

The background features a dynamic, abstract composition of glowing binary digits (0s and 1s) and light trails in shades of blue, orange, and red, creating a sense of motion and digital data flow.

EFFICIENT ADDITIONS AND MONTGOMERY REDUCTIONS OF LARGE INTEGERS FOR SIMD

Pengchang REN

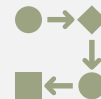
INTRODUCTION



Large prime field arithmetic (e.g. 511-bit) is used by many post-quantum cryptography.



We want to optimize for ARM and x86.



We want to use SIMD for optimization.

SIMD INSTRUCTION LATENCY COMPARISON

	Tigerlake [1]		A64FX [2]	
Instruction set	x64	AVX-512	A64	SVE
Vector length	-	512 bit	-	512 bit
Integer multiplication support	~64-bit	~52-bit	~64-bit	~64-bit
Addition latency	1 cycle	1 cycle	1 cycle	4 cycles
Integer multiplication latency (Input size ->Output size)	3 cycles 64-bit->128-bit	4 cycles 52-bit->52-bit	5 cycles 64-bit->64-bit	9 cycles 64-bit->64-bit
Table lookup latency	-	3 cycles	-	6 cycles

High instruction latency even for the easiest type of instruction

[1] A.Fog, "Instruction tables: List of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs (2012),"

[2] A64FX Microarchitecture Manual, Fujitsu, 2022, revision 1.8.1.

WHAT WE DID

Proposal 1: A SIMD **addition** algorithm for SVE

Proposal 2: An optimized algorithm for **Montgomery reduction** for SIMD by reducing data dependency

Proposal 3: A **Montgomery reduction** algorithm for specific prime field to utilize Karatsuba method



PROPOSAL 1
LARGE INTEGER ADDITION FOR SVE

ADD WITH CARRY (1)

- E.g. Calculating $2023 + 6789$

$$\begin{array}{r} 2023 \\ + 6789 \\ \hline \end{array}$$

ADD WITH CARRY (3)

- E.g. Calculating $2023 + 6789$

- Addition: $3 + 9 \rightarrow 12$

- Add-with-carry: $2 + 8 + 1 \rightarrow \underline{11}$

Carry-In (C_{in})

Carry-Out (C_{out})

Carry \longrightarrow

Sum \longrightarrow

$$\begin{array}{r} 2 2 \\ + 6 8 \\ \hline 1 2 \end{array}$$

ADD WITH CARRY (4)

- E.g. Calculating $2023 + 6789$

- Addition: $3 + 9 \rightarrow 12$

- Add-with-carry: $2 + 8 + 1 \rightarrow 11$



- Add-with-carry: $0 + 7 + 1 \rightarrow 08$

- Add-with-carry: $2 + 6 + 0 \rightarrow 08$

Carry \longrightarrow

	0	0	1	1
	2	0	2	3
+	6	7	8	9
<hr style="border: 0.5px solid black;"/>				
	0	8	8	1
				2

Sum \longrightarrow

LARGE INTEGER ADDITION

$$\begin{array}{r}
 0 \quad 0 \quad 1 \quad 1 \\
 2 \quad 0 \quad 2 \quad 3 \\
 + 6 \quad 7 \quad 8 \quad 9 \\
 \hline
 0 \quad 8 \quad 8 \quad 1 \quad 2
 \end{array}$$

- SISD
- Addition: $3 + 9 \rightarrow 12$
- Add-with-carry: $2 + 8 + 1 \rightarrow 11$
- Add-with-carry: $0 + 7 + 1 \rightarrow 08$
- Add-with-carry: $2 + 6 + 0 \rightarrow 08$

- SIMD, naïve way
- Addition: $2|0|2|3 + 6|7|8|9 \rightarrow 08|07|10|12$
Additional instruction required
- Addition: $8|7|0|2 + 0|1|1|0 \rightarrow 08|08|01|02$
- Addition: $8|8|1|2 + 0|0|0|0 \rightarrow 08|08|01|02$
- Addition: $8|8|1|2 + 0|0|0|0 \rightarrow 08|08|01|02$

More instructions and dependency



CARRY SELECT ADDER[3]

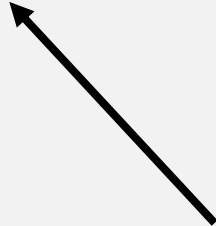
- A hardware implementation of addition in parallel.
- How to select efficiently?
- Our idea: select it by a smaller addition
(Explained in next page).

[3] Bedrij, O. J. (1962). Carry-select adder. *IRE Transactions on Electronic Computers*, (3), 340-346.

16-bit addition

	0000	1111	1111	1111
	0000	0000	1111	0000
+	0000	0000	1111	0000
$C_{in}=0$	00000	01111	11110	01111
$C_{in}=1$	00001	10000	11111	10000
Select	????	????	????	????

In hardware: Select by carry-lookahead



OUR IMPLEMENTATION (1)

- How to select? Calculate a smaller addition
- Conversion:
 - Case **N**: $C_{out} = 0$
 - Case **P**: $C_{out} = C_{in}$
 - Case **G**: $C_{out} = 1$

16-bit addition

	0000	1111	1111	1111
+	0000	0000	1111	0000
$C_{in}=0$	00000	01111	11110	01111
$C_{in}=1$	00001	10000	11111	10000
Case	N	P	G	P
Select	????	????	????	????

OUR IMPLEMENTATION (2)

- How to select? Calculate a smaller addition
- Conversion:
 - Case **N**: $C_{out} = 0 \rightarrow 0 + 0$
 - Case **P**: $C_{out} = C_{in} \rightarrow 1 + 0$
 - Case **G**: $C_{out} = 1 \rightarrow 1 + 1$

16-bit addition

	0000	1111	1111	1111
+	0000	0000	1111	0000
$C_{in}=0$	00000	01111	11110	01111
$C_{in}=1$	00001	10000	11111	10000
Case	N	P	G	P
Select	????	????	????	????



OUR IMPLEMENTATION (3)

- How to select? Calculate a smaller addition
- Conversion:
 - Case **N**: $C_{out} = 0 \rightarrow 0 + 0$
 - Case **P**: $C_{out} = C_{in} \rightarrow 1 + 0$
 - Case **G**: $C_{out} = 1 \rightarrow 1 + 1$

16-bit addition

	0000	1111	1111	1111
+	0000	0000	1111	0000
$C_{in}=0$	00000	01111	11110	01111
$C_{in}=1$	00001	10000	11111	10000
Case	N	P	G	P
Select	????	????	????	????

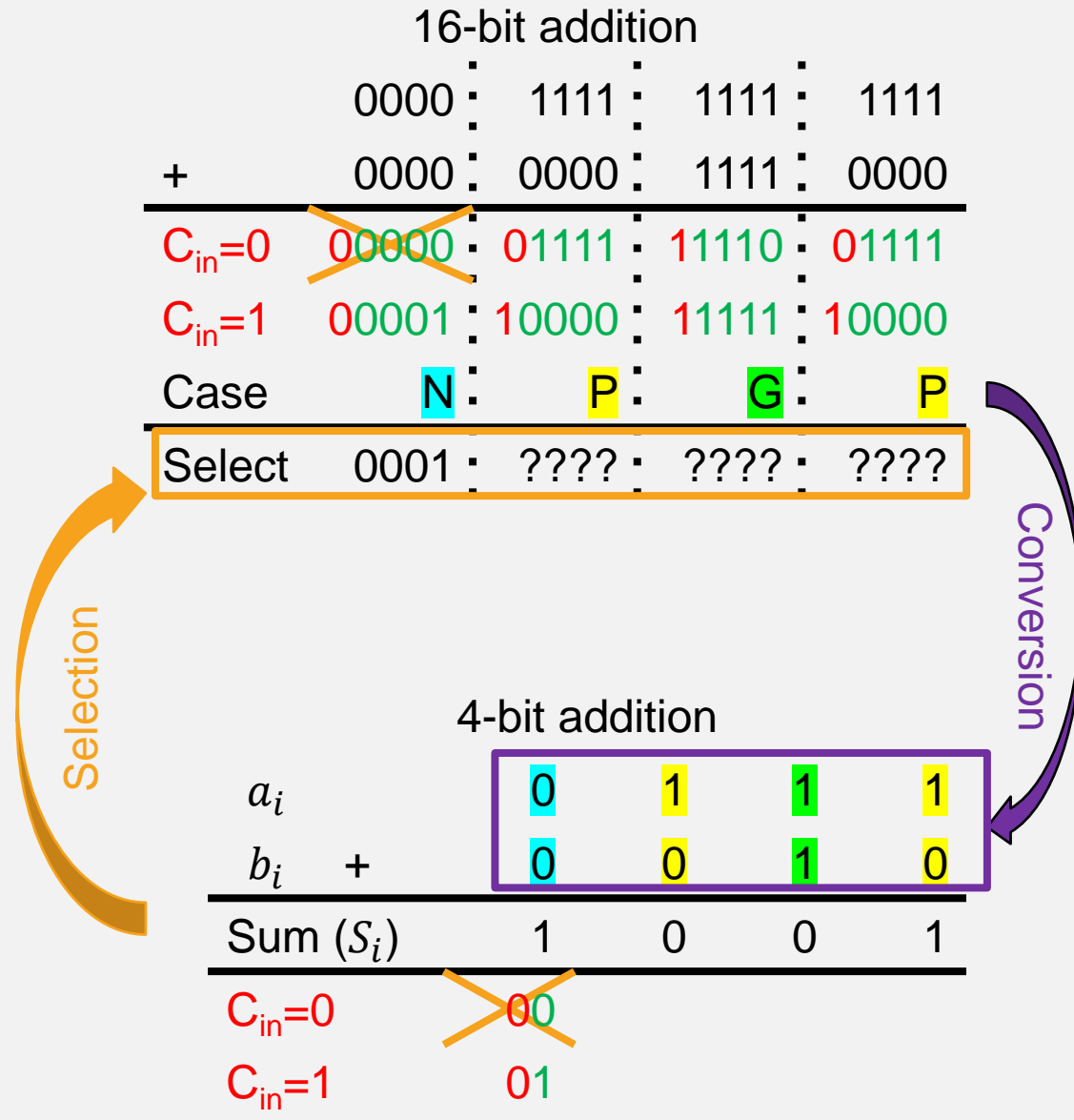
4-bit addition

a_i	0	1	1	1
b_i	0	0	1	0
+				
Sum (S_i)	1	0	0	1

Conversion

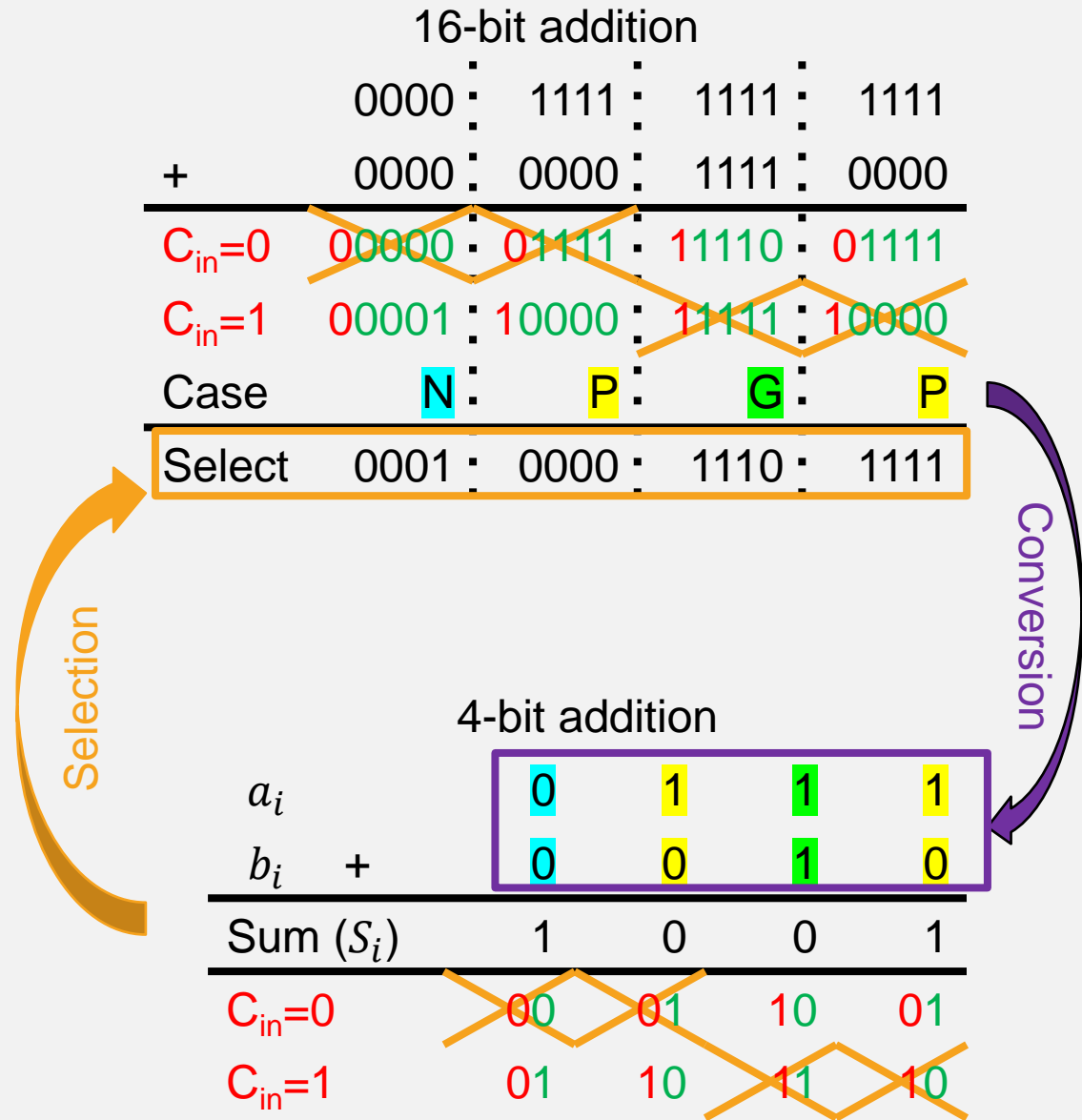
OUR IMPLEMENTATION (4)

- How to select? Calculate a smaller addition
- Conversion:
 - Case **N**: $C_{out} = 0 \rightarrow 0 + 0$
 - Case **P**: $C_{out} = C_{in} \rightarrow 1 + 0$
 - Case **G**: $C_{out} = 1 \rightarrow 1 + 1$
- Selection:
 - $S_i = a_i + b_i + C_{in}$ by definition, so $C_{in} = S_i - a_i - b_i$



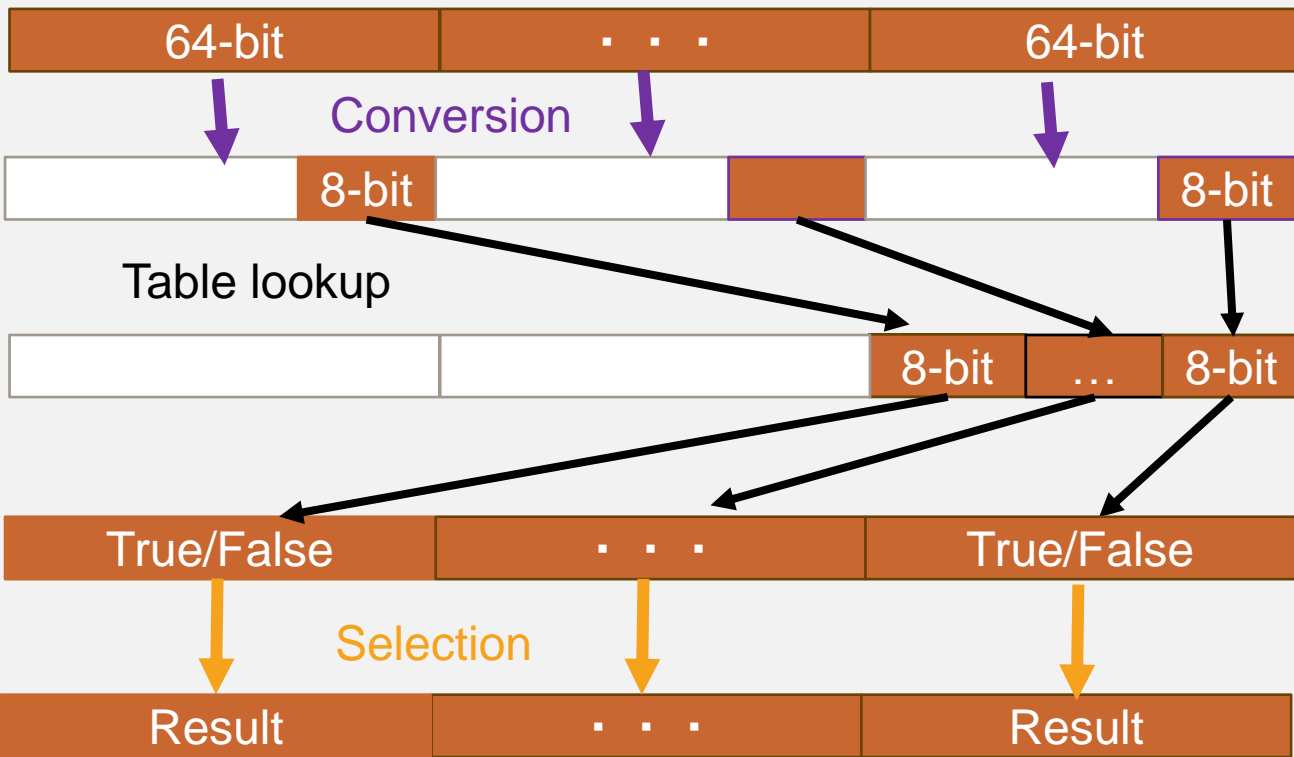
OUR IMPLEMENTATION (5)

- How to select? Calculate a smaller addition
- **Conversion:**
 - Case **N**: $C_{out} = 0 \rightarrow 0 + 0$
 - Case **P**: $C_{out} = C_{in} \rightarrow 1 + 0$
 - Case **G**: $C_{out} = 1 \rightarrow 1 + 1$
- **Selection:**
 - $S_i = a_i + b_i + C_{in}$ by definition, so $C_{in} = S_i - a_i - b_i$
- **No dependency between words.**



REAL IMPLEMENTATION ON SVE

- How to convert with SVE? 64-bit example.

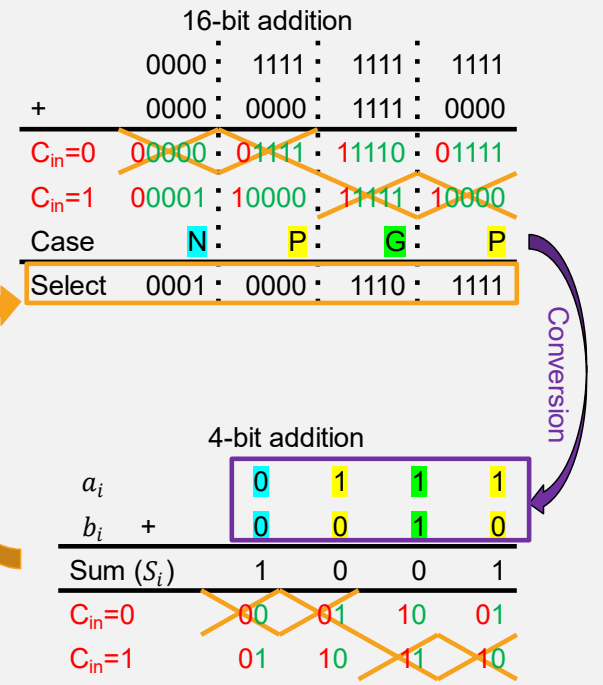


512-bit addition

8-bit smaller addition for each 64-bit word

One 64-bit addition

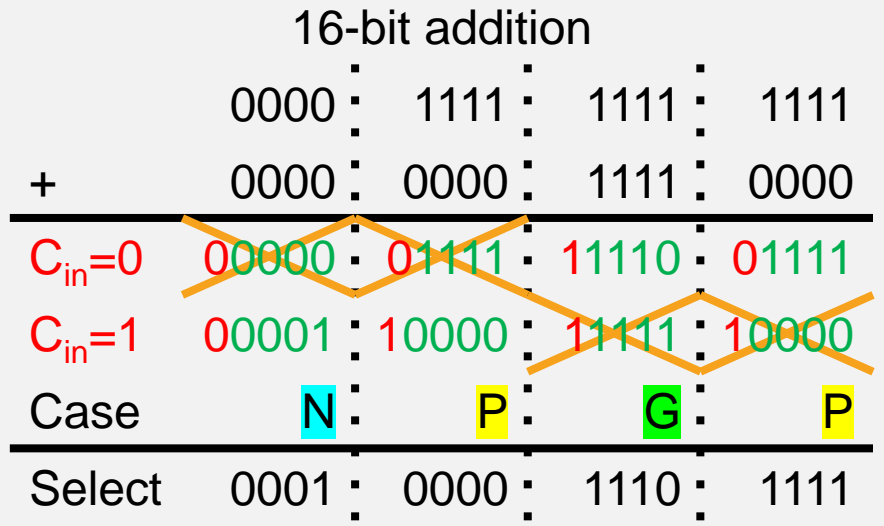
Selector



OUR IMPLEMENTATION HOW WE CONVERT

- 64-bit example.

	Case N	Case P	Case G
A	0	0	1
B	-2	-1	-1
$D = A + B$	-2	-1	0
$G = \text{popcnt}(D)$	63	64	0
$m = D < A$	False	False	True
$t = \text{mADD}(G, 65, m)$	63 + 0	64 + 0	0 + 65
p	191	191	191
$s = t + p$	254	255	256

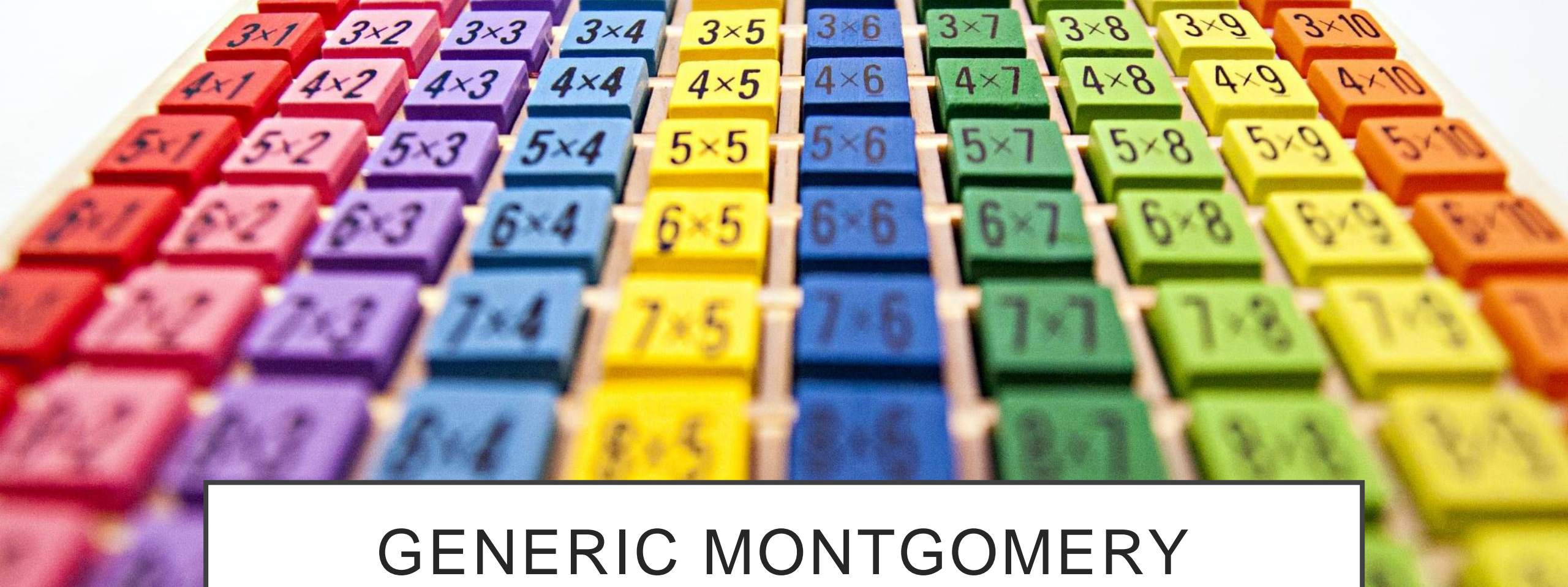


```

Algorithm
-----
 $G_i \leftarrow \text{popcnt}(D_i)$ 
 $m_i \leftarrow (D_i < A_i)$ 
 $t_i \leftarrow \text{mADD}(G_i, 65, m_i)$ 

```

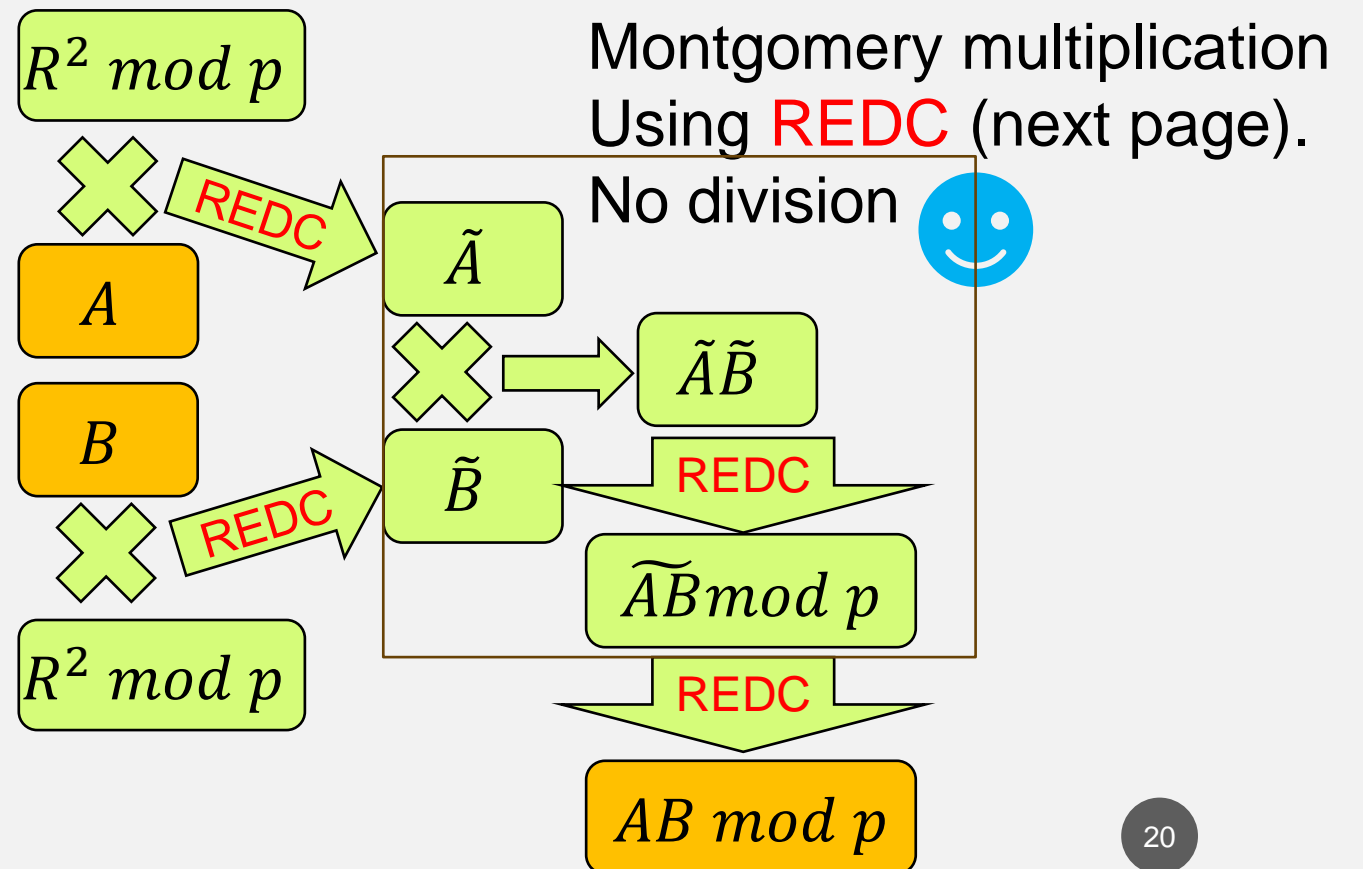
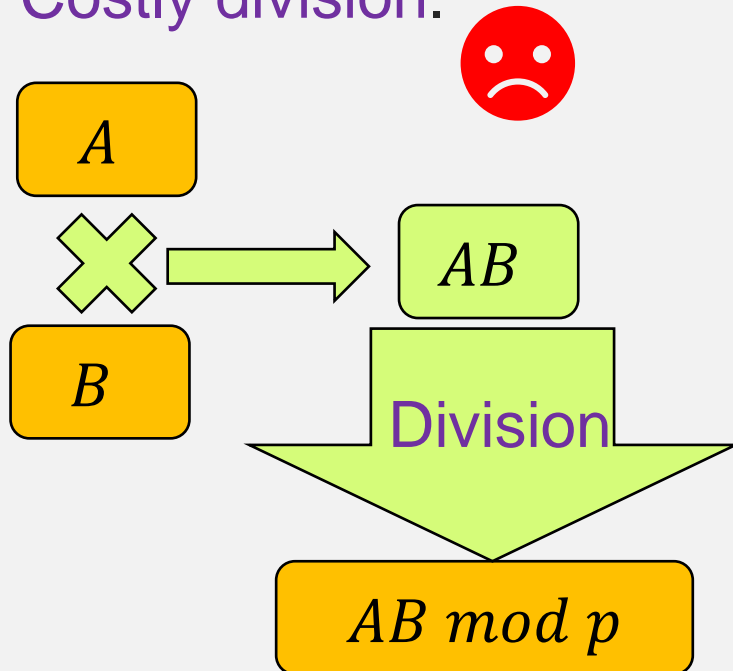
If $m = \text{True}$: $t \leftarrow G + 65$
Else: $t \leftarrow G$



GENERIC MONTGOMERY REDUCTION

MODULAR MULTIPLICATION

- Naïve
- Costly division.



GENERIC MONTGOMERY REDUCTION[4]

$$T < pR$$

$$p < R = r^n = 2^{\omega n}$$

$$T^{(i)} \equiv Tr^{-i} \pmod{p}$$

[4] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of computation*, vol. 44, no. 170, pp. 519–521, 1985.

Word length ω

Algorithm ExistingGenericRedc: Generic Montgomery reduction

Input: $T < pR$, where $p < 2^{\omega n}$, $r = 2^\omega$, $R = 2^{\omega n}$,
 r^{-1}, R^{-1} are positive integers such that
 $rr^{-1} \equiv 1 \pmod{p}$, $RR^{-1} \equiv 1 \pmod{p}$, and
 $p' \leftarrow \frac{rr^{-1}-1}{p}$

Output: $REDC(T) = TR^{-1} \pmod{p}$

- 1 $T^{(0)} \leftarrow T$
- 2 **for** $i = 1$ **to** n **do**
- 3 $Q \leftarrow T^{(i-1)}p' \pmod{r}$
- 4 $T^{(i)} \leftarrow (T^{(i-1)} + Qp)/r$
- 5 **end**
- 6 **if** $T^{(n)} > p$ **then**
- 7 $T^{(n)} \leftarrow T^{(n)} - p$
- 8 **return** $T^{(n)}$

For the selected R

Calculate this

GENERIC MONTGOMERY REDUCTION

$$T < pR$$

$$p < R = r^n = 2^{\omega n}$$

$$T^{(i)} \equiv Tr^{-i} \pmod{p}$$

Algorithm ExistingGenericRedc: Generic Montgomery reduction

Input: $T < pR$, where $p < 2^{\omega n}$, $r = 2^{\omega}$, $R = 2^{\omega n}$,
 r^{-1} , R^{-1} are positive integers such that
 $rr^{-1} \equiv 1 \pmod{p}$, $RR^{-1} \equiv 1 \pmod{p}$, and
 $p' \leftarrow \frac{rr^{-1}-1}{p}$

Output: $REDC(T) = TR^{-1} \pmod{p}$

```

1  $T^{(0)} \leftarrow T$ 
2 for  $i = 1$  to  $n$  do
3   |  $Q \leftarrow T^{(i-1)}p' \pmod{r}$ 
4   |  $T^{(i)} \leftarrow (T^{(i-1)} + Qp)/r$ 
5 end
6 if  $T^{(n)} > p$  then
7   |  $T^{(n)} \leftarrow T^{(n)} - p$ 
8 return  $T^{(n)}$ 

```

← $T^{(i)} \leftarrow T^{(i-1)}r^{-1} \pmod{p}$

← $REDC(T) = T^{(i)} = Tr^{-n} \pmod{p}$

GENERIC MONTGOMERY REDUCTION DEPENDENCY

$$T < pR$$

$$p < R = r^n = 2^{\omega n}$$

$$T^{(i)} \equiv Tr^{-i} \pmod{p}$$

Algorithm ExistingGenericRedc: Generic Montgomery reduction

Input: $T < pR$, where $p < 2^{\omega n}$, $r = 2^{\omega}$, $R = 2^{\omega n}$,
 r^{-1}, R^{-1} are positive integers such that
 $rr^{-1} \equiv 1 \pmod{p}$, $RR^{-1} \equiv 1 \pmod{p}$, and
 $p' \leftarrow \frac{rr^{-1}-1}{p}$

Output: $REDC(T) = TR^{-1} \pmod{p}$

- 1 $T^{(0)} \leftarrow T$
- 2 **for** $i = 1$ **to** n **do**
- 3 $Q \leftarrow T^{(i-1)}p' \pmod{r}$ $T^{(i-1)} \rightarrow Q$
- 4 $T^{(i)} \leftarrow (T^{(i-1)} + Qp)/r$ $Q \rightarrow T^{(i)}$
- 5 **end**
- 6 **if** $T^{(n)} > p$ **then** Data dependency
- 7 $T^{(n)} \leftarrow T^{(n)} - p$ $Q \rightarrow T^{(1)} \rightarrow Q \rightarrow T^{(2)} \rightarrow \dots \rightarrow Q \rightarrow T^{(n)}$
- 8 **return** $T^{(n)}$

PROBLEM OF DEPENDENCY

```
2 for i = 1 to n do
```

```
3   |  $Q \leftarrow T^{(i-1)}p' \bmod r$ 
```

```
4   |  $T^{(i)} \leftarrow (T^{(i-1)} + Qp)/r$ 
```

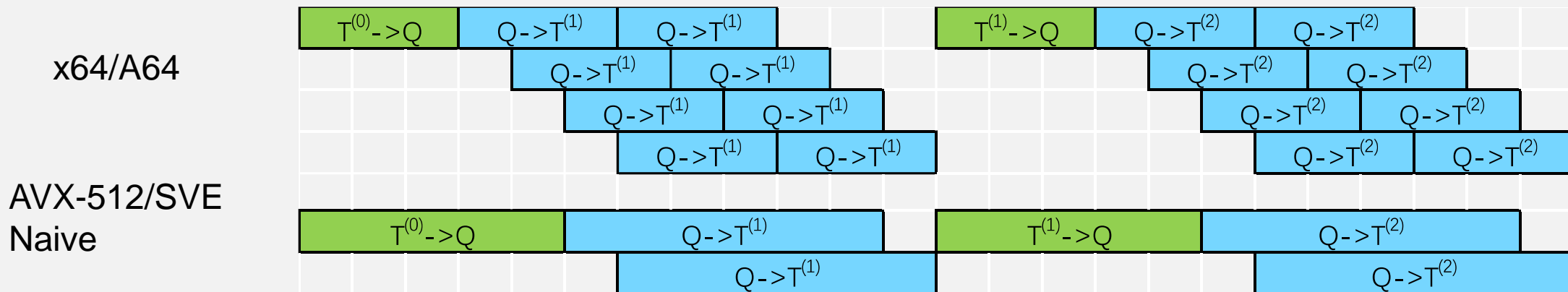
```
5 end
```

$T^{(i-1)} \rightarrow Q$

$Q \rightarrow T^{(i)}$

Enough parallelism

Time 



Just an illustration, not real ratio!
Additions are omitted for simplicity

PROBLEM OF DEPENDENCY

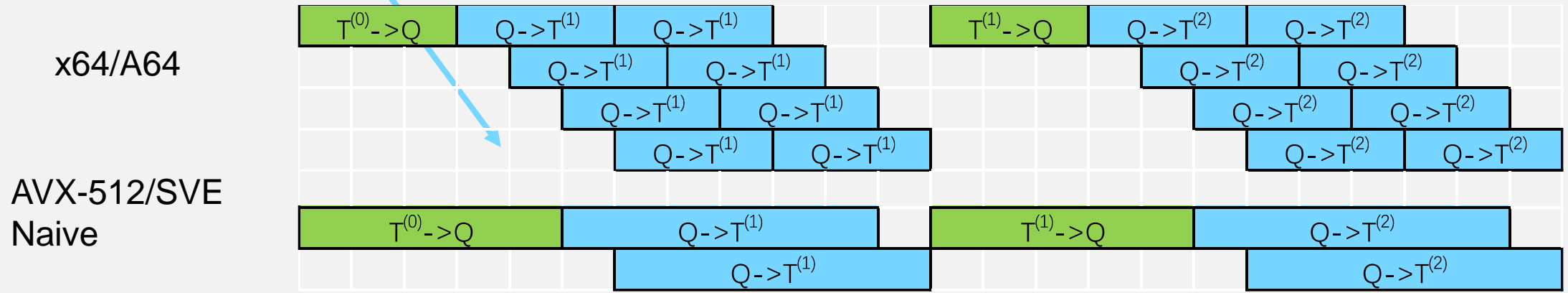
Less parallelism
Due to less instruction needed

```

2 for i = 1 to n do
3   |  $Q \leftarrow T^{(i-1)}p' \bmod r$ 
4   |  $T^{(i)} \leftarrow (T^{(i-1)} + Qp)/r$ 
5 end
    
```

$T^{(i-1)} \rightarrow Q$
 $Q \rightarrow T^{(i)}$

Time →



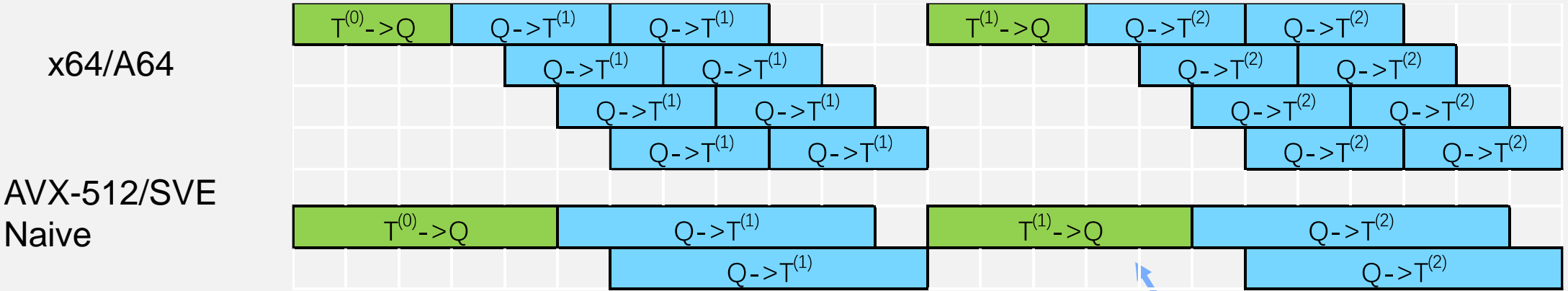
Just an illustration, not real ratio!
Additions are omitted for simplicity

PROBLEM OF DEPENDENCY

```

2 for i = 1 to n do
3   |  $Q \leftarrow T^{(i-1)}p' \bmod r$        $T^{(i-1)} \rightarrow Q$ 
4   |  $T^{(i)} \leftarrow (T^{(i-1)} + Qp)/r$    $Q \rightarrow T^{(i)}$ 
5 end
  
```

Time →



Just an illustration, not real ratio!
Additions are omitted for simplicity

More instruction needed than x64/A64

OUR PROPOSED REDC(T) WITH LESS DEPENDENCY

Since

$$T = \sum_{i=0}^{n-1} t_i r^i, t_i < r \text{ except } t_{n-1}$$

Then

$$\begin{aligned} \text{REDC}(T) &\equiv R^{-1} \sum_{i=0}^{n-1} t_i r^i \\ &\equiv \sum_{i=0}^{n-1} t_i r^{i-n} \pmod{p} \quad \text{Dependency free} \end{aligned}$$

However, we want $\text{REDC}(T) < p$

OUR PROPOSED REDC(T) - CONTINUE

Since

$$T = \sum_{i=0}^{n-1} t_i r^i, t_i < r \text{ except } t_{n-1}$$

Then

$$REDC(T) \equiv R^{-1} \sum_{i=0}^{n-1} t_i r^i \quad R = r^n$$

$$\equiv \sum_{i=0}^{n-1} t_i r^{i-n}$$

$$\equiv r^{-2} \sum_{i=0}^{n-1} t_i r^{i-n+2} \pmod{p} < 3p$$

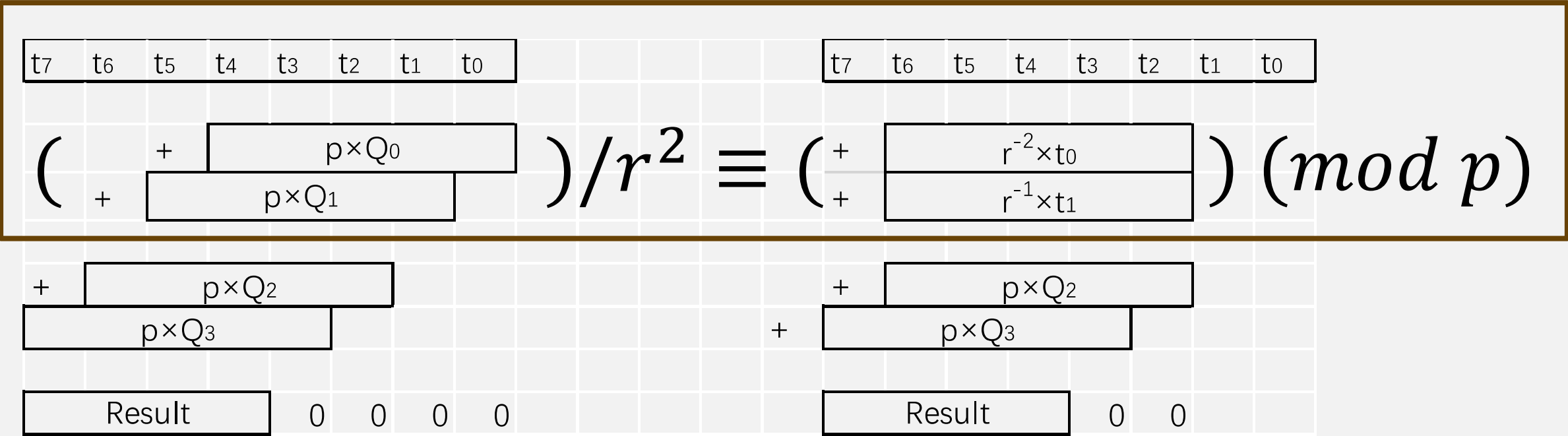
Can be reduced
to $[0, p)$ easily

$< npr$

OUR PROPOSED REDC(T) ILLUSTRATION

Naïve

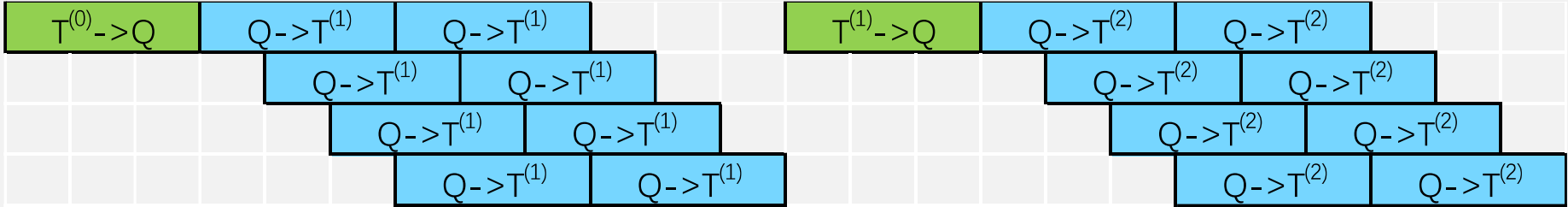
Proposed method



Last two iterations are the same

HOW IT WORKS

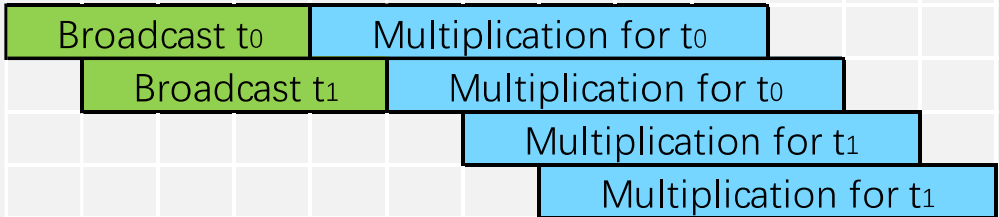
x64/A64



AVX-512/SVE
Naive



Proposal 2



Both steps have enough parallelism



REDUCTION FOR MONTGOMERY-FRIENDLY PRIME

MONTGOMERY
REDUCTION
SPECIAL
CASE[5]

Algorithm ExistingGenericRedc: Generic Montgomery reduction

Input: $T < pR$, where $p < 2^{\omega n}$, $r = 2^{\omega}$, $R = 2^{\omega n}$,
 r^{-1}, R^{-1} are positive integers such that
 $rr^{-1} \equiv 1 \pmod{p}$, $RR^{-1} \equiv 1 \pmod{p}$, and
 $p' \leftarrow \frac{rr^{-1}-1}{p}$ ← $p' = 1$ if $p \equiv -1 \pmod{r}$

Output: $REDC(T) = TR^{-1} \pmod{p}$

```

1  $T^{(0)} \leftarrow T$ 
2 for  $i = 1$  to  $n$  do
3    $Q \leftarrow T^{(i-1)}p' \pmod{r}$  ←  $Q = T^{(i-1)} \pmod{r}$ 
4    $T^{(i)} \leftarrow (T^{(i-1)} + Qp)/r$ 
5 end
6 if  $T^{(n)} > p$  then      Replace  $p$  with  $p+1$ 
7    $T^{(n)} \leftarrow T^{(n)} - p$ 
8 return  $T^{(n)}$ 

```

[5]. A. Faz-Hernández, J. López, E. Ochoa-Jiménez, and F. Rodríguez-Henríquez, “A faster software implementation of the supersingular isogeny Diffie-Hellman key exchange protocol,” *IEEE Transactions on Computers*, vol. 67, no. 11, pp. 1622–1636, 2017.

MONTGOMERY
REDUCTION
SPECIAL
CASE[5]

Algorithm ExistingSpecificRedc: Montgomery reduction with λ -Montgomery-friendly modulus [27]

Input: $p < 2^{\omega n}$, $r = 2^\omega$, $R = 2^{\omega n}$, $T < pR$, $1 < m \leq \lambda$ such that $p \bmod r^m \equiv -1$, $\lambda_0 \leftarrow \lfloor \omega n / m \rfloor$, $\lambda'_0 \leftarrow (\omega \cdot n) \bmod m$, and $M \leftarrow (p + 1) / 2^{\lambda \cdot \omega}$.

Output: $TR^{-1} \bmod p$

1 $T^{(0)} \leftarrow T$

2 **for** $i \leftarrow 1$ **to** λ_0 **do**

3 | $Q \leftarrow T^{(i-1)} \bmod 2^{m \cdot \omega}$

4 | $T^{(i)} \leftarrow \lfloor (T^{(i-1)} + 2^{\lambda \cdot \omega} Q \cdot M) / 2^{m \cdot \omega} \rfloor$

5 **end**

6 **if** $\lambda'_0 \neq 0$ **then**

7 | $Q \leftarrow T^{(\lambda_0)} \bmod 2^{\lambda'_0 \cdot \omega}$

8 | $T^{(\lambda_0+1)} \leftarrow \lfloor (T^{(\lambda_0)} + 2^{\lambda \cdot \omega} Q \cdot M) / 2^{\lambda'_0 \cdot \omega} \rfloor$

9 **end**

10 **if** $T^{(\lambda_0+1)} \geq p$ **then**

11 | $T^{(\lambda_0+1)} \leftarrow T^{(\lambda_0+1)} - p$

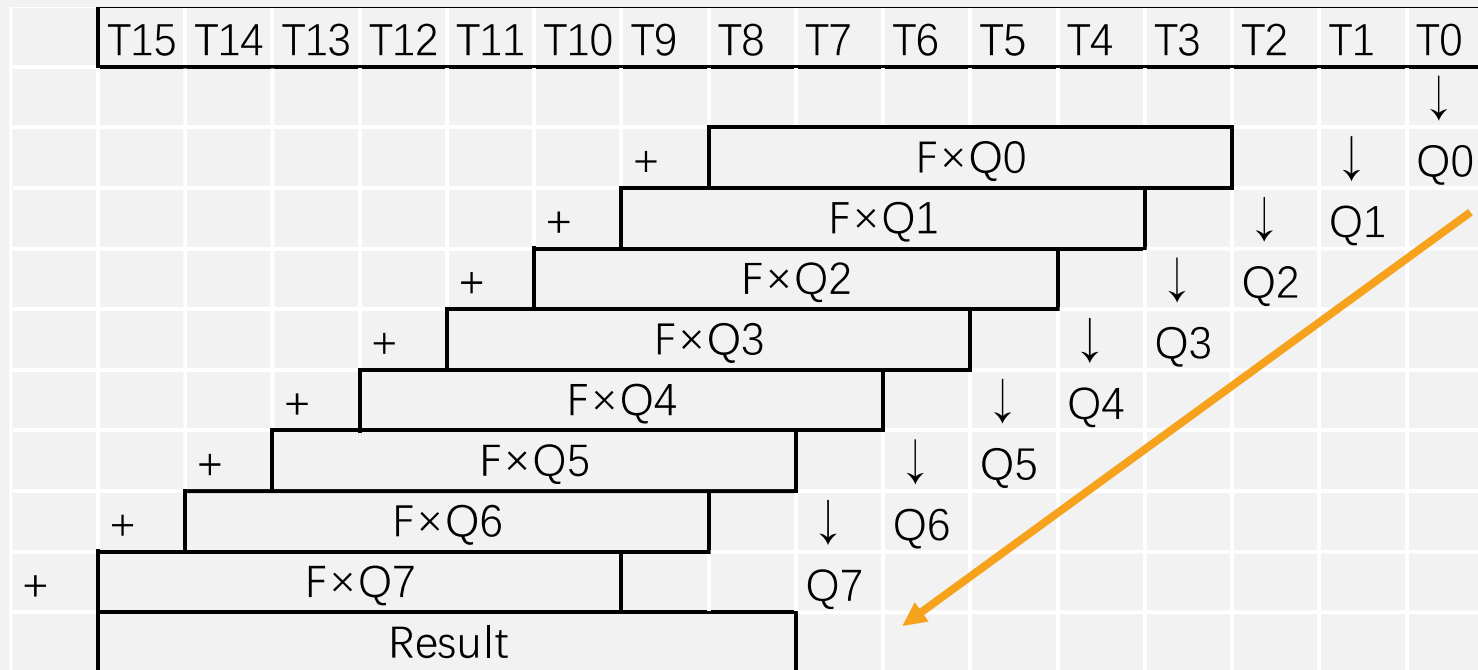
12 **end**

13 **return** $T^{(\lambda_0+1)}$

Q by M large multiplication, we want

To accelerate with Karatsuba method, not easy

MONTGOMERY-FRIENDLY REDUCTION – CALCULATION FLOW



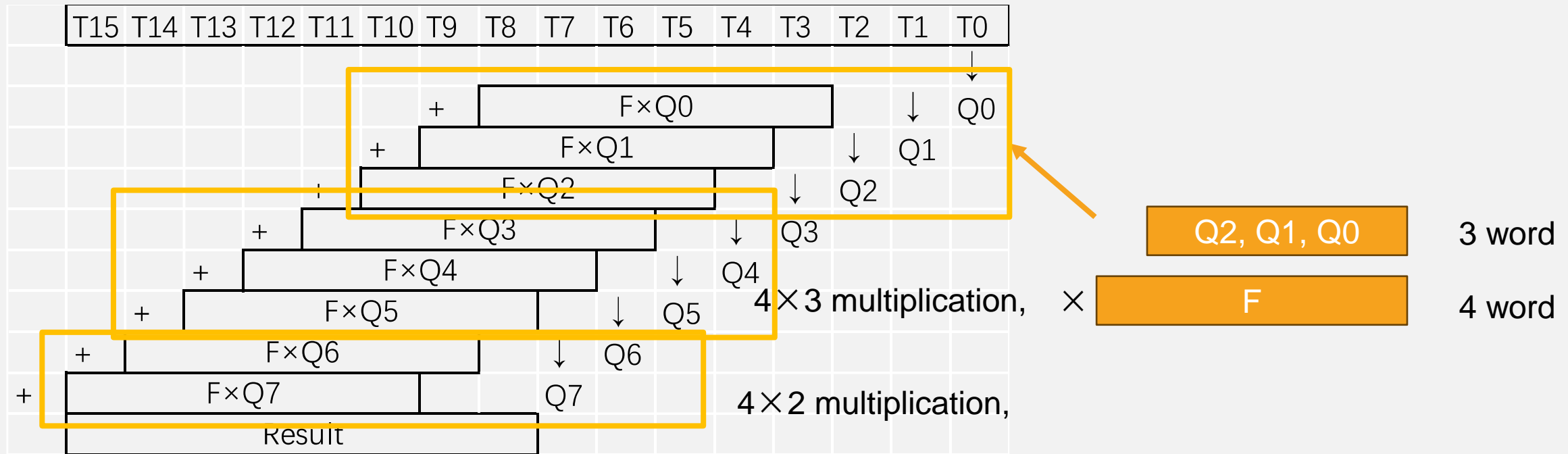
E.g. $p_{503} = 2^{250} \times 3^{159} - 1, r = 2^{64},$
 $F = 3^{159}$ (4 word)

Special case:

$$p = 2^l F - 1$$

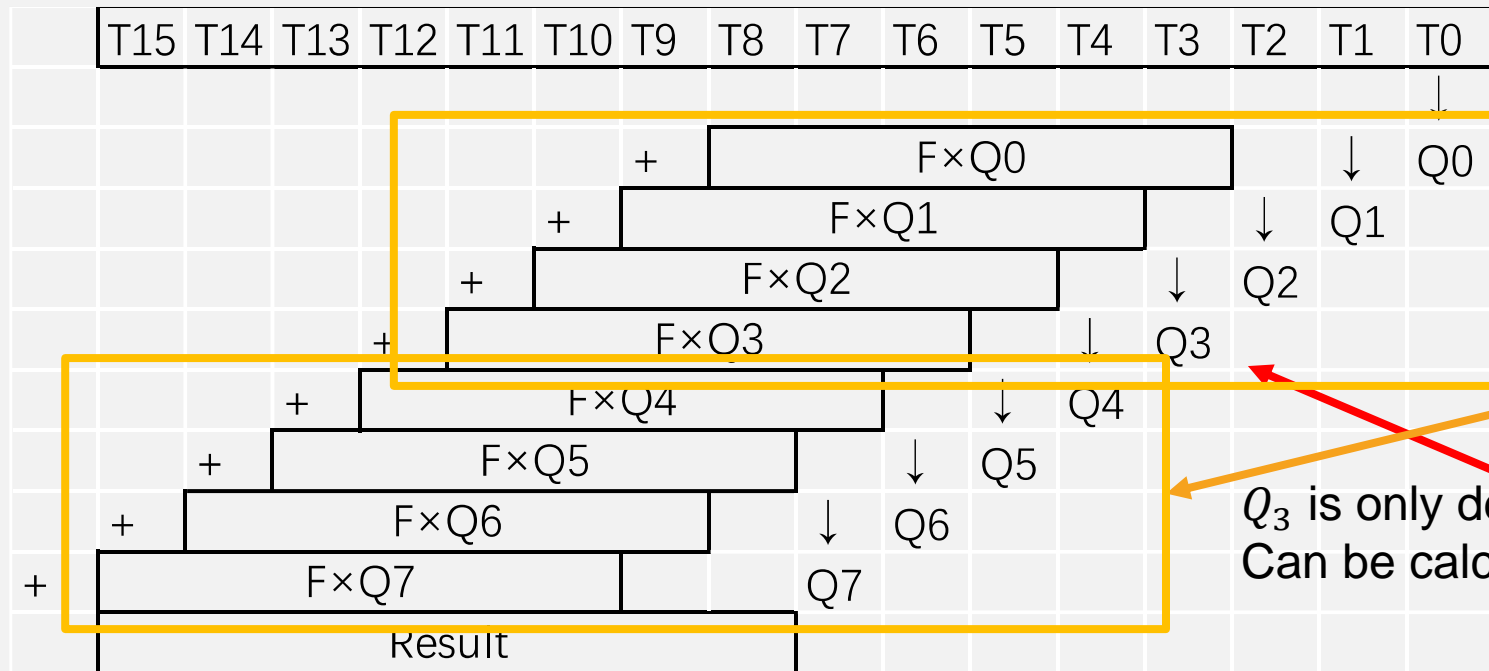
$$F \approx 2^l$$

MONTGOMERY-FRIENDLY REDUCTION – CALCULATION FLOW



E.g. $p_{503} = 2^{250} \times 3^{159} - 1, r = 2^{64},$
 $F = 3^{159}$

MONTGOMERY-FRIENDLY REDUCTION PROPOSED METHOD



4 × 4 multiplication,
Can use Karatsuba to accelerate.

Q_3 is only dependent on Q_0
Can be calculated ahead with one multiplication.

$$Q_3 = T_3 + ((F \bmod r)Q_0 \bmod r)$$

E.g. $p_{503} = 2^{250} \times 3^{159} - 1, r = 2^{64},$
 $F = 3^{159}$



RESULTS

Using 511-bit general prime



**CTIDH-511 WITH PROPOSED METHOD
ON SVE**

	ARM64[6] Runtime (cycles)	SVE Runtime (cycles)	Speedup
Addition	16.07	13.72 Proposal 1	1.17x
Montgomery Multiplication	406.98	258.96 Proposal 2	1.57x
CTIDH[7] Action	316,308,640	242,948,411 Proposal 1+2	1.30x

Benchmarked with A64FX@2.20GHz on Wisteria BDEC/01 (Odyssey) at U-Tokyo

[6]. Jalali, A. et al. (2019). Towards Optimized and Constant-Time CSIDH on Embedded Devices. In: Polian, I., Stöttinger, M. (eds) Constructive Side-Channel Analysis and Secure Design. COSADE 2019. Lecture Notes in Computer Science, vol 11421. Springer, Cham.

[7]. Banegas, Gustavo, et al. (2021). CTIDH: faster constant-time CSIDH. *IACR Transactions on Cryptographic Hardware and Embedded Systems*

Using 511-bit general prime

CSIDH-511 WITH PROPOSED METHOD ON AVX-512

Calls multiplication directly

Specially designed
squaring algorithm

Runtime	x64[8] (cycles)	AVX-512 by [9] (cycles)	AVX-512 Ours with Proposal 2
Montgomery Squaring 2-packed	262 1.00x	142 1.85x	104 2.51x
Montgomery Multiplication 2-packed	258 1.00x	145 1.78x	131 1.97x
CSIDH[10] Action	132,051,574 1.00x	83,099,556 1.59x	74,491,118 1.77x

Benchmarked with i7-1165G7@2701MHz

[8]. D. Cervantes-Vázquez, et. al., “Stronger and faster side-channel protections for CSIDH,” in LATINCRYPT 2019

[9]. H. Cheng, et.al, “Batching CSIDH group actions using AVX-512,” IACR TCHES, vol. 2021

[10]. W. Castryck, et. al., “CSIDH: An efficient post-quantum commutative group action,” in ASIACRYPT 2018

512-BIT SIMD ADD-WITH-CARRY COMPARISON

	8x64-bit->8x8-bit	8x64-bit->8x1-bit
	SVE	AVX-512
Kogge-Stone Vector Addition (cycles)	55.28	4.16 [11]
Proposal 1 (cycles)	42.90 1.29x	6.41 0.65x

Mask->GPR available

Efficient Mask -> SVE or GPR not available,

[11] A. Yee, "Integer addition and carryout,"
<http://www.numberworld.org/y-cruncher/internals/addition.html#ksadd>, 2019.

Using 503-bit special prime

SIKEP503 WITH PROPOSED REDUCTION

Highly optimized ARM64 assembly Implementation.
Worth to compared although SIKE has broken

	SIDHv3.5[12] Time (cycles)	New Reduction Proposal 3 Time (cycles)	Speedup
Reduction	196.84	156.48	1.26x
Keygen	36,749,182	35,204,915	1.04x
Encapsulation	60,642,713	56,449,034	1.07x
Decapsulation	65,017,001	60,901,455	1.07x

Benchmarked with A64FX@2.20GHz on Wisteria BDEC/01 (Odyssey)

[12]. Microsoft, "PQCrypto-SIDH," 2020. [Online]. Available: <https://github.com/Microsoft/PQCrypto-SIDH>

CONCLUSION

- A parallel algorithm for large integer addition using SIMD is proposed by **converting** 512-bit addition to 64-bit to calculate carries
- A parallel algorithm for Montgomery reduction using SIMD is proposed by **breaking dependency chains**
- An optimized reduction algorithm for SIKE-like primes is proposed by **using Karatsuba method**
- 30% speedup for CTIDH on SVE
- 10% speedup for CSIDH on AVX-512
- 27% speedup for the Montgomery reduction of SIKE

Q&A