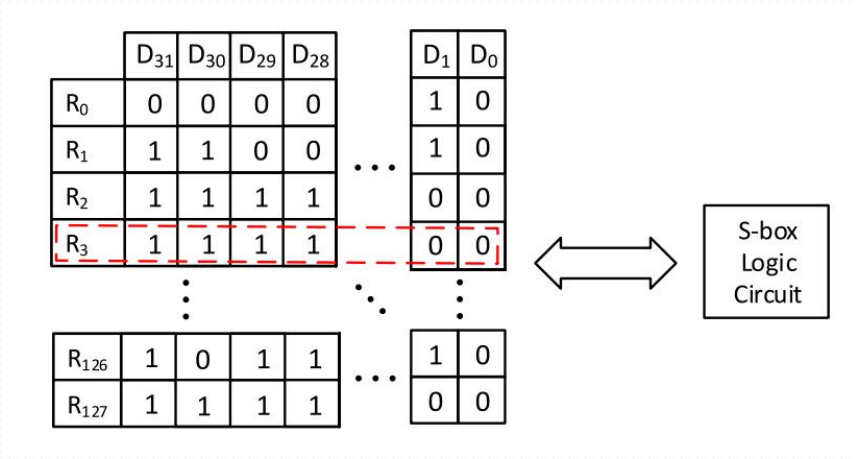
1.3 BitSliced Improvement

- SubBytes Stage

LUT Implementation



Bs Implementation



1-《Fast AES Implementation: A High-Throughput Bitsliced Approach》

1.3 BitSliced Improvement

- SubBytes Stage

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}.$$

```
SBOX MACRO x7, x6, x5, x4, x3, x2, x1, x0, t6, t5, t4, t3, t2, t1, t0
  MOV t0,x1   XOR x5,x6   XOR t0,x7   XOR x5,x0   XOR x6,x1   XOR x7,x5   XOR x6,x3
  XOR x1,x5   XOR x6,x0   XOR x3,x4   XOR x6,x2   XOR x4,x5   XOR x2,t0   XOR x3,x0
  XOR x2,x5   XOR x3,t0   MOV t1,x4   MOV t2,x2   XOR t1,x7   XOR t2,x1
  MOV [RSP-16],t1   MOV t4,t1   MOV t5,x0   MOV [RSP-8],t2   MOV t6,x3   XOR t5,x6
  XOR t6,x5   MOV [RSP-32],t5   XOR t4,t2   AND t1,t5   MOV [RSP-24],t6 AND t2,t6
  MOV [RSP-40],t4   XOR t5,t6   XOR t2,t1   AND t4,t5   MOV [RSP-48],t5 MOV t6,x2
  XOR t1,t4   AND t6,x3   MOV [RSP-56],t1   MOV t0,x3   MOV t1,x2   XOR t0,x0
  XOR t1,x4   MOV t5,x4   MOV [RSP-64],t1   MOV t3,x7   AND t1,t0   AND t3,x6
  MOV [RSP-72],t0   AND t5,x0   XOR t6,t1   XOR t5,t1   MOV [RSP-96],x6 MOV t1,x1
  MOV t0,x5   XOR t1,x7   XOR t0,x6   MOV [RSP-80],t1   XOR t5,t2   MOV t4,x1
  MOV [RSP-104],x5   AND t1,t0   AND t4,x5   XOR x6,x7   MOV [RSP-88],t0
  XOR t4,t1   XOR t3,t1   XOR x5,x1   MOV t1,[RSP-56]   XOR t3,t2   XOR t5,x5
  XOR t3,x6   XOR t5,x2   XOR t1,x6   XOR t6,x0   XOR t4,t1   XOR t6,t1   XOR t4,x5
  XOR t6,x4   MOV t2,t4   XOR t5,x3   XOR t2,t3   MOV x6,t5   MOV t0,t5   XOR x6,t6
  AND t0,t3   MOV x5,t2   XOR t0,t3   AND x5,x6   XOR t0,t5   XOR x5,t6   MOV t1,t6
  XOR x5,t4   AND t1,t4   XOR t0,x5   XOR t1,x5   AND t6,t0   AND t4,t0
  MOV x5,[RSP-104]   AND t5,t1   XOR t0,t1   AND t3,t1   AND x6,t0   AND t2,t0
  XOR t6,x6   XOR t4,t2   XOR t5,x6   XOR t3,t2   MOV t1,t6   MOV x6,[RSP-96]
  AND x3,t4   XOR t1,t4   MOV t2,[RSP-24]   AND x2,t4   AND [RSP-8],t1
  AND x5,t6   XOR t4,t3   AND x1,t6   AND [RSP-64],t4   AND x0,t3   XOR t6,t5
  AND t4,[RSP-72]   AND x4,t3   AND [RSP-80],t6   AND x7,t5   XOR t3,t5
  AND t6,[RSP-88]   AND x6,t5   AND t2,t1   AND [RSP-16],t3   XOR x5,t6
  MOV t5,[RSP-64]   XOR x6,t6   MOV t6,t1   XOR x3,t4   XOR t1,t3 AND t3,[RSP-32]
  XOR x0,t4   XOR x2,t5   AND [RSP-40],t1   XOR t2,t3   AND t1,[RSP-48] XOR x0,t2
  MOV t0,[RSP-8]   XOR t1,t3   XOR x4,t5   MOV t3,[RSP-80]   XOR x6,t2   XOR x3,t1
  XOR x5,t1   MOV t6,x0   MOV t1,[RSP-16]   XOR x0,x6   XOR x1,t3   XOR t0,t1
  XOR x7,t3   XOR t1,[RSP-40]   XOR x4,t0   XOR x7,t0   XOR x1,t1   XOR x2,t1
  MOV t2,x1   MOV t1,x4   XOR x1,x6   MOV x4,x5   MOV x6,x2   XOR x1,x5   XOR x6,x3
  MOV t0,x7   XOR x4,x6   MOV x3,x0   MOV x7,x5   XOR x3,x4   XOR x7,x2   MOV x5,t6
  MOV x2,x7   XOR x5,t0   XOR x2,t1   XOR x0,t2   XOR x2,x5
ENDM
```

- S-box calculation having 253 logic gates
- uses subfields of 4 and 2-bit for the 8-bit S-box calculations [1]

[1] D. Canright, "A very compact S-box for AES," in Proc. Int. Workshop Cryptographic Hardware Embedded Syst., 2005, pp. 441–455.

1-《Fast AES Implementation: A High-Throughput Bitsliced Approach》

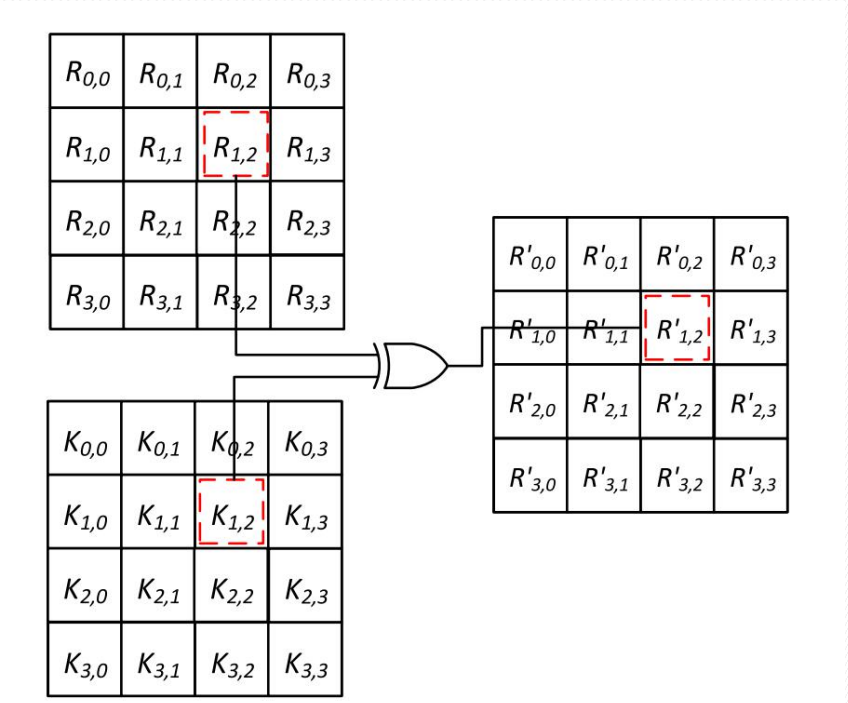
1.3 BitSliced Improvement

- AddRoundKey Stage

The value of each of the 128 bits of a single round key should be copied 32 times to fill each register from the total of the 128 registers. As the value of the round key bits are duplicated, each of the 128 round key registers are either 0xFFFFFFFF or 0x00000000

store the round keys and to keep the values for all the round
keys in the GPU's global memory.

storing the round keys in the global memory and fetching
them into the shared memory at the end of each round



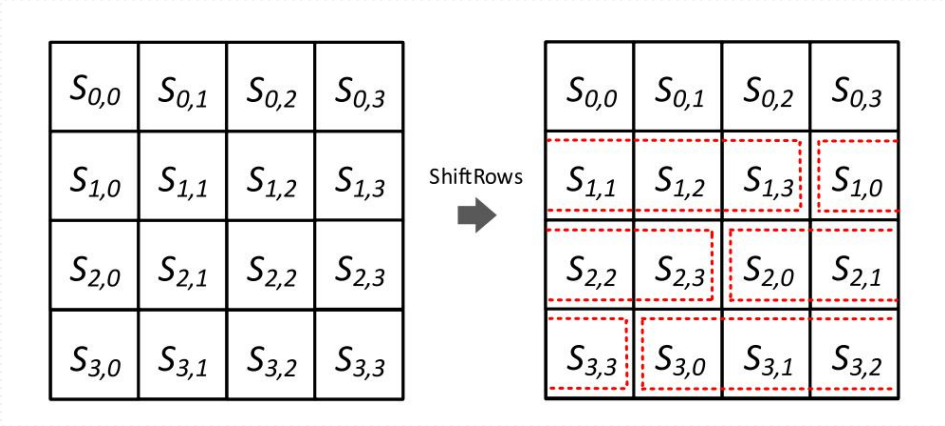
Using PyCUDA(Scripting Manner) to diminish I/O loss

1-《Fast AES Implementation: A High-Throughput Bitsliced Approach》

1.3 BitSliced Improvement

- ShiftRows Stage

LUT Implementation



Bs Implementation

ShiftRows is implemented by register shift and swapping
instead of the costly Byte shift and swapping
no need to perform any explicit shift operations

1-《Fast AES Implementation: A High-Throughput Bitsliced Approach》

1.3 BitSliced Improvement

- MixColums Stage

LUT Implementation

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$

$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$

$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}).$$

Bs Implementation

Algorithm 1. Multiplication of X by 2 in $GF(2^8)$

Input: $X = x_0, x_1, \ldots,$ *and* x_7
Output: $2 \times X$
{ $Carry = x_7$
$x_7 = (x_6)$
$x_6 = (x_5)$
$x_5 = (x_4)$
$x_4 = (x_3) \oplus \text{Carry}$
$x_3 = (x_2) \oplus \text{Carry}$
$x_2 = (x_1)$
$x_1 = (x_0) \oplus \text{Carry}$
$x_0 = \text{Carry}$
}

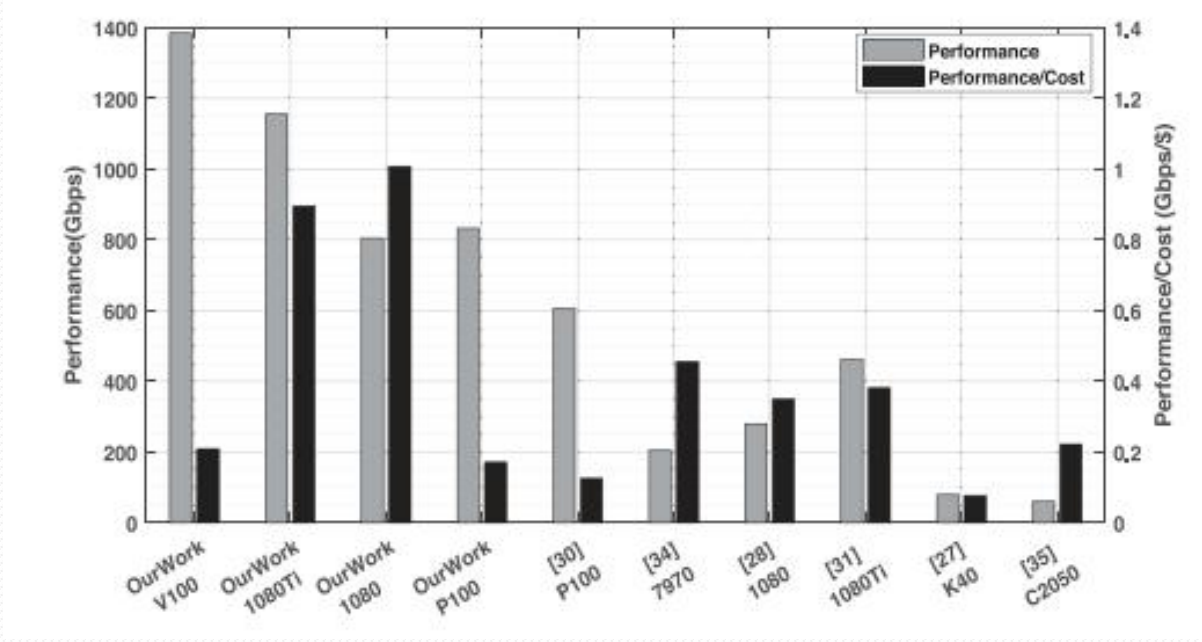
$$\begin{aligned} R'_{i,j} &= 2 \times R_{i,j} \oplus 3 \times R_{|i+1|_4,j} \oplus R_{|i+2|_4,j} \oplus R_{|i+3|_4,j} \\ &= 2 \times (R_{i,j} \oplus R_{|i+1|_4,j}) \oplus R_{|i+1|_4,j} \oplus R_{|i+2|_4,j} \oplus R_{|i+3|_4,j} \end{aligned}$$

Other Works &*Evaluation Results*

- Implementation of ECB and CTR operation modes
- Decryption Implementation
- Implementation on Multi-GPU

ECB Encryption Throughput

Name	Throughput (Gbits per sec.)	Performance /GPU cost (Gbits per sec./$)
GTX 980	489.44	1.529
GTX 1050 Ti	296.74	1.484
GTX 1080	805.16	1.008
GTX 1080 Ti	1156.07	0.896
Tesla P100	832.58	0.171
Tesla V100	**1384.51**	0.209



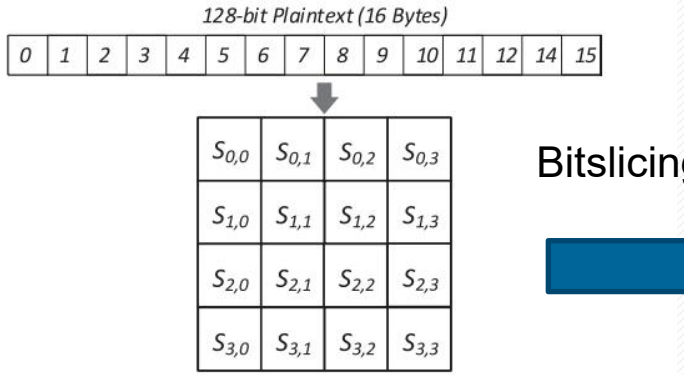
*Operation Modes[2]

- ✓ ECB - Electronic Code Book
- ✓ CBC - Cipher Block Chaining
- ✓ CFB - Cipher FeedBack
- ✓ OFB - Output-Feedback
- ✓ CTR - CounTeR

[2] https://www.jianshu.com/p/b900ad0cc150

2- Other Bitslicing Implement

On Intel Core i7 Platform



Bitslicing to Byteslicing

8×128 bits Registers

	row 3							·····	row 0						
	column 0			···	column 3			·····	column 0			···	column 3		
	block 0	···	block 7	···	block 0	···	block 7	·····	block 0	···	block 7	···	block 0	···	block 7
R_0	b_{24}^0	···	b_{24}^7	···	b_{120}^0	···	b_{120}^7	·····	b_0^0	···	b_0^7	···	b_{96}^0	···	b_{96}^7
R_1	b_{25}^0	···	b_{25}^7	···	b_{121}^0	···	b_{121}^7	·····	b_1^0	···	b_1^7	···	b_{97}^0	···	b_{97}^7
R_2	b_{26}^0	···	b_{26}^7	···	b_{122}^0	···	b_{122}^7	·····	b_2^0	···	b_2^7	···	b_{98}^0	···	b_{98}^7
R_3	b_{27}^0	···	b_{27}^7	···	b_{123}^0	···	b_{123}^7	·····	b_3^0	···	b_3^7	···	b_{99}^0	···	b_{99}^7
R_4	b_{28}^0	···	b_{28}^7	···	b_{124}^0	···	b_{124}^7	·····	b_4^0	···	b_4^7	···	b_{100}^0	···	b_{100}^7
R_5	b_{29}^0	···	b_{29}^7	···	b_{125}^0	···	b_{125}^7	·····	b_5^0	···	b_5^7	···	b_{101}^0	···	b_{101}^7
R_6	b_{30}^0	···	b_{30}^7	···	b_{126}^0	···	b_{126}^7	·····	b_6^0	···	b_6^7	···	b_{102}^0	···	b_{102}^7
R_7	b_{31}^0	···	b_{31}^7	···	b_{127}^0	···	b_{127}^7	·····	b_7^0	···	b_7^7	···	b_{103}^0	···	b_{103}^7

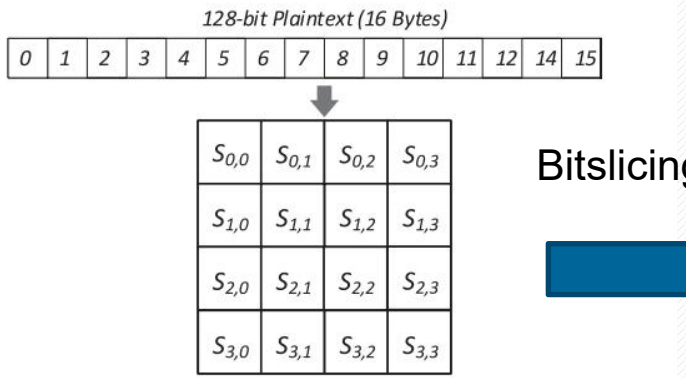
Figure 3: Bitsliced representation from [KS09] using 8 128-bit registers $R_0, \cdots, R_7$ to process 8 blocks $b^0, \cdots, b^7$ in parallel where b_j^i refers to the j-th bit of the i-th block.

Reached 6.9 cpb for AES-128 on Intel Core i7 processors by processing 8 blocks in parallel[3]

[3] Emilia Käsper and Peter Schwabe. Faster and Timing-Attack Resistant AES GCM. In Christophe Clavier and Kris Gaj, editors, Cryptographic Hardware and Embedded Systems pages 1–17, Berlin, Heidelberg, 2009.Springer Berlin Heidelberg

2- Other Bitslicing Implement

On Arm Cortex-M3 Platform



Bitslicing to Byteslicing

	row 3								.....	row 0								
	column 0		column 1		column 2		column 3		.....	column 0		column 1		column 2		column 3		
	block 0	block 1	block 0	block 1	block 0	block 1	block 0	block 1	.....	block 0	block 1	block 0	block 1	block 0	block 1	block 0	block 1	
R_0	b_{24}^0	b_{24}^1	b_{56}^0	b_{56}^1	b_{88}^0	b_{88}^1	b_{120}^0	b_{120}^1	.....	b_0^0	b_0^1	b_{32}^0	b_{32}^1	b_{64}^0	b_{64}^1	b_{96}^0	b_{96}^1	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	.....	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	
R_7	b_{31}^0	b_{31}^1	b_{63}^0	b_{63}^1	b_{95}^0	b_{95}^1	b_{127}^0	b_{127}^1	.....	b_7^0	b_7^1	b_{39}^0	b_{39}^1	b_{71}^0	b_{71}^1	b_{103}^0	b_{103}^1	

8×32 bits Registers

On 32-bit platforms, the most efficient bitsliced AES implementation reported in the literature is th
Schwabe and Stoffelen allowing to reach 101 cpb on ARM Cortex-M3, their implementation heavily

[4] Emilia Käsper and Peter Schwabe. Faster and Timing-Attack Resistant AES GCM. In Christophe Clavier and Kris Gaj, editors, Cryptographic Hardware and Embedded Systems pages 1–17, Berlin, Heidelberg, 2009.Springer Berlin Heidelberg

Fixslicing Implementation

	row 3								·····	row 0							
	column 0		column 1		column 2		column 3		·····	column 0		column 1		column 2		column 3	
	block 0	block 1	block 0	block 1	block 0	block 1	block 0	block 1	·····	block 0	block 1	block 0	block 1	block 0	block 1	block 0	block 1
R_0	b_{24}^0	b_{24}^1	b_{56}^0	b_{56}^1	b_{88}^0	b_{88}^1	b_{120}^0	b_{120}^1	·····	b_0^0	b_0^1	b_{32}^0	b_{32}^1	b_{64}^0	b_{64}^1	b_{96}^0	b_{96}^1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	·····	⋮	⋮	⋮	⋮	⋮	·	·	·
R_7	b_{31}^0	b_{31}^1	b_{63}^0	b_{63}^1	b_{95}^0	b_{95}^1	b_{127}^0	b_{127}^1	·····	b_7^0	b_7^1	b_{39}^0	b_{39}^1	b_{71}^0	b_{71}^1	b_{103}^0	b_{103}^1

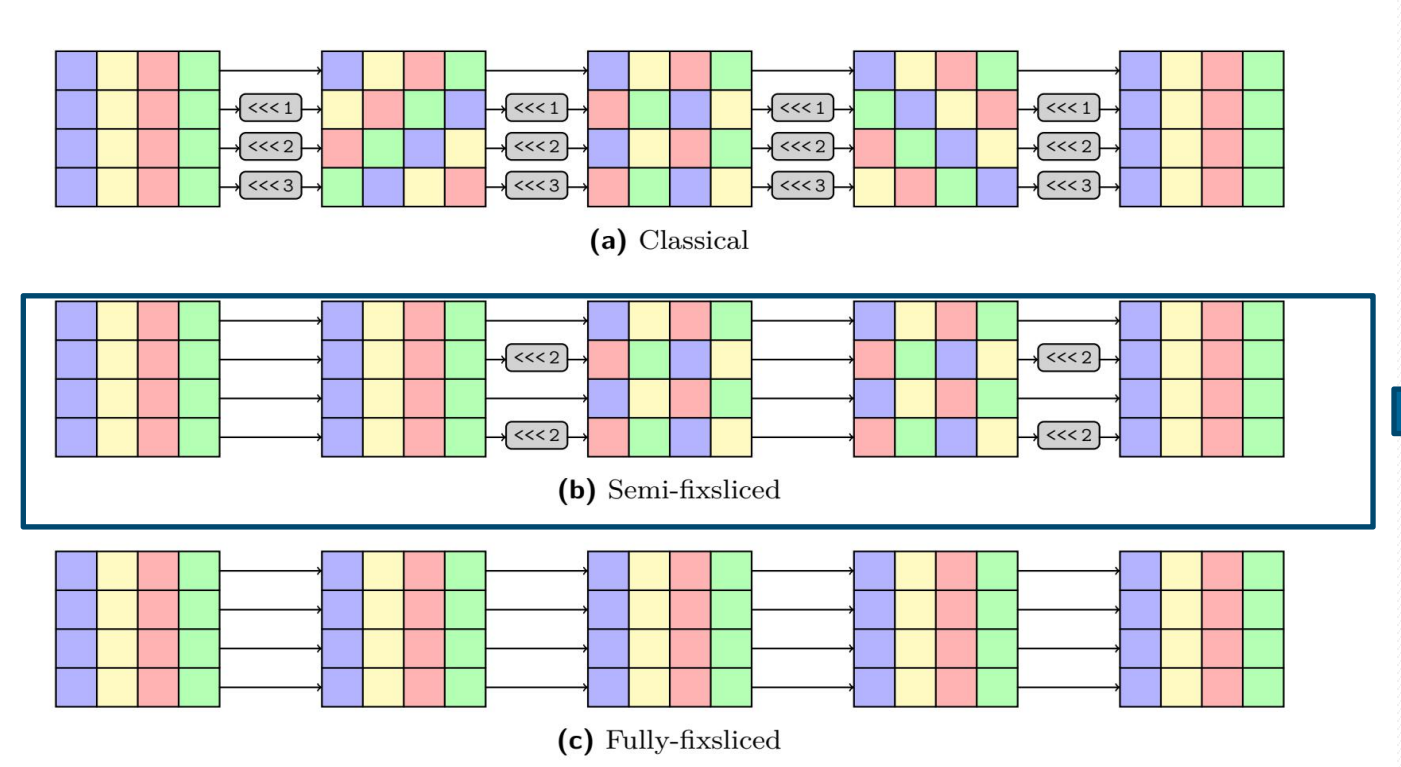
8×32 bits Registers

Byte to Column-Major

- *Fixslicing*[5] mainly consists in fixing the bits within a register (or slice) to never move and to adjust the other slices accordingly so that the proper bits are involved in the SubBytes operation.

[5] Alexandre Adomnicai, Thomas Peyrin:Fixslicing AES-like Ciphers New bitsliced AES speed records on ARM-Cortex M and RISC-V. IACR Trans. Cryptogr. Hardw. Embed. Syst. 2021(1): 402-425 (2021)

Fixslicing Implementation



FixSliced

- *As it can be proved that F = ShiftRows²(F), fixing the bits within a register*
- *barrel-shiftrows : byte-wise rotation per register*
- *relies on a quadruple round, only one AES version has a number of rounds which is a multiple of 4, namely AES-192, AES-128 and AES-256 needs additional transformation*