A Fast Large-Integer Extended GCD Algorithm and Hardware Design for Verifiable Delay Functions and Modular Inversion

Kavya Sreedhar, Mark Horowitz, Christopher Torng
Stanford University
skavya@stanford.edu

September 19, 2022

Extended GCD

Computes Bézout coefficients $\boldsymbol{b_a, b_b}$ satisfying Bézout's Identity

$$\boldsymbol{b_a, b_b} \ : \ \boldsymbol{b_a} * a_0 + \boldsymbol{b_b} * b_0 = \gcd(a_0, b_0)$$

Extended GCD is widely used in cryptography

Computes Bézout coefficients $\boldsymbol{b_a, b_b}$ satisfying Bézout's Identity

$$\boldsymbol{b_a, b_b} \ : \ \boldsymbol{b_a} * a_0 + \boldsymbol{b_b} * b_0 = \gcd(a_0, b_0)$$

Modular Multiplicative Inverse

RSA

Elliptic Curve Cryptography

ElGamal Encryption

⋮

There is an increasing need for faster XGCD

1. Modular Inversion for Curve25519 [Ber06]
   - Constant-time XGCD faster than Fermat's Little Theorem [BY19]

$$x^{-1} = x^{p-2} \ (mod \ p)$$

There is an increasing need for faster XGCD

1. Modular Inversion for Curve25519 [Ber06]
   - Constant-time XGCD faster than Fermat's Little Theorem [BY19]

$$x^{-1} = x^{p-2} \ (mod\ p)$$

2. Squaring binary quadratic forms over class groups [Wes19] as a VDF [BBBF18]
   - XGCD is the bottleneck

$f(x) = x^{2^T}$ in a class group

There is an increasing need for faster XGCD

1. Modular Inversion for Curve25519 [Ber06]
   - Constant-time XGCD faster than Fermat's Little Theorem [BY19]

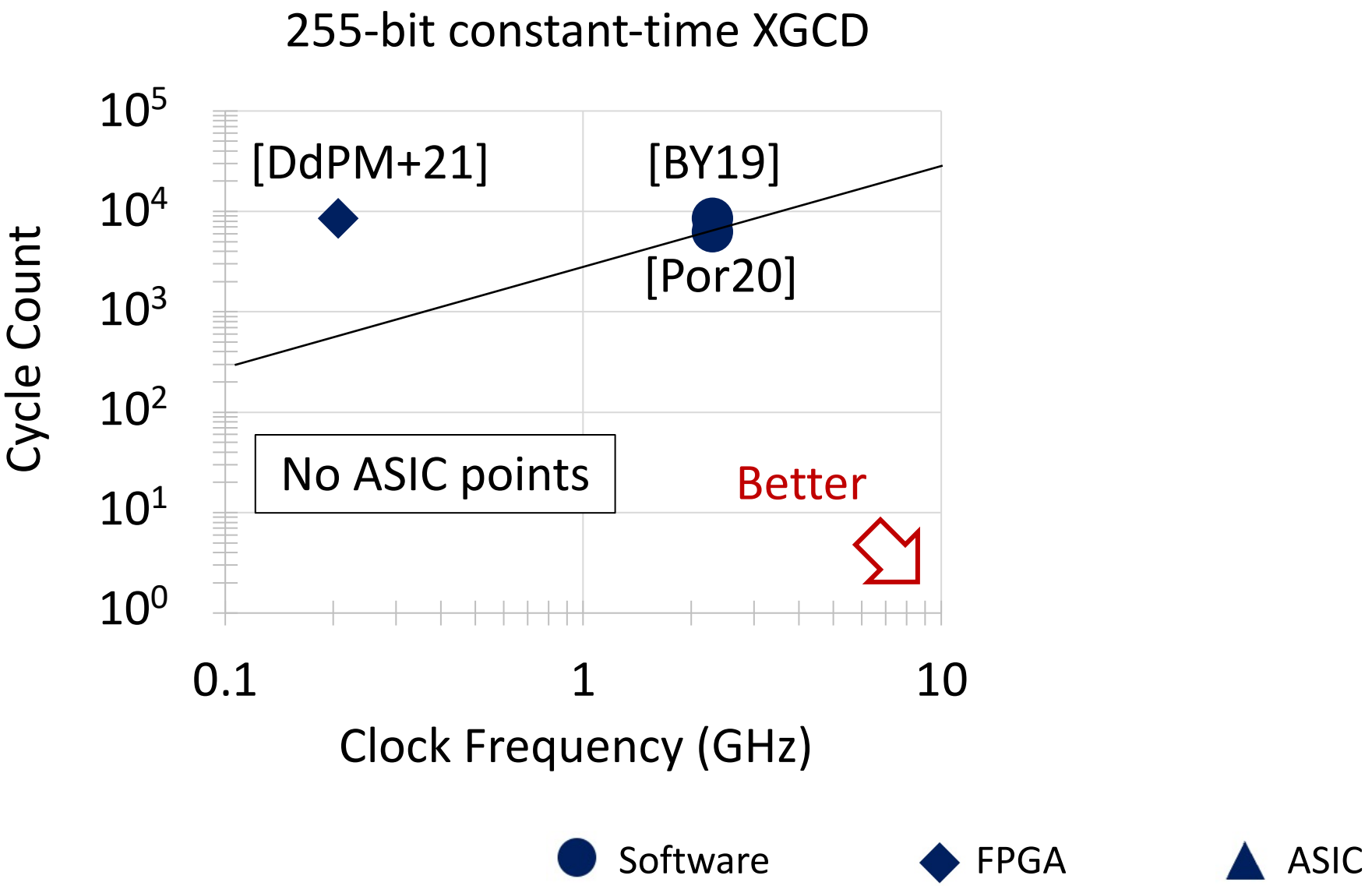
255-bits, constant-time

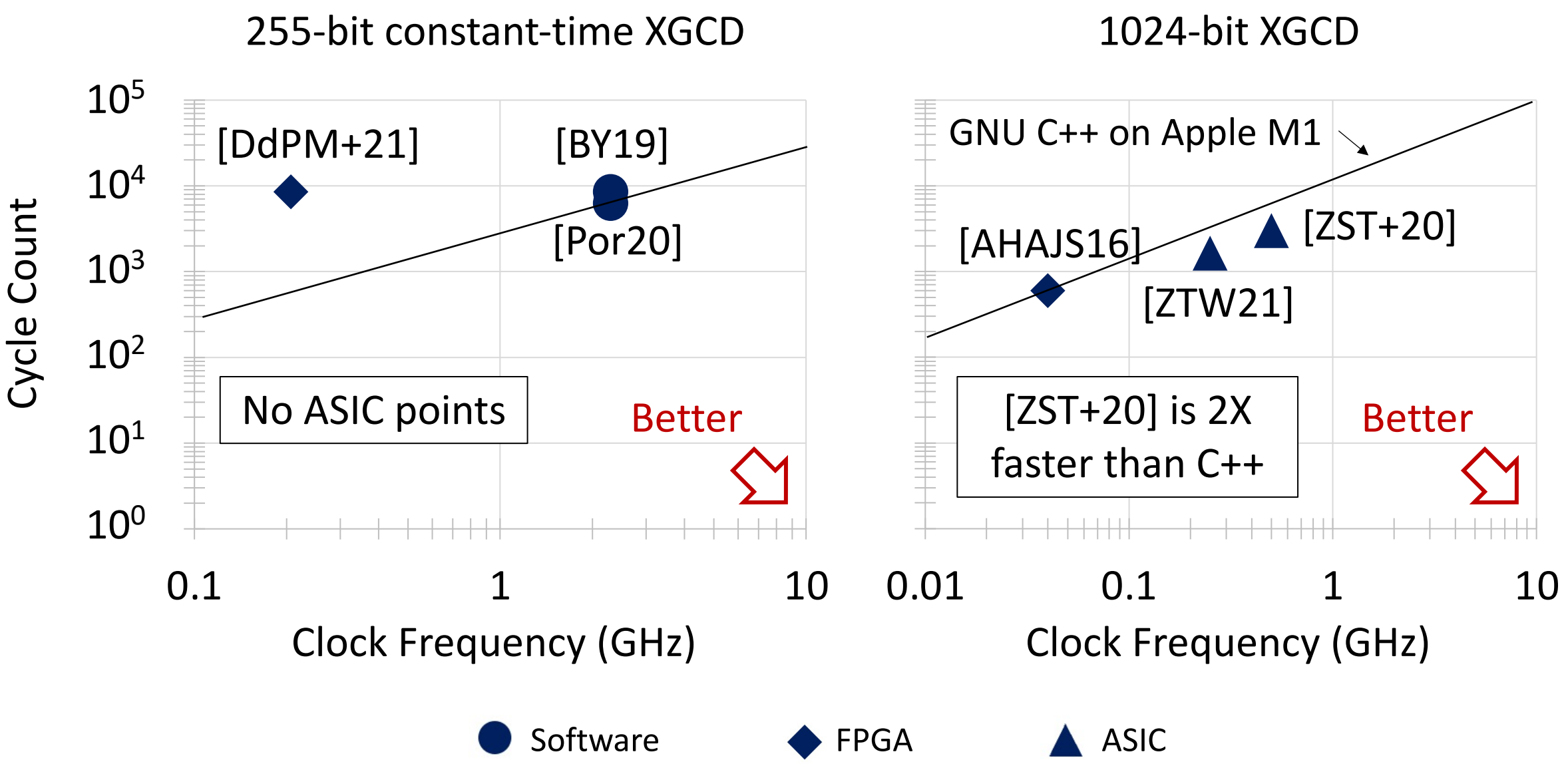
$$x^{-1} = x^{p-2} \ (mod\ p)$$

2. Squaring binary quadratic forms over class groups [Wes19] as a VDF [BBBF18]
   - XGCD is the bottleneck

$f(x) = x^{2^T}$ in a class group

1024-bits, not constant-time


255-bit constant-time XGCD
Cycle Count
10^5
10^4
10^3
10^2
10^1
10^0
[DdPM+21]
[BY19]
[Por20]
No ASIC points
Better
0.1
1
10
Clock Frequency (GHz)
Software
FPGA
ASIC

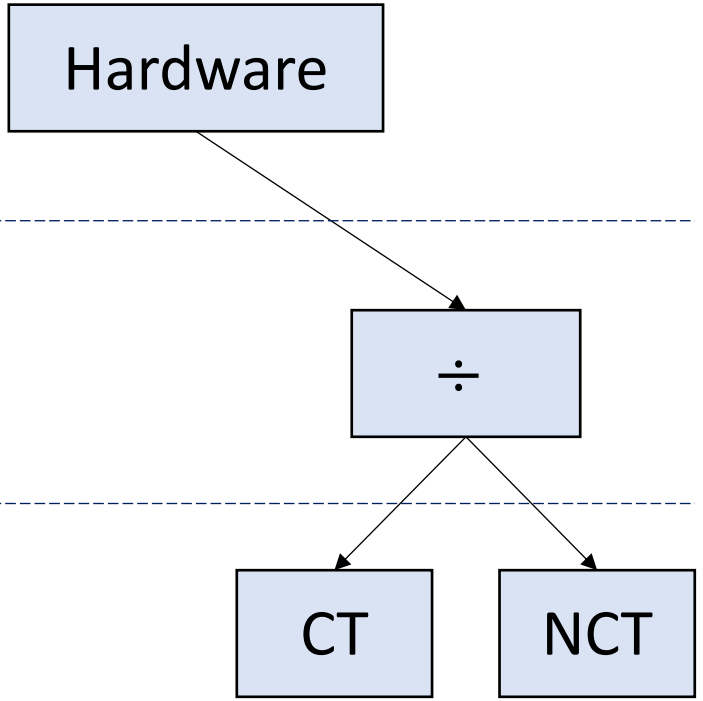

255-bit constant-time XGCD
1024-bit XGCD
Cycle Count
10^5
10^4
10^3
10^2
10^1
10^0
[DdPM+21]
[BY19]
[Por20]
No ASIC points
Better
GNU C++ on Apple M1
[AHAJS16]
[ZST+20]
[ZTW21]
[ZST+20] is 2X faster than C++
Better
0.1
1
10
0.01
0.1
1
10
Clock Frequency (GHz)
Clock Frequency (GHz)
Software
FPGA
ASIC

Current view of XGCD design space

Target Platform

Algorithm

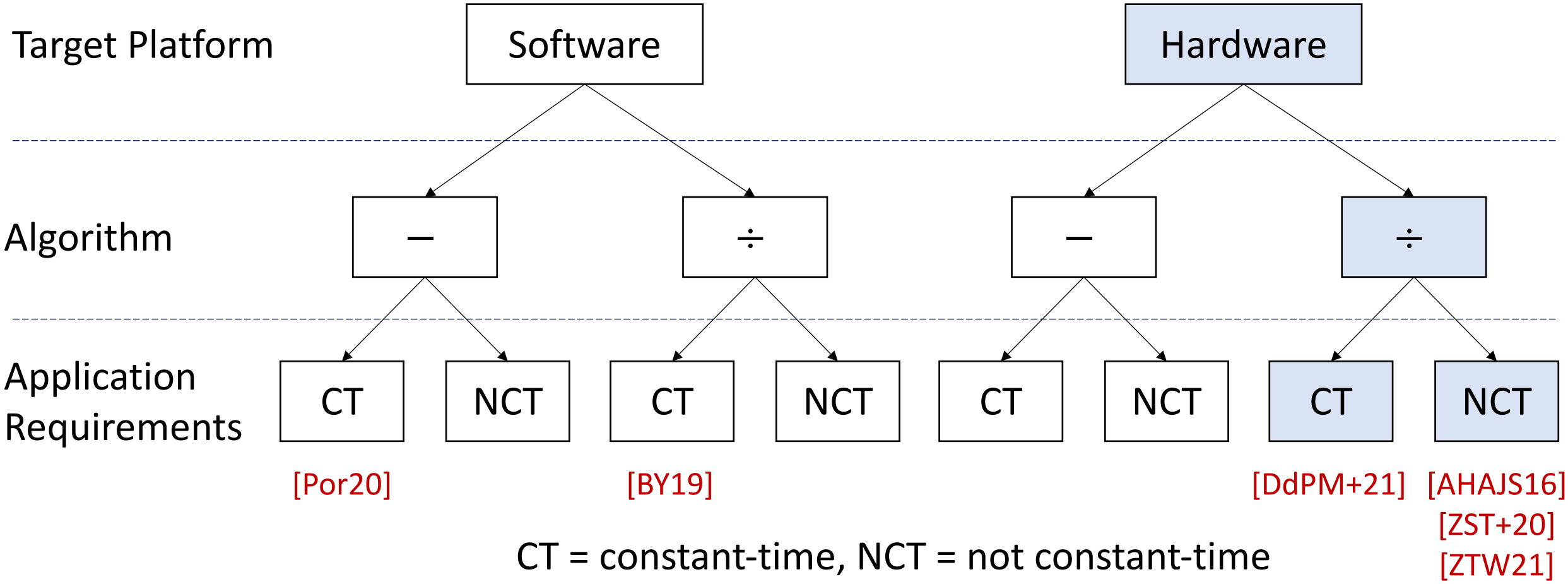
Application Requirements



[DdPM+21] [AHAJS16] [ZST+20] [ZTW21]

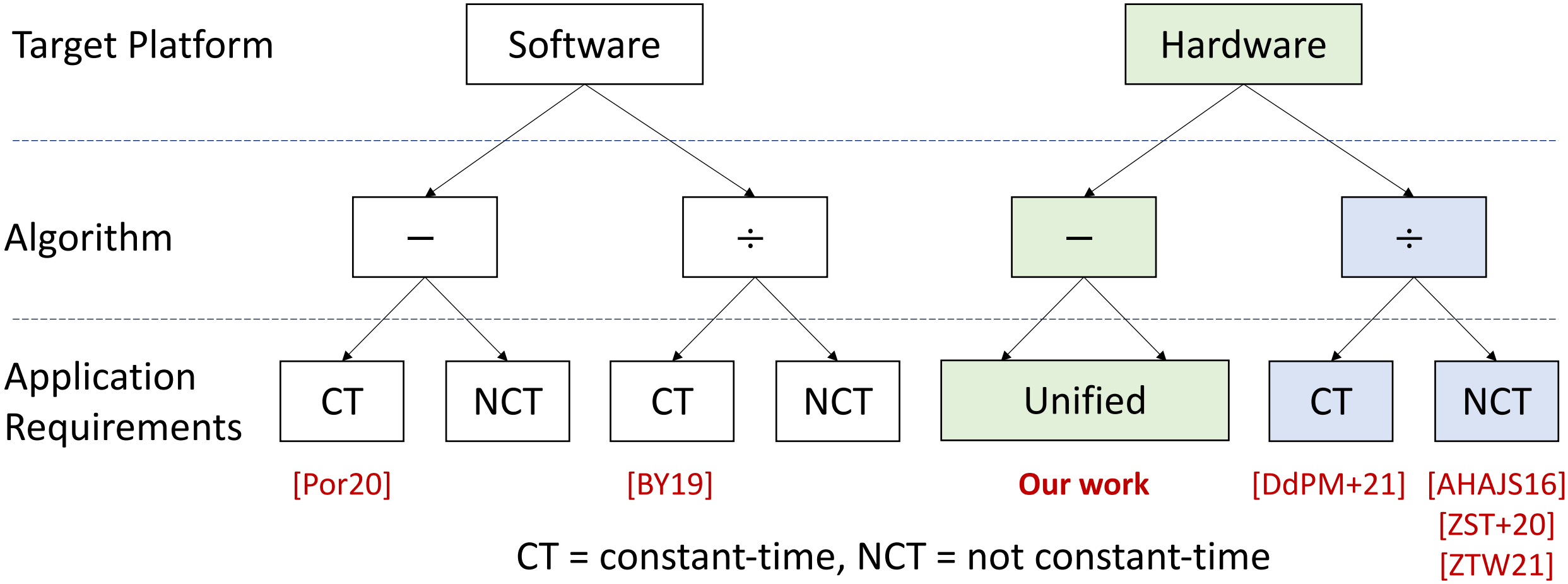
CT = constant-time, NCT = not constant-time

We explore the broader design space



CT = constant-time, NCT = not constant-time

We explore the broader design space



CT = constant-time, NCT = not constant-time

Hardware allows for short iteration times

Target Platform	**Software**	**VS**	**Hardware**
Number of Iterations	From algorithm		From algorithm
Constrained to ISA	Yes		No

Execution time = number of iterations $*$ iteration time

The control over iteration time in hardware opens the opportunity to accelerate simpler algorithms that require more iterations.

GCD Algorithms Comparison

Algorithm	Stein [Ste67] $-$	**VS**	Euclid (300 BC) $\div$
GCD-preserving Transformation	$\gcd(a,b) = \gcd(a-b,b)$		$\gcd(a,b) = \gcd(a \bmod b, b)$

GCD Algorithms Comparison

* Two-bit PM [YZ86]

Algorithm	Stein [Ste67] $-$	**VS**	Euclid (300 BC) $\div$
GCD-preserving Transformation	$\gcd(a,b) = \gcd(a-b,b)$		$\gcd(a,b) = \gcd(a \bmod b, b)$
Worst-Case Iterations	387 * (1X difference for 255 bits) 1548 * (1X difference for 1024 bits)		384 1542

GCD Algorithms Comparison

* Two-bit PM [YZ86]

Algorithm	Stein [Ste67] $-$	**VS**	Euclid (300 BC) $\div$
GCD-preserving Transformation	$\gcd(a,b) = \gcd(a-b,b)$		$\gcd(a,b) = \gcd(a \bmod b, b)$
Worst-Case Iterations	387 *	1X difference for 255 bits	384
	1548 *	1X difference for 1024 bits	1542
Average Iterations	300 *	1.6X difference for 255 bits	189
	1195 *	2X difference for 1024 bits	598

From GCD to Extended GCD (XGCD)

- Compute Bézout coefficients satisfying Bézout Identity

$$\boldsymbol{b_a}, \boldsymbol{b_b} \ : \ \boldsymbol{b_a} * a_0 + \boldsymbol{b_b} * b_0 = \gcd(a_0, b_0)$$

- Maintain these relations each cycle, where $\gcd(a_0, b_0) = \gcd(a, b)$

$$u * a_0 + m * b_0 = a$$
$$y * a_0 + n * b_0 = b$$

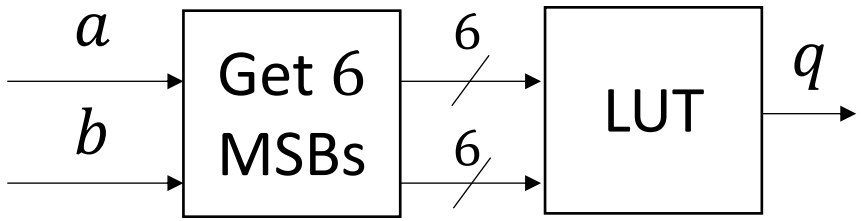
Two-bit PM Critical Path

GCD update: $a = \frac{a-b}{4}$

XGCD update: $m = \frac{m-n-a_m}{4}$

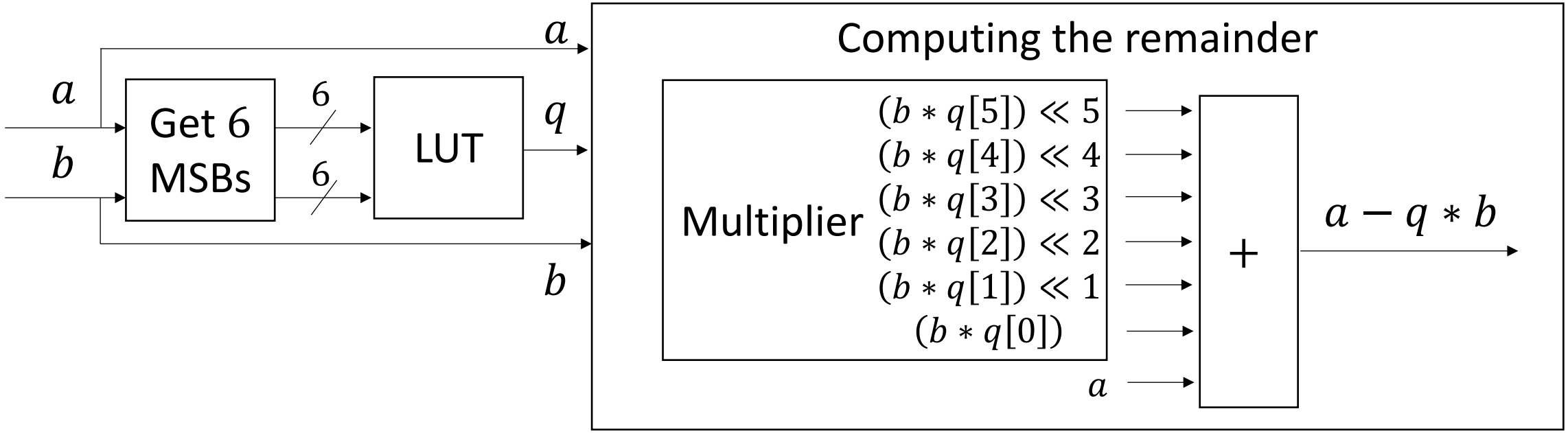
Euclid critical path

Compute $q \leq \lfloor \frac{a}{b} \rfloor$ ⟶ Compute $q * b$ ⟶ Compute a $- q * b$



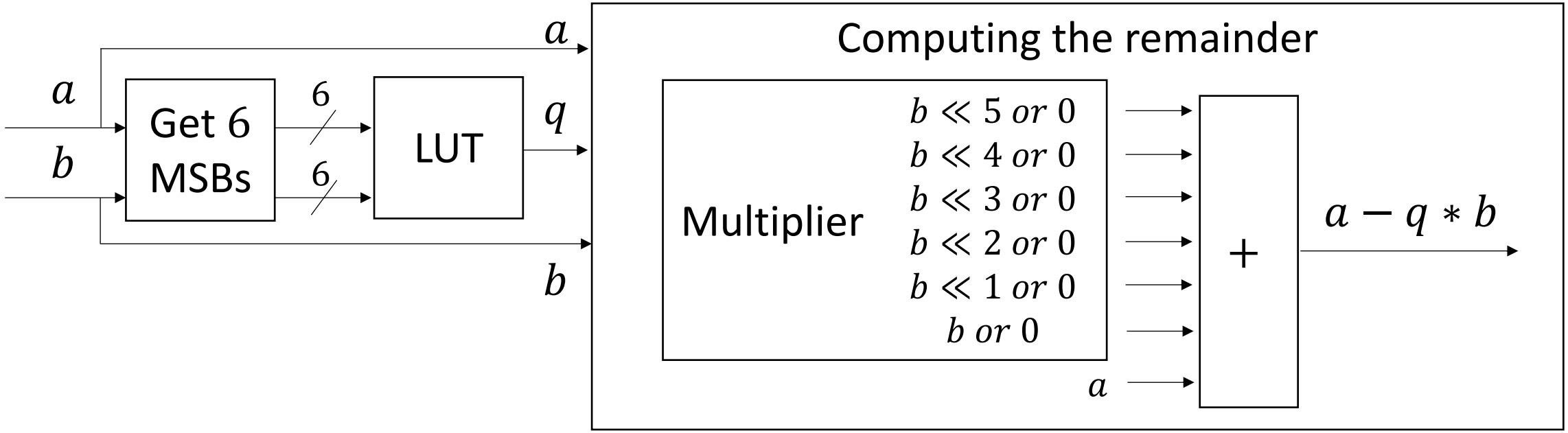
Euclid critical path

Compute $q \leq \lfloor \frac{a}{b} \rfloor$ ⟶ Compute $q * b$ ⟶ Compute a $- q * b$



Euclid critical path

Compute $q \leq \lfloor \frac{a}{b} \rfloor$ ⟶ Compute $q * b$ ⟶ Compute a $- q * b$



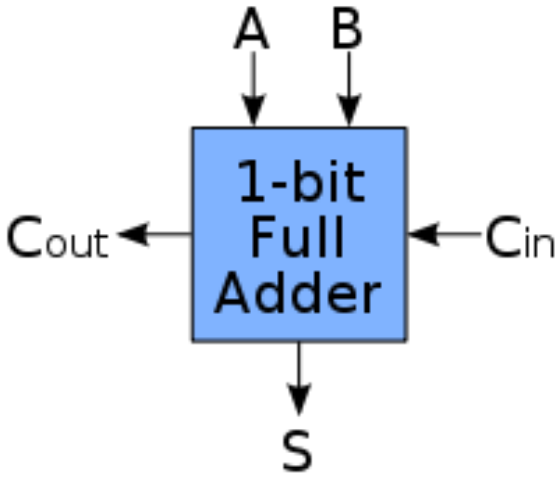
Critical paths primarily require additions

- The fastest adder is a carry-save adder (CSA)
  - Eliminates carry propagation, requiring $O(1)$ delay
  - Stores numbers in CSA form or redundant binary form

```
  1 1 1 0
    1 1 0 1 (a)
+   1 1 1 0 (b)
    0 0 1 0 (c)
 ---------------
  1 1 1 0 1
```

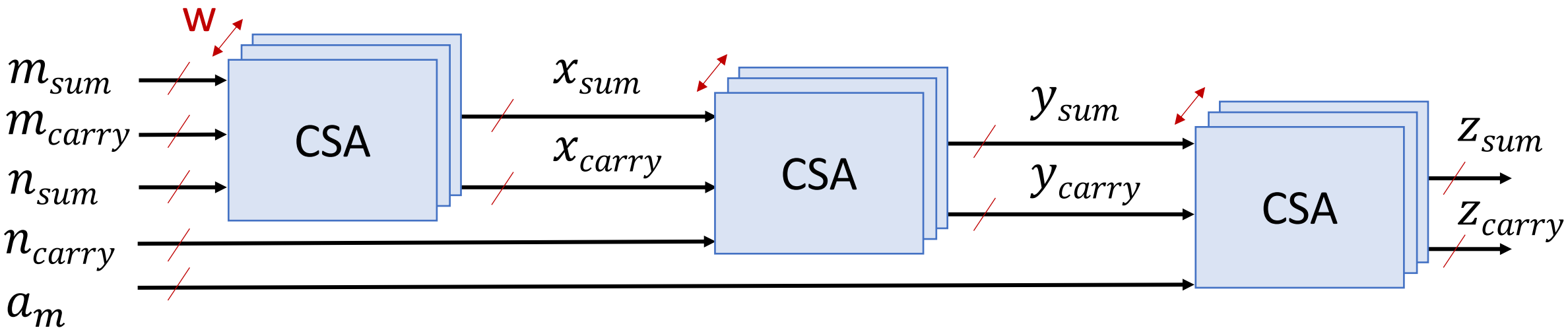
→

```
    1 1 0 1 (a)
+   1 1 1 0 (b)
    0 0 1 0 (c)
 ------------------
+   0 0 0 1 sum
  1 1 1 0 0 carry
 ------------------
  1 1 1 0 1
```



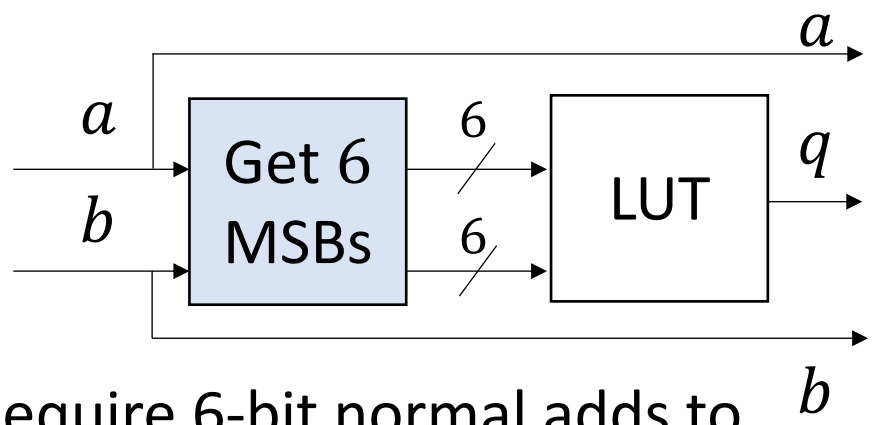
Two-bit PM critical path: 3 CSA delays

$$\frac{m - n - a_m}{4}$$



Euclid critical path

Compute $q \leq \lfloor \frac{a}{b} \rfloor$ ⟶ Compute $q * b$ ⟶ Compute a $- q * b$

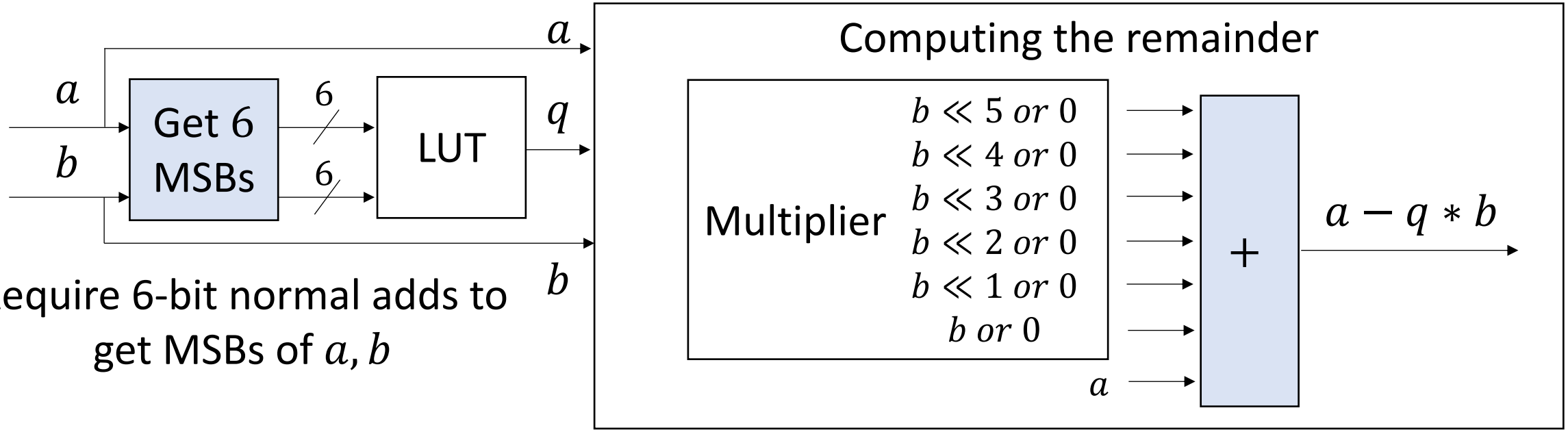


Require 6-bit normal adds to get MSBs of a, b

$\lfloor \log_2(6) \rfloor + 1 = 3$ CSA delays

Euclid critical path: at least 9 CSA delays

Compute $q \leq \lfloor \frac{a}{b} \rfloor$ ⟶ Compute $q * b$ ⟶ Compute a $- q * b$



Require 6-bit normal adds to get MSBs of a, b

$\lfloor \log_2(6) \rfloor + 1 = 3$ CSA delays

Add 14 values with CSAs $\approx \lfloor \log_{3/2}(14) \rfloor = 6$ CSA delays

Two-bit PM is a faster starting point

- Two-bit PM critical path is at least 3X shorter than Euclid's
- Two-bit PM iteration counts are at most 2X higher than Euclid's

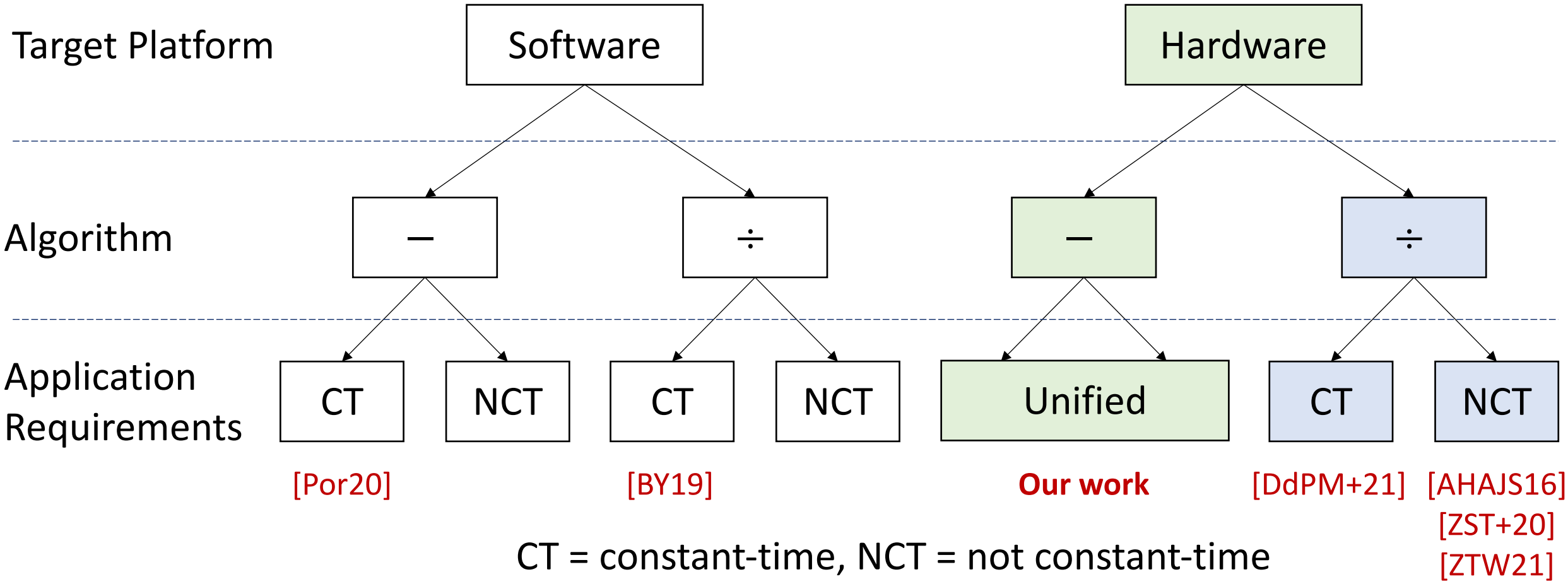
Two-bit PM with carry-save adders is the more promising starting point for hardware in the average and the worst-case.

Our unified design with constant-time config

Application Requirements	**CT**	**VS**	**NCT**
Approach	Pad to worst-case cycle count		Reduce inputs until GCD
Termination Condition	Cycle count equal to worst case		$a == 0$ or $b == 0$

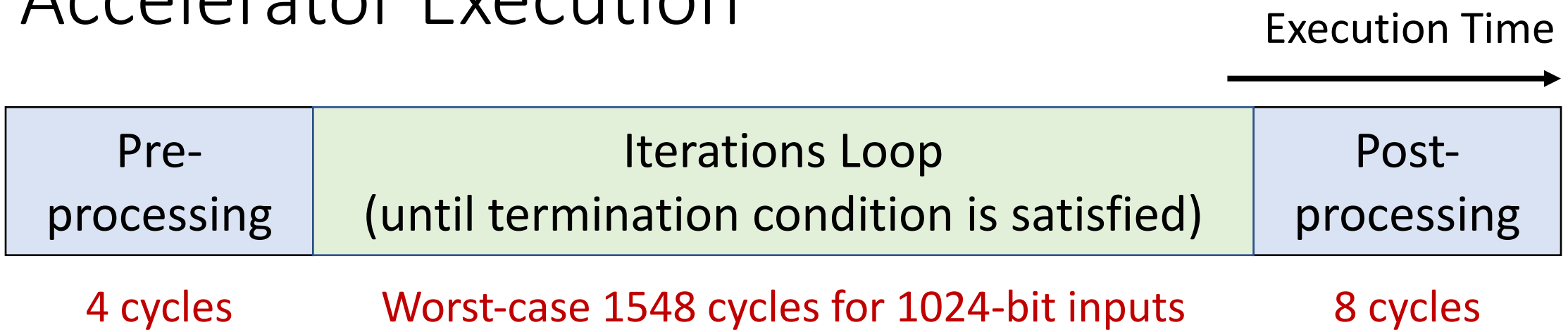
Note that since a, b are in CSA form, we do not know when they become 0

We focus on the optimal design space

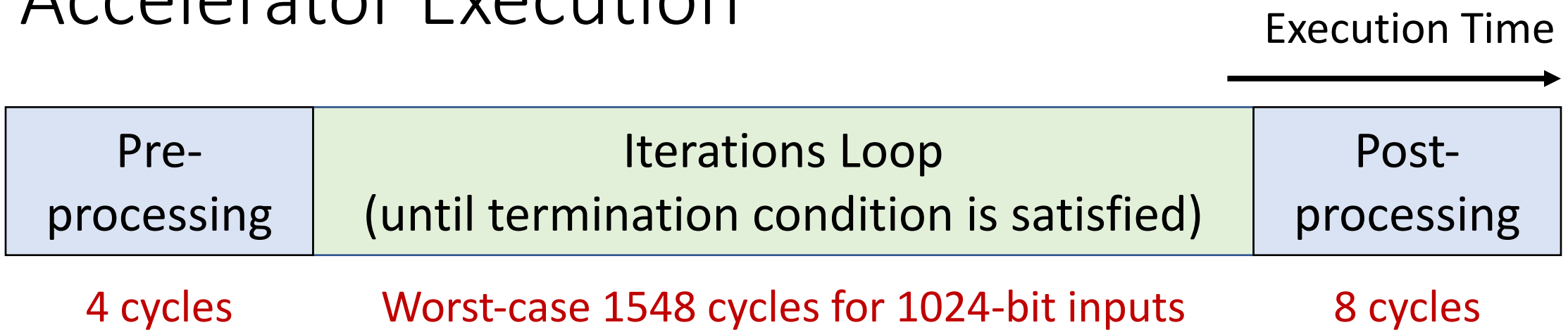


CT = constant-time, NCT = not constant-time

Accelerator Execution

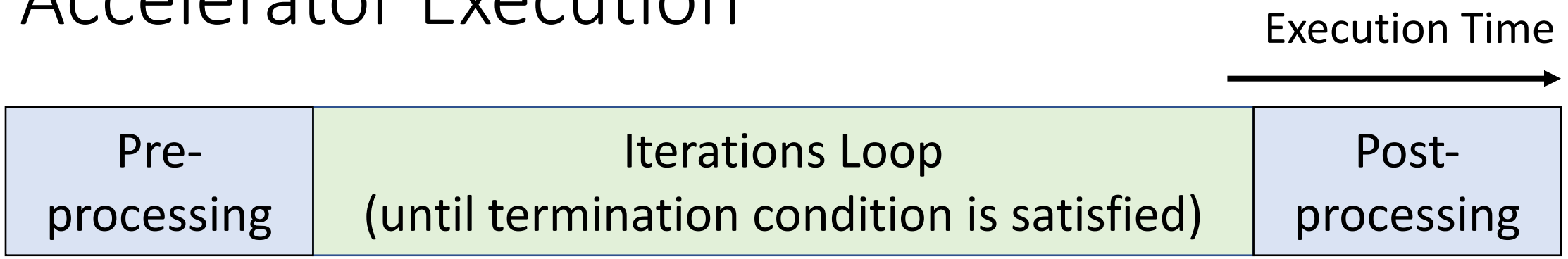


Accelerator Execution



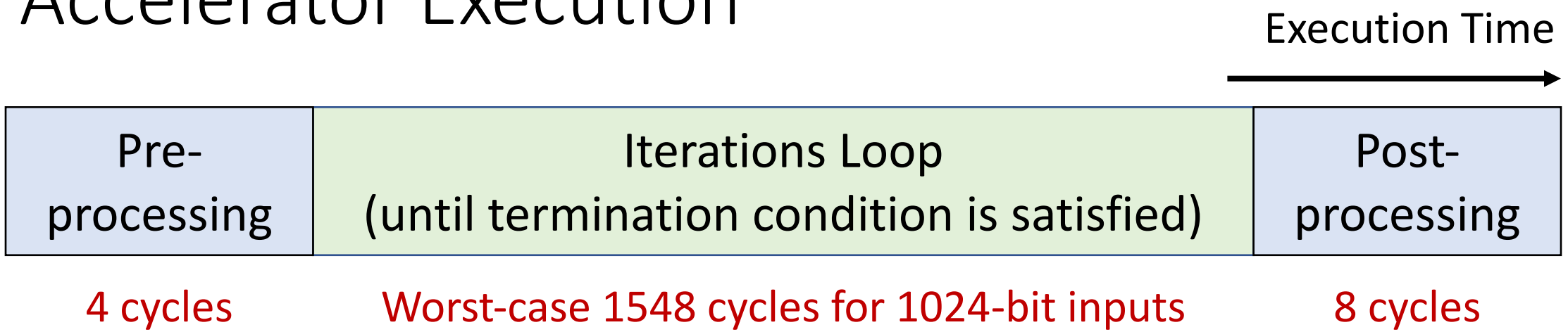
- Preserve results when shifting in CSA form

Accelerator Execution



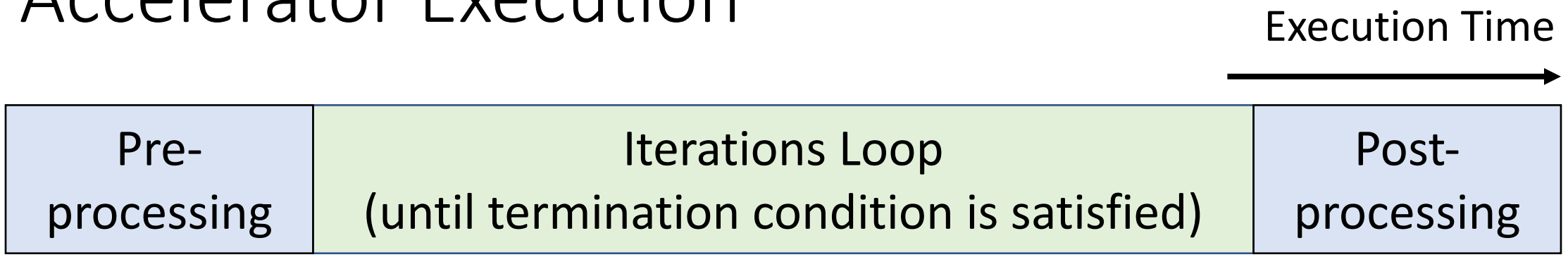
- Preserve results when shifting in CSA form
- Allocate multiple cycles for processing steps

Accelerator Execution



- Preserve results when shifting in CSA form
- Allocate multiple cycles for processing steps
- Subsample a, b for termination condition

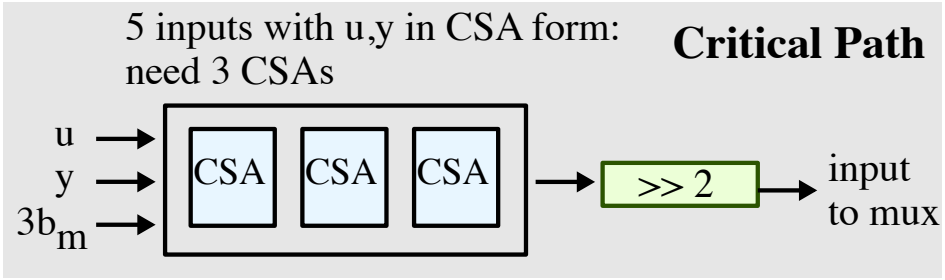
Accelerator Execution



- Preserve results when shifting in CSA form
- Allocate multiple cycles for processing steps
- Subsample a, b for termination condition
- Minimize control overhead

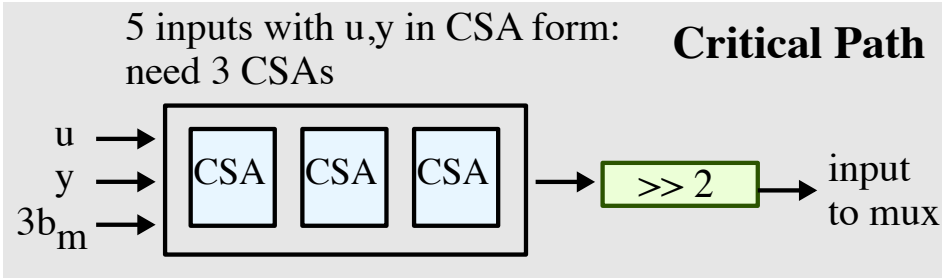
Critical Path for ASIC in 16nm

	255-bit XGCD	1024-bit XGCD
DFF clk to Q	45	40
Inverter	7	0
CSA	18	39
CSA	31	39
Buffer	13	0
CSA	30	34
Shift in CSA form	15	18
Late select muxes	18	18
Precomputing control	27	22
Setup Time	2	5
Total	204	215



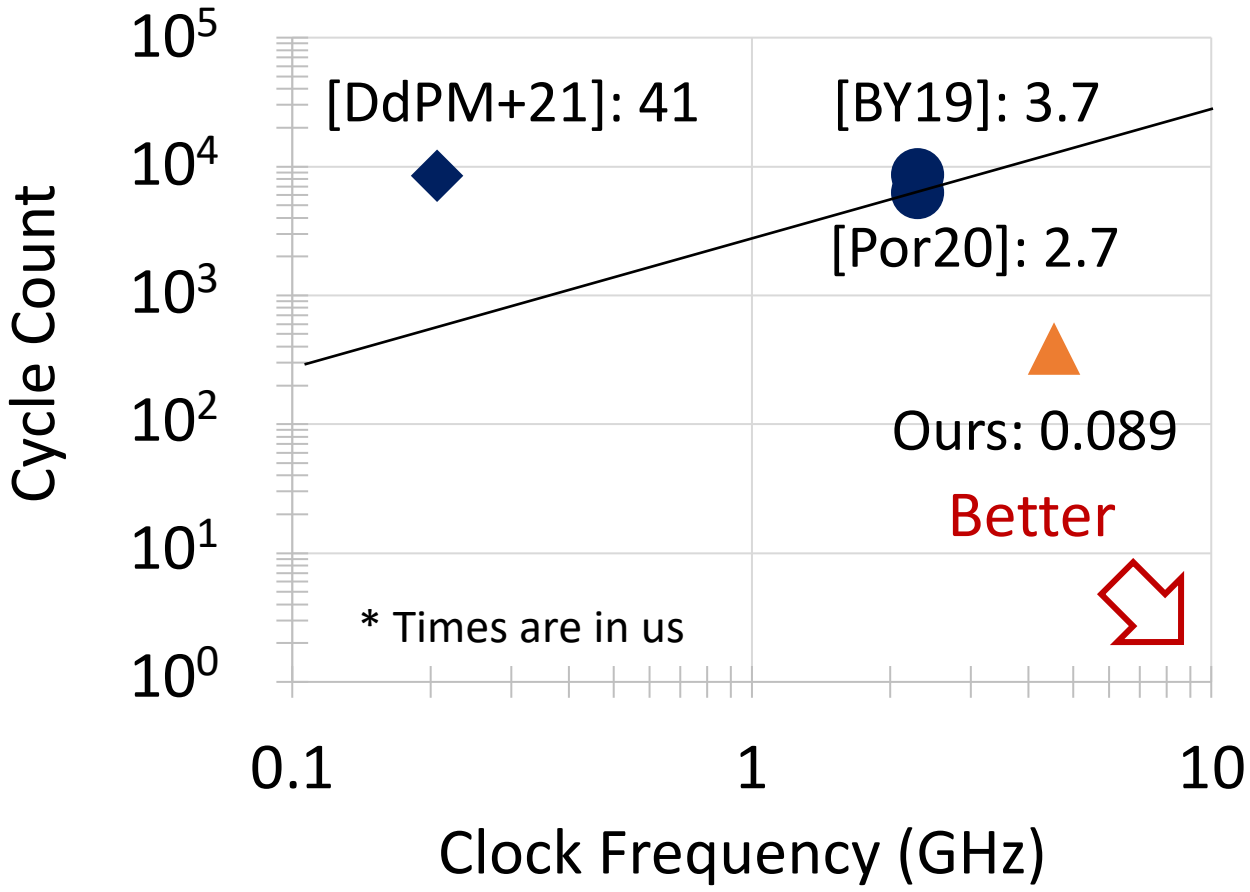
Critical Path for ASIC in 16nm

	255-bit XGCD	1024-bit XGCD
DFF clk to Q	45	40
Inverter	7	0
CSA	18	39
CSA	31	39
Buffer	13	0
CSA	30	34
Shift in CSA form	15	18
Late select muxes	18	18
Precomputing control	27	22
Setup Time	2	5
Clock Skew	16	41
Total	220	257



These are post-layout numbers for a fabrication-ready design

255-bit Constant-time XGCD Comparison

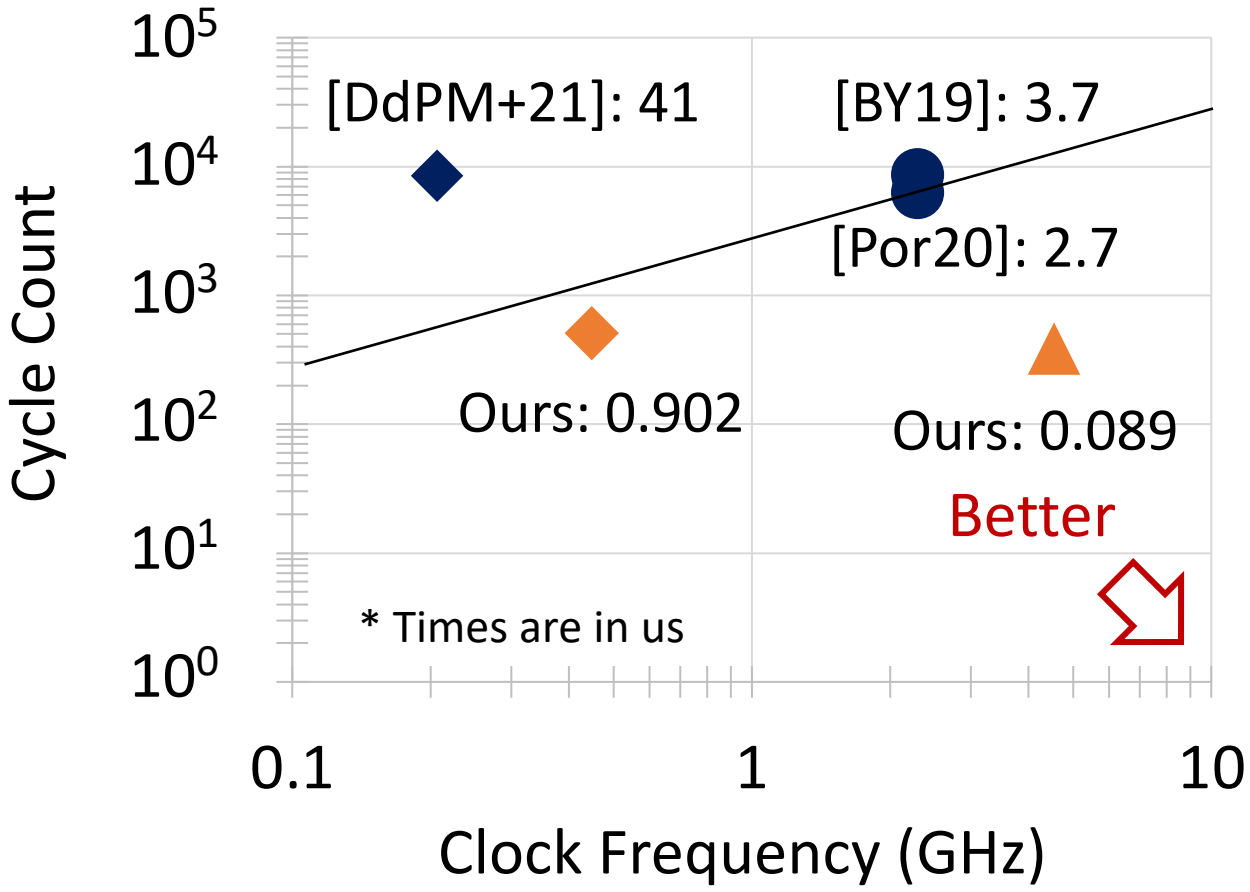


Our ASIC

- 31X faster than [Por20]
- First for constant-time 255-bit XGCD

Software FPGA ASIC

255-bit Constant-time XGCD Comparison



<u>Our ASIC</u>

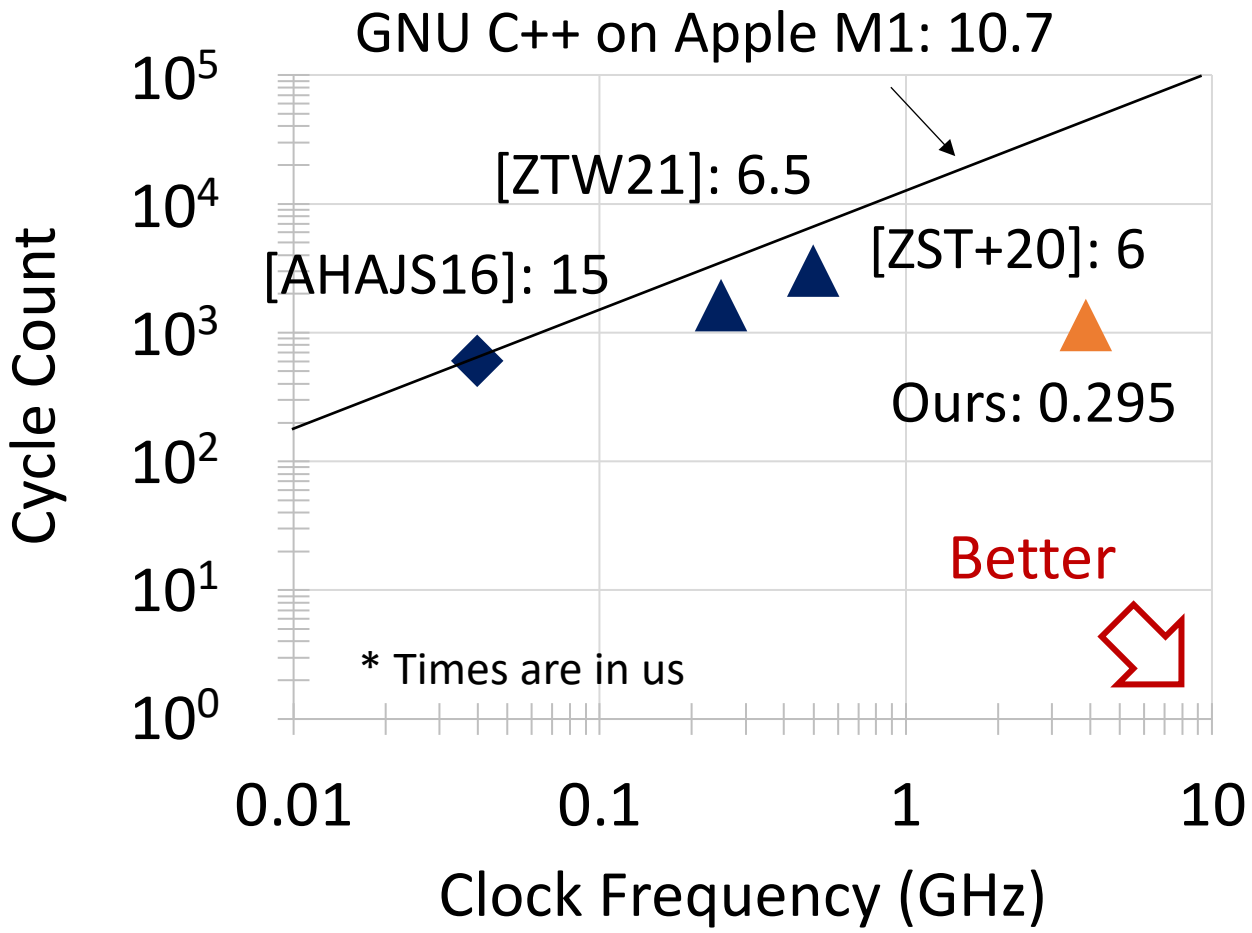
- 31X faster than [Por20]
- First for constant-time 255-bit XGCD

<u>Direct FPGA Comparison</u>

Our design is 45X faster

Software FPGA ASIC

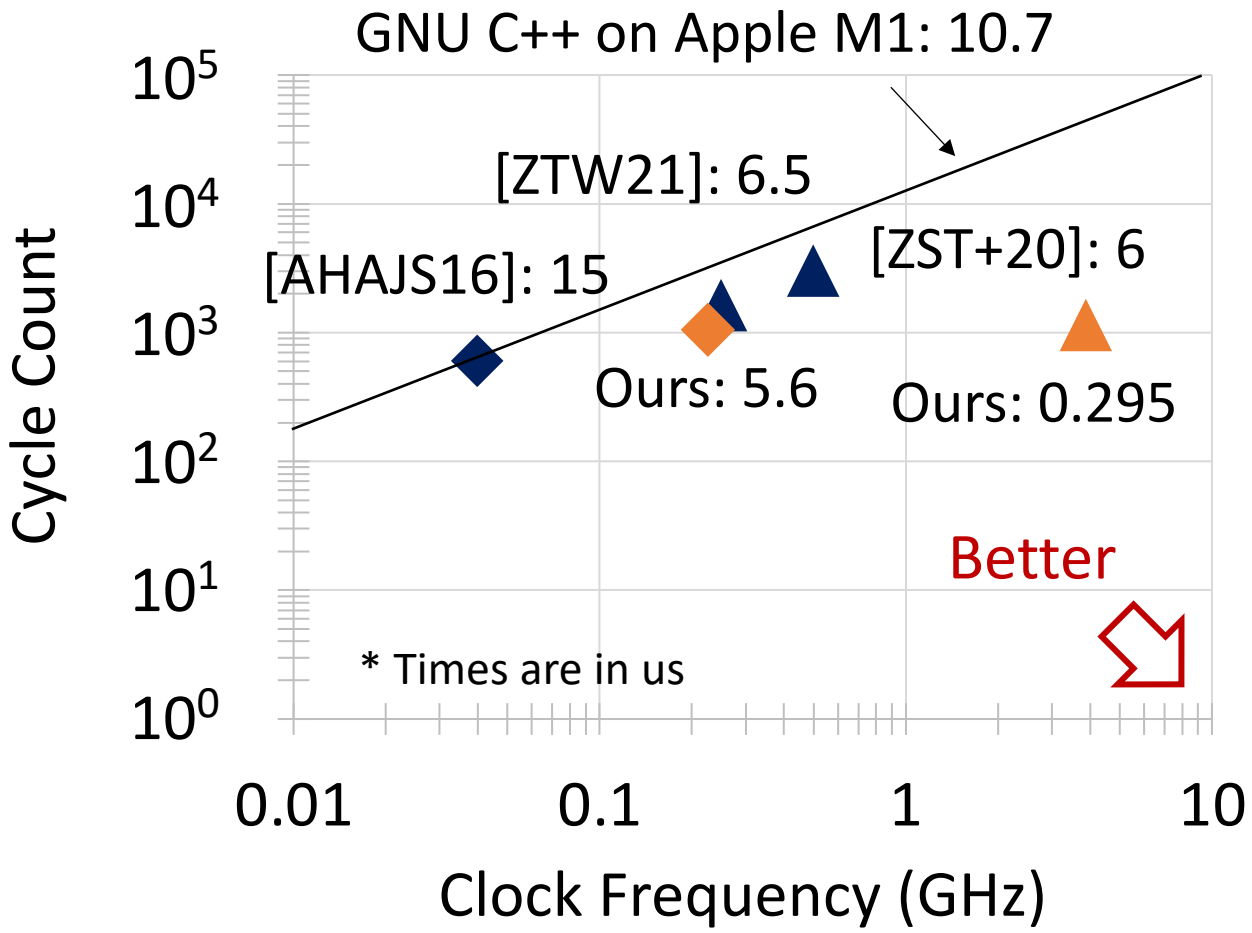
1024-bit XGCD Comparison



Our ASIC

- 36X faster than software
- 8X faster than state-of-the-art ASIC

1024-bit XGCD Comparison



Our ASIC

- 36X faster than software
- 8X faster than state-of-the-art ASIC

Direct FPGA Comparison

Our design is 2.7X faster

Our design impacts application approaches

1. Supports progression in state of the art for Curve25519

Our design impacts application approaches

1. Supports progression in state of the art for Curve25519

2. Informs reasonable security levels for this type of VDF

Our design impacts application approaches

1. Supports progression in state of the art for Curve25519
2. Informs reasonable security levels for this type of VDF
3. May be useful for other applications

Our design impacts application approaches

1. Supports progression in state of the art for Curve25519
2. Informs reasonable security levels for this type of VDF
3. May be useful for other applications

https://github.com/kavyasreedhar/sreedhar-xgcd-hardware-ches2022