Hardware Accelerated Proofs

Erdinc Ozturk, Justin Drake, Simon Peffers, Sean Gulley, Kelly Olson

05.14.20

Outline

- Why acceleration
- RSA/ECC primitives
- Algorithms
- Platforms - CPU, GPU, FPGA, ASIC
- Performance comparisons
- SNARKs

Intro

Why Accelerate?

Conventional cryptography is well-studied
Innovation in cryptography is enabling new capabilities
- VDFs, proofs of exponentiation, SNARKs, accumulators, ...
- Unbiasable randomness, privacy, scalability, compression (compute/storage), ...

Most of these are computationally intensive

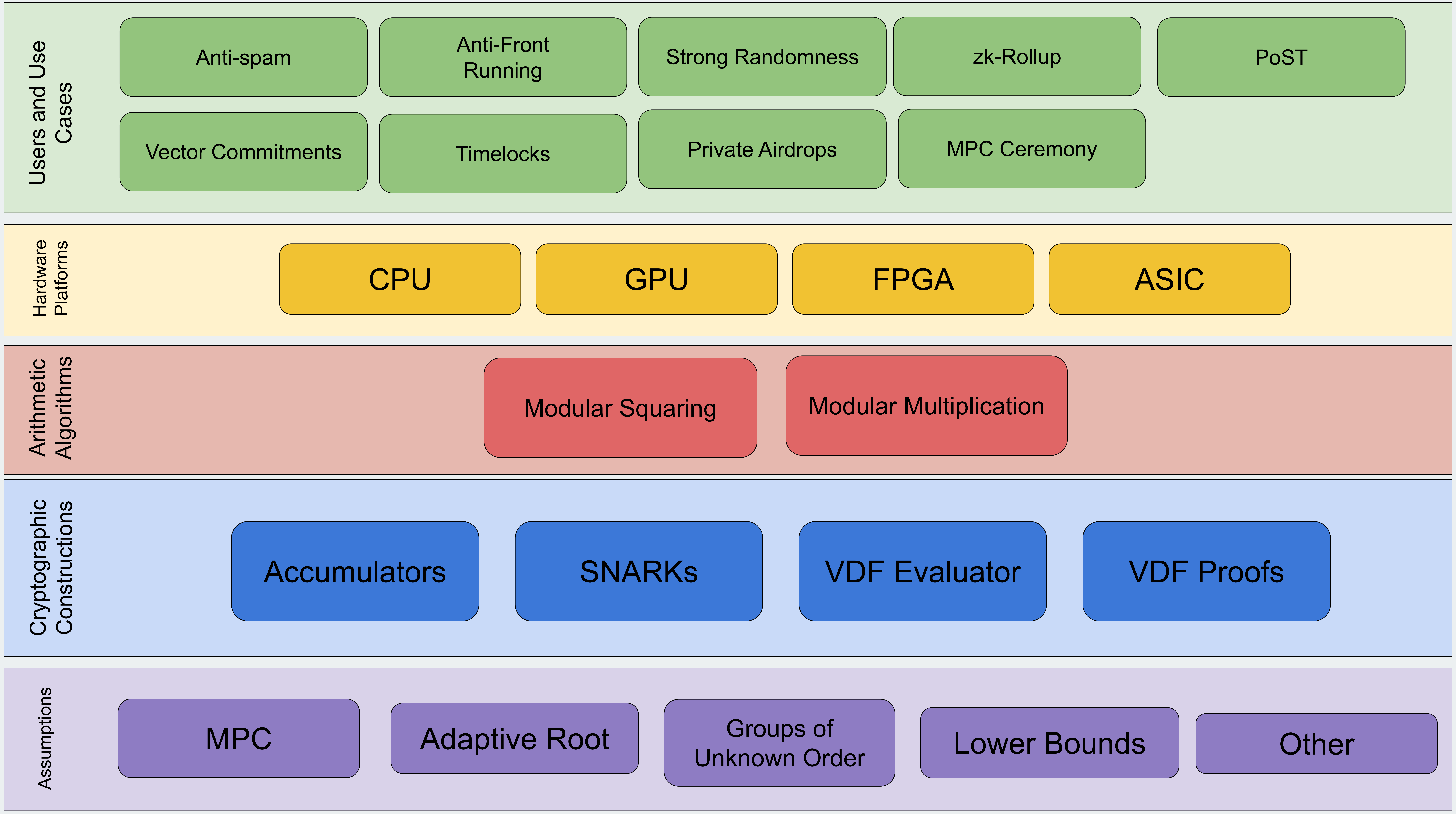
Acceleration enables:
- Higher security: larger key size = increased security
- Greater scale
- Improved user experience
- Lower costs

The same large integer arithmetic is fundamental across cryptography - RSA, ECC, etc.

Process

Recipe for Acceleration

- Define the use case
- Benchmark / profile to find bottlenecks
- Identify the underlying key operations to accelerate
- Select target platform(s) based on required performance characteristics
- Carefully map the operations onto the capabilities provided by the target platform


Users and Use Cases
Anti-spam
Anti-Front Running
Strong Randomness
zk-Rollup
PoST
Vector Commitments
Timelocks
Private Airdrops
MPC Ceremony
Hardware Platforms
CPU
GPU
FPGA
ASIC
Arithmetic Algorithms
Modular Squaring
Modular Multiplication
Cryptographic Constructions
Accumulators
SNARKs
VDF Evaluator
VDF Proofs
Assumptions
MPC
Adaptive Root
Groups of Unknown Order
Lower Bounds
Other

RSA Primitives

RSA consists of modular exponentiation

Exponentiation:	x^y	<- square and multiply
VDF:	x^(2^t)	<- repeated square

Many different exponentiation techniques

- Example: left-to-right square and multiply
- Example: left-to-right square and multiply with windowing -> precomputed lookup tables
- All rely on these large integer modular arithmetic primitives:
    - Square
    - Multiply
    - Add
    - Subtract

ECC Primitives

ECC consists of scalar point multiplication

Q=k*P

All exponentiation techniques can be used for scalar point multiplication.
High level primitives are replaced:

Modular squaring	-> point doubling
Modular multiplication	-> point addition

Point addition and Point doubling operations rely on same large integer modular arithmetic primitives:

Smaller operands

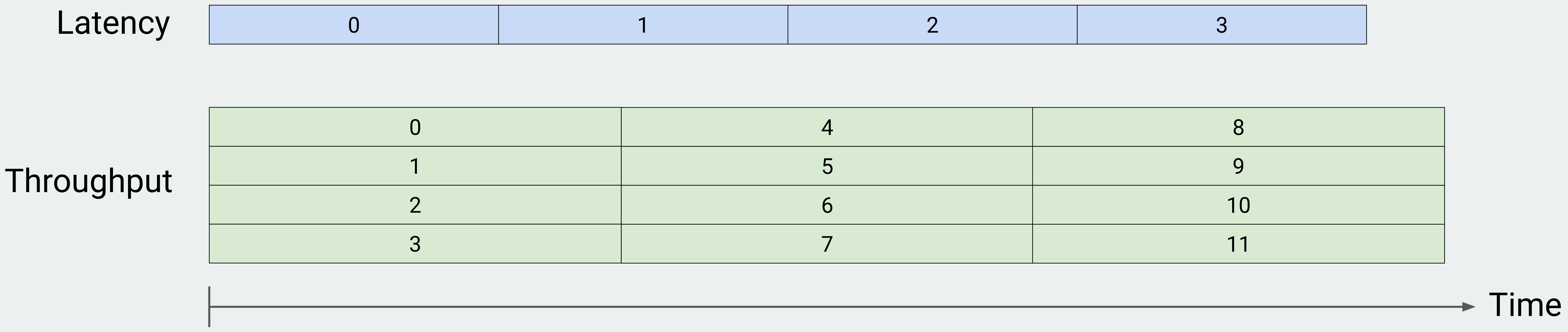
One major caveat: point at infinity

Latency vs. Throughput Oriented Problems

Latency oriented is performing an individual task as quickly as possible
- Example - VDF Evaluation

Throughput oriented trades off time of an individual task for performing multiple tasks in parallel
- Example - VDF Proof

Choose the appropriate algorithm and platform based on problem priority



Background

Performance Impacts - Algorithms

Implementation algorithms can lead to orders of magnitude difference in performance

Examples

Modular Squaring/Multiplication

- Schoolbook, Comba, Karatsuba, Schonhage-Strassen
- Montgomery, Barrett

Modular Exponentiation

- Binary square and multiply, Montgomery Ladder, Sliding window, wNAF, Addition Chain

Modular Multi-Exponentiation

- Pippenger, Bos-Coster, Lim-Lee, Comb, Shamir

Exponentiation/Multi-Exponentiation Considerations

- Fixed base, fixed exponent

Multi-exponentiation

Many different flavors:

- Different Base, Different Exponent: g_1^{x1} g_2^{x2} … g_n^{xn}
- Same Base, Different Exponent: g^{x1} g^{x2} … g^{xn}
- Different Base, Same Exponent: g_1^{x} g_2^{x} … g_n^{x}

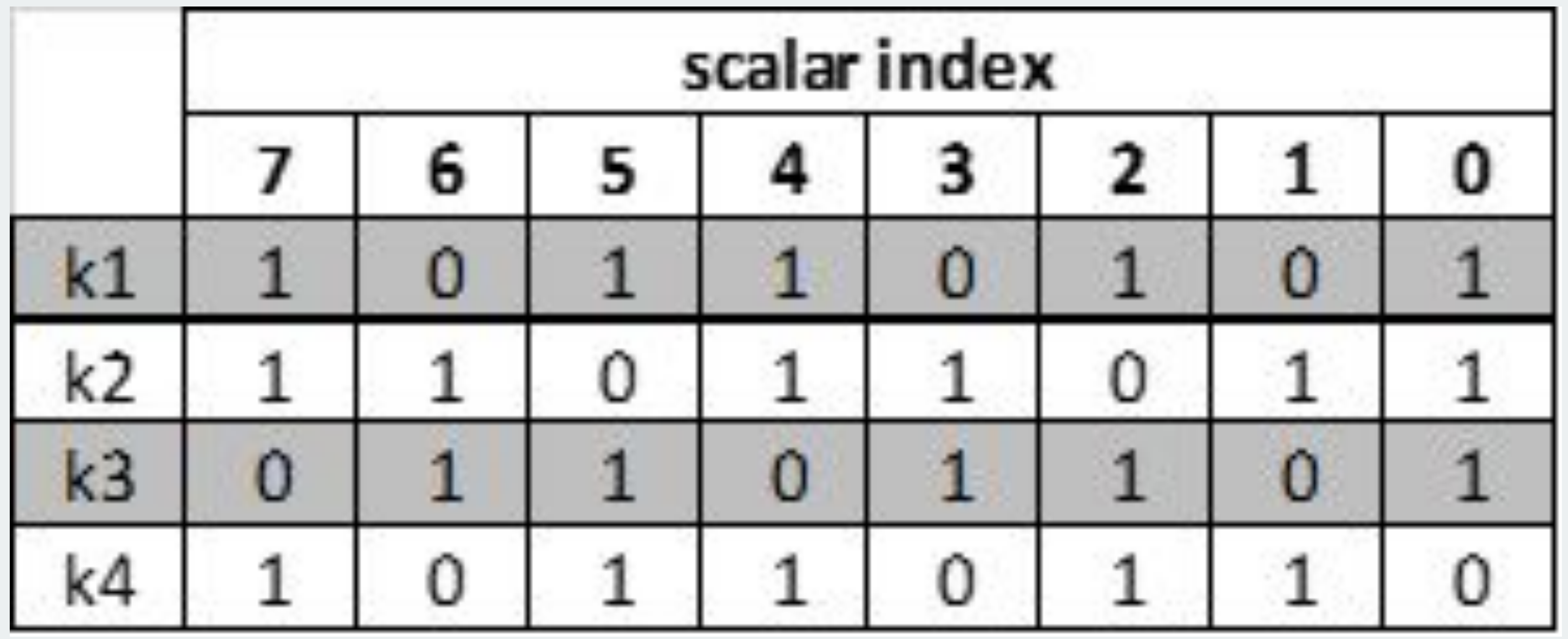
ECC:

- Different Base, Different Exponent: k_1*P_1 k_2*P_2 … k_n*P_n
- Same Base, Different Exponent: k_1*P k_2*P … k_n*P
- Different Base, Same Exponent: $k*P_1$ $k*P_2$ … $k*P_n$

Multi-exponentiation

Example:

Calculate $(k_1*P_1)+(k_2*P_2)+(k_3*P_3)+(k_4*P_4)$



	scalar index							
	7	6	5	4	3	2	1	0
k1	1	0	1	1	0	1	0	1
k2	1	1	0	1	1	0	1	1
k3	0	1	1	0	1	1	0	1
k4	1	0	1	1	0	1	1	0

Complexity for 4 bases:

Single exponentiation:	Point Doublings: 4*7	Point Additions: 20
Multi exponentiation:	Point Doublings: 7	Point Additions: 20

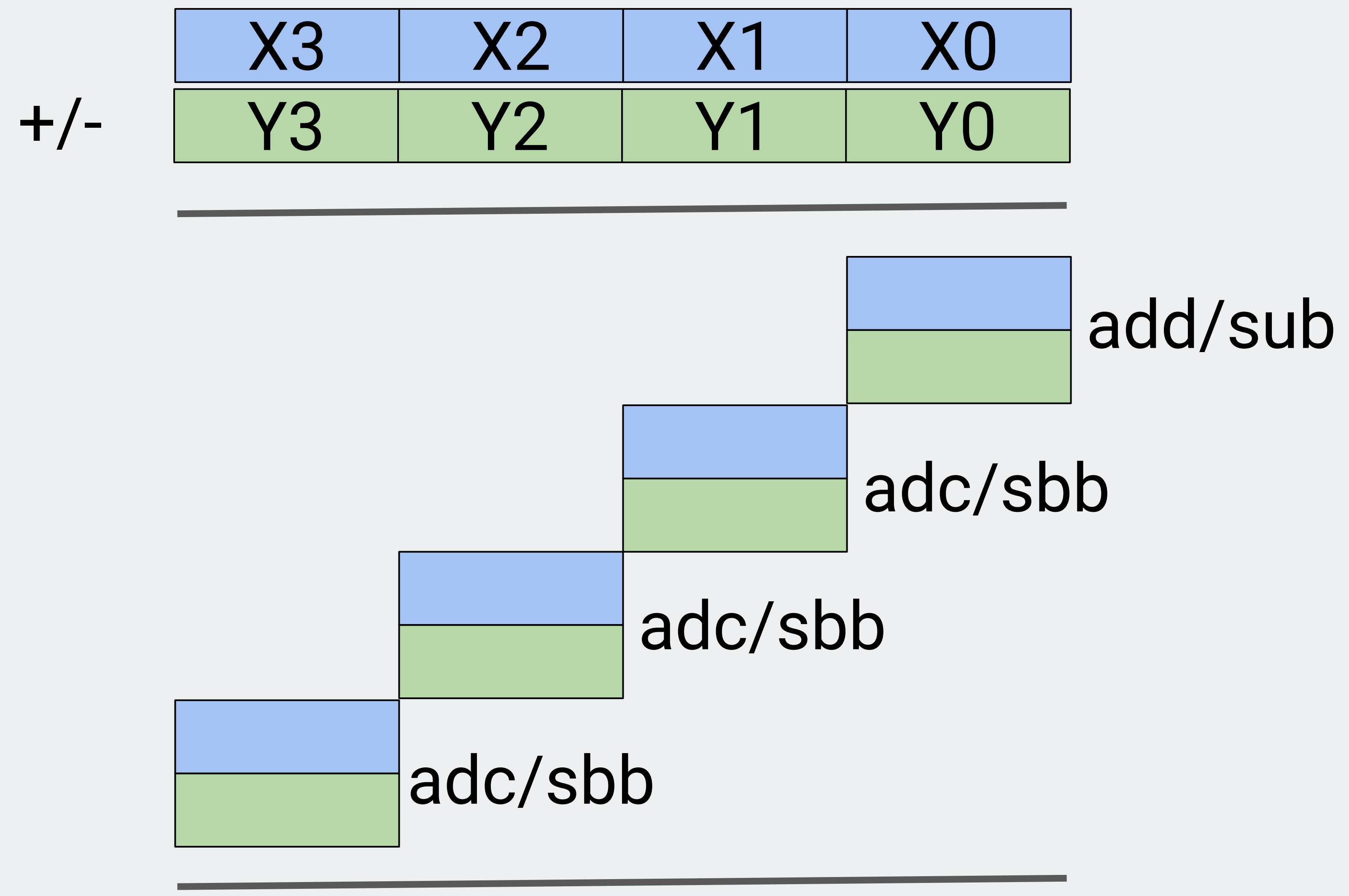
No constant time!

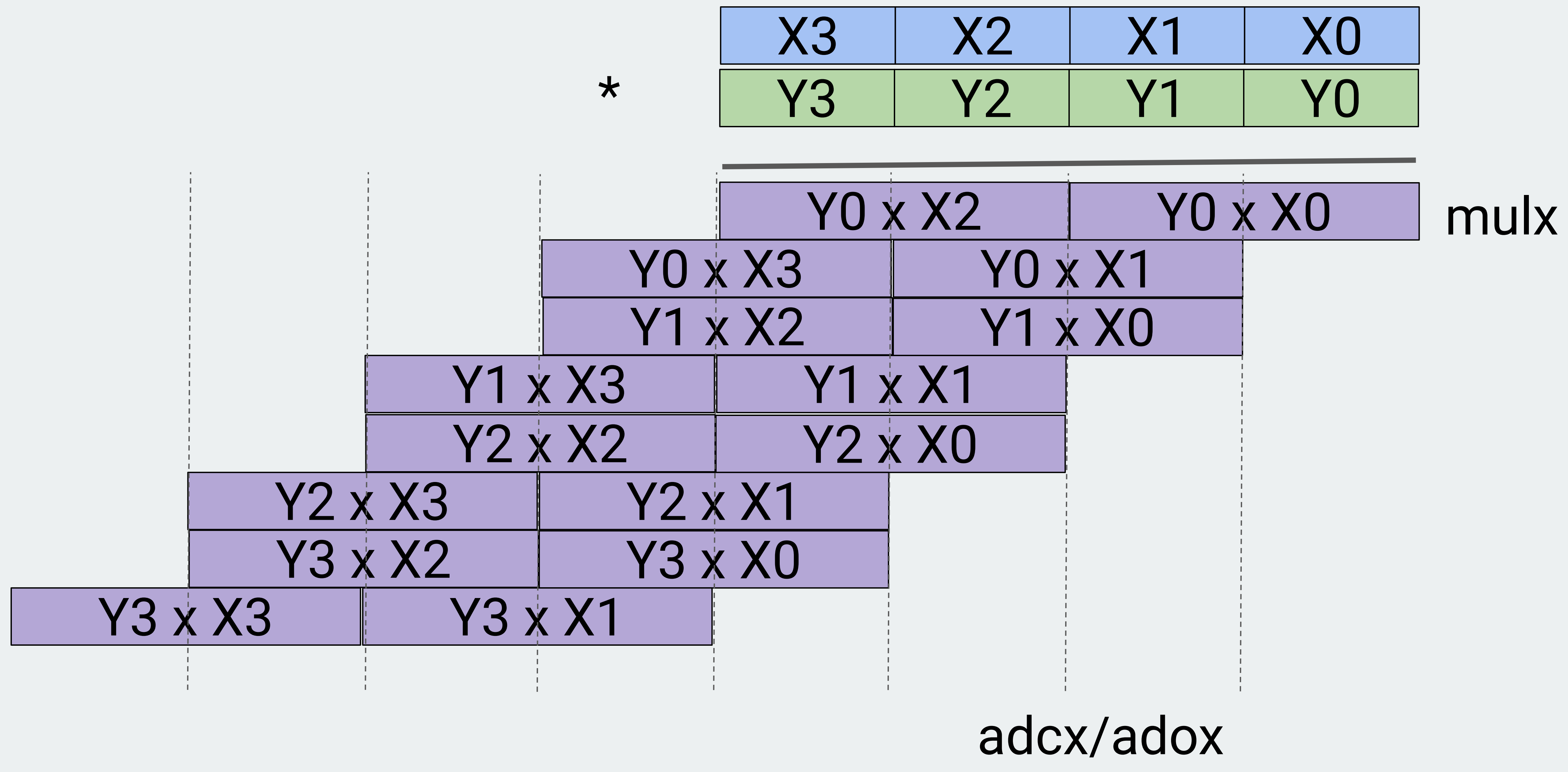
Large Integer Arithmetic

RSA integer sizes typically exceed compute element word size (e.g. 8, 16, 32, 64 bits)

Break integer down into n limbs: X = (X[n-1], …, X[1], X[0])

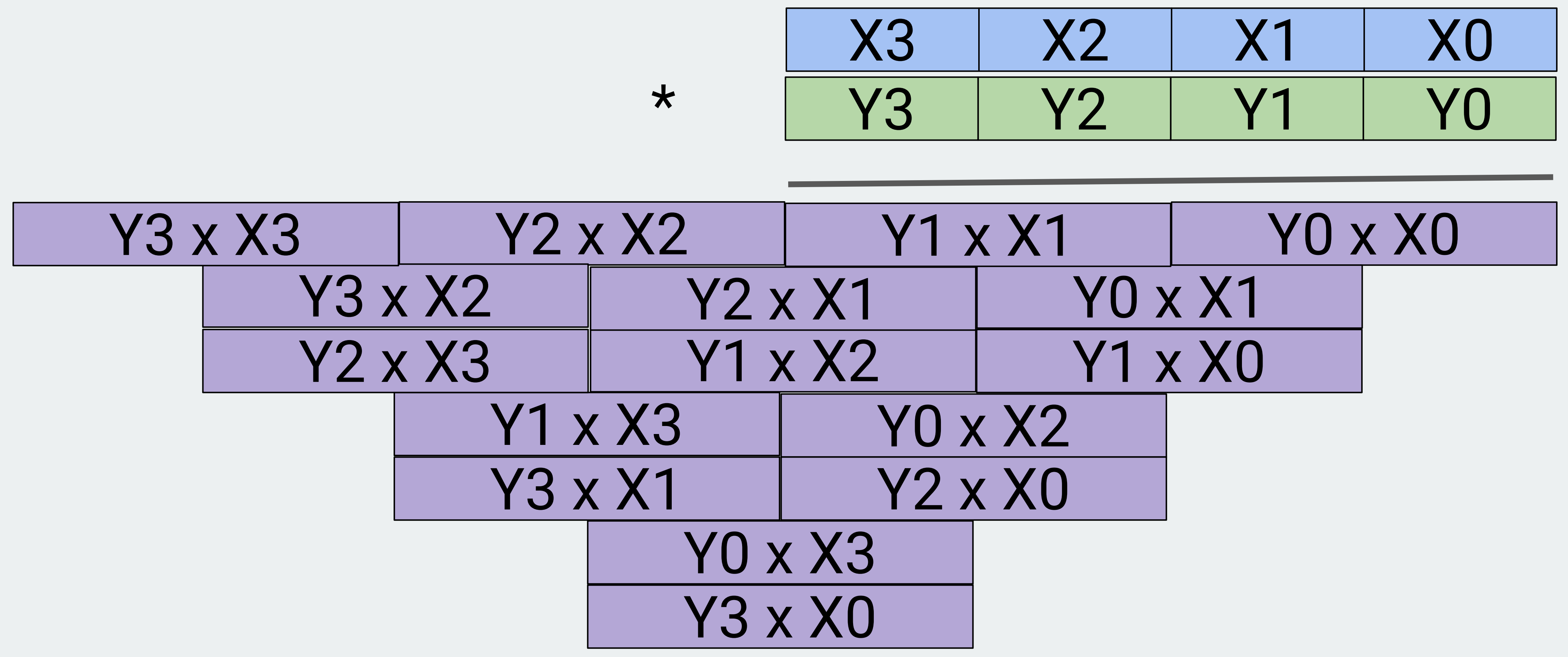
n = ceil(len/w) where len is the integers bit length and w is compute word size

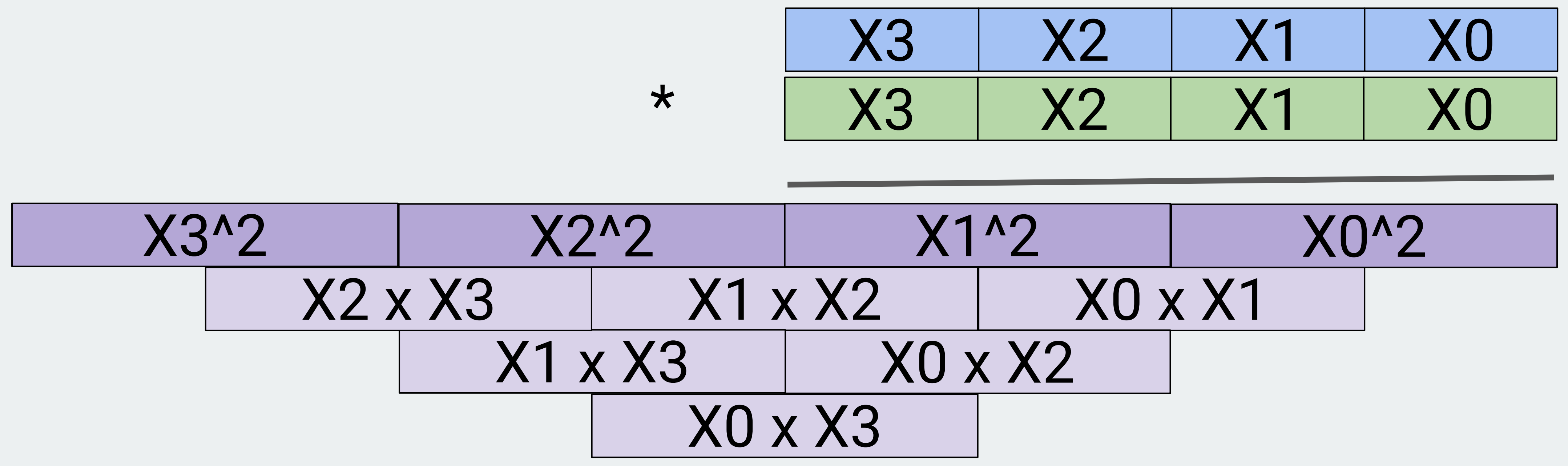




Parallel carry chain, use adox for low word accumulation and adcx for high word (or vice versa)

Large Integer Arithmetic

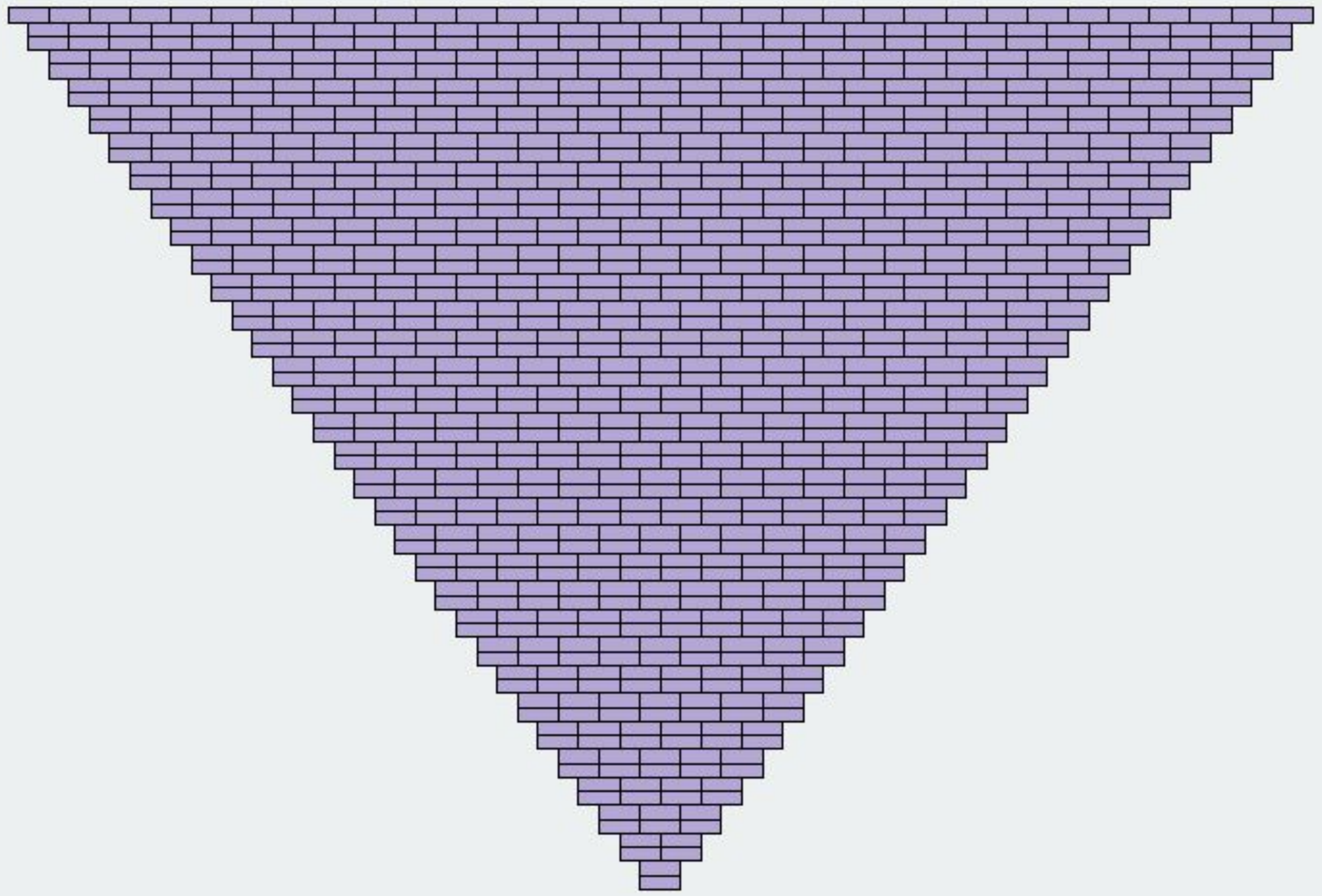




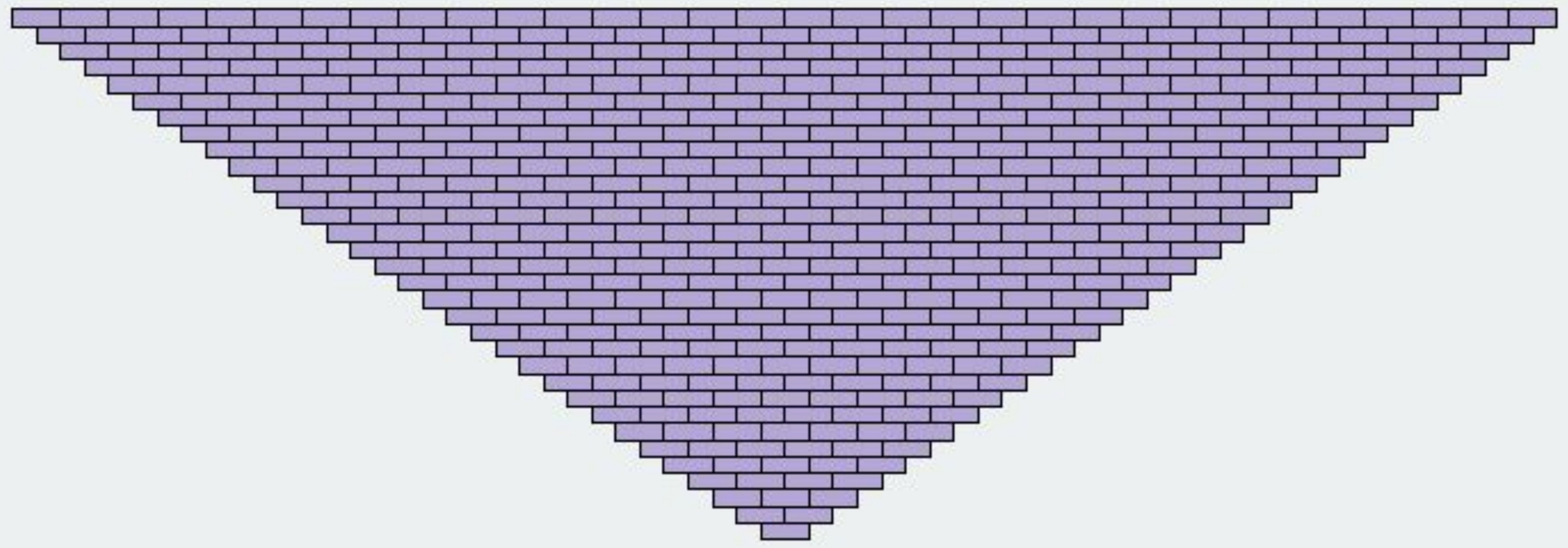
Schoolbook multiply requires n^2 operations

Square requires (n^2 + n)/2 operations
Light purple partial products are doubled

Large Integer Arithmetic



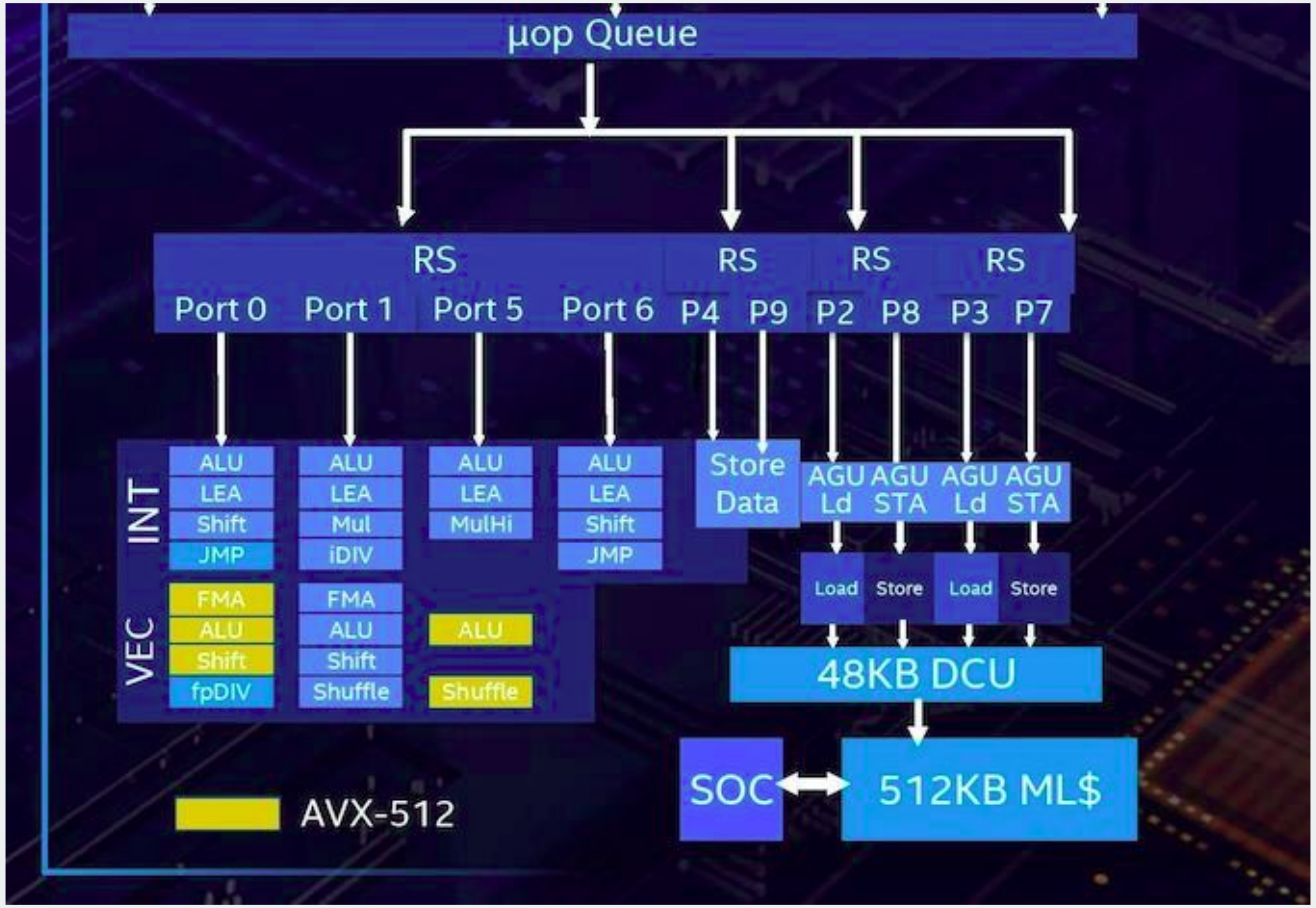
Schoolbook multiply requires 1024 operations



Square requires 528 operations

RSA Arithmetic Platform Comparison

Performance Impacts - Platform - CPU

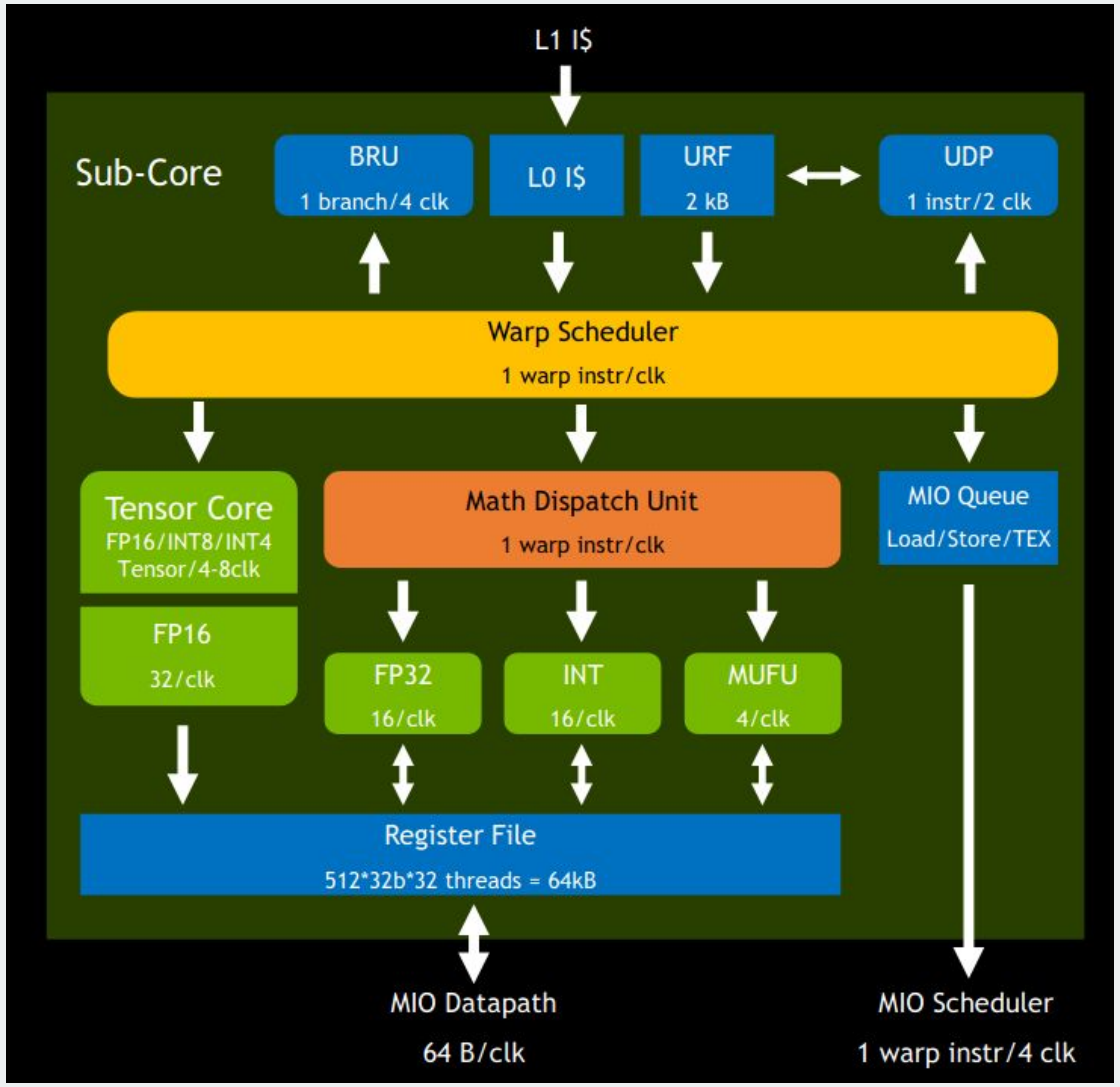


Frequency	5 Ghz
Multiplier Size	64-bit
Cores	8
Multiplies Req'd for 2K	1024
Theoretical Throughput	39 M ops/s

2048-bit multiply throughput

https://www.anandtech.com/show/14514/examining-intels-ice-lake-microarchitecture-and-sunny-cove/3

Performance Impacts - Platform - GPU



Frequency	1545 MHz
Multiplier Size	32-bit
Cores	4608
Multiplies Req'd for 2K	4096
Theoretical Throughput	1738 M ops/s

2048-bit multiply throughput

https://www.hotchips.org/hc31/HC31_2.12_NVIDIA_final.pdf

Performance Impacts - Platform - FPGA

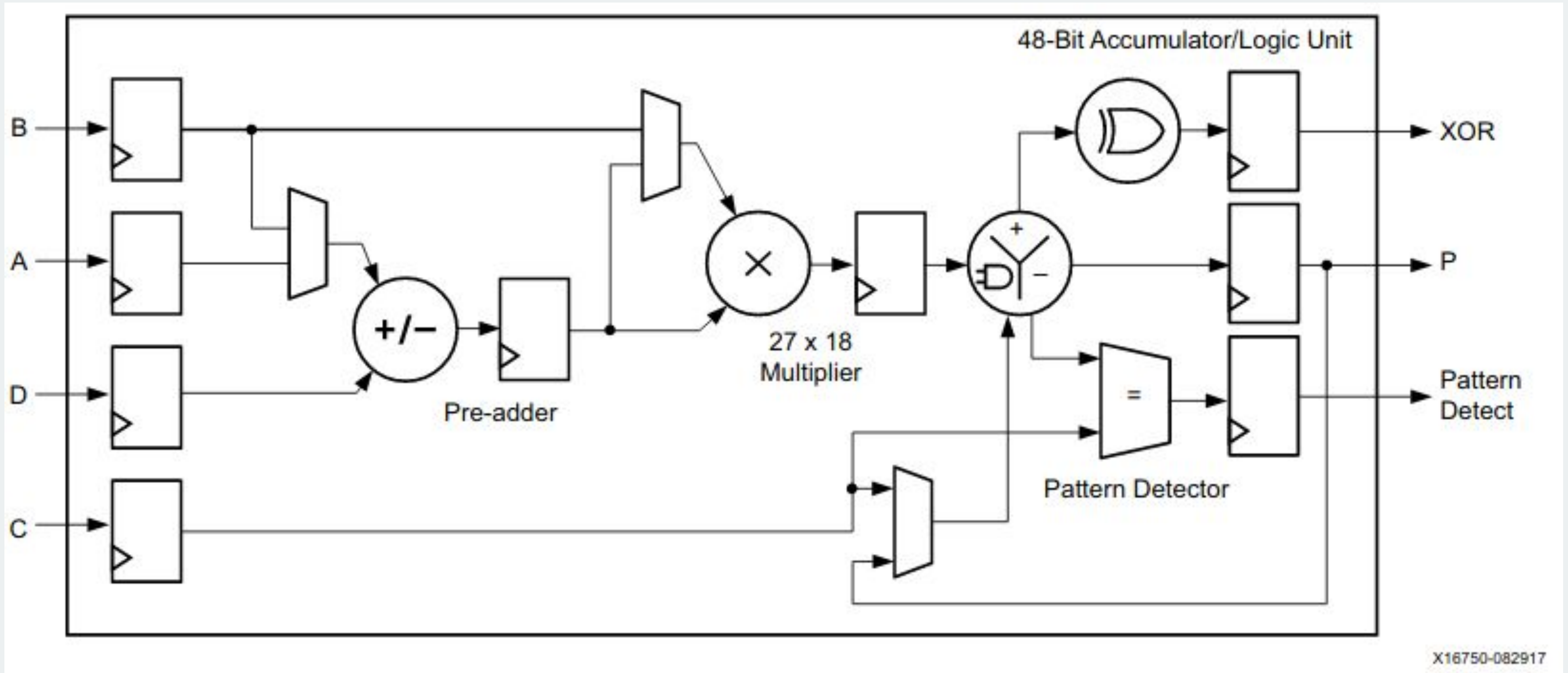


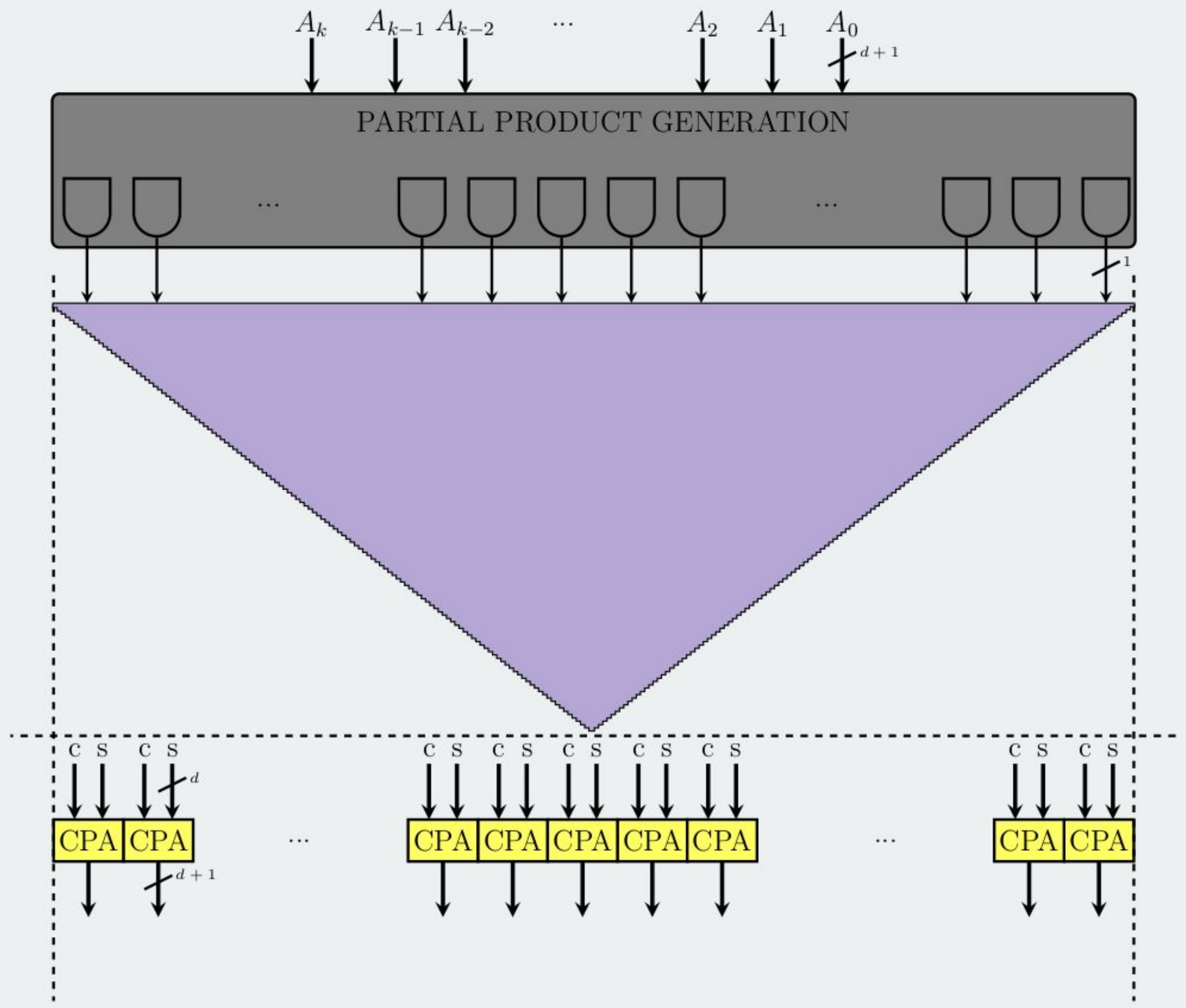
Figure 1-1: **Basic DSP48E2 Functionality**

Frequency	462 MHz
Multiplier Size	27x18-bit
Cores	6840
Multiplies Req'd for 2K	9678
Theoretical Throughput	327 M ops/s

2048-bit multiply throughput

https://www.xilinx.com/support/documentation/user_guides/ug579-ultrascale-dsp.pdf

Performance Impacts - Platform - ASIC (estimated)



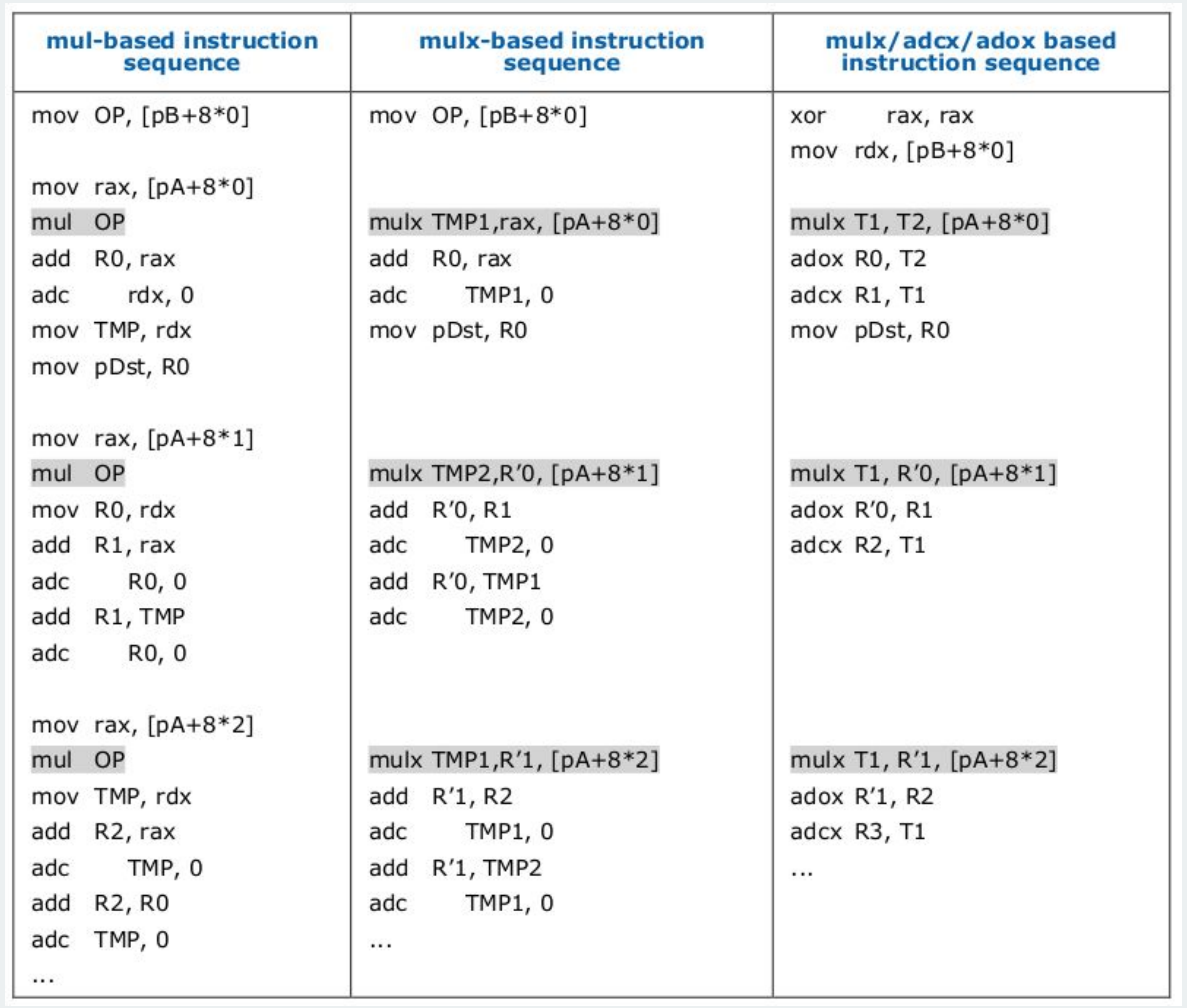
Frequency	1 GHz
Multiplier Size	256-bit
Cores	1,000
Multiplies Req'd for 2K	64
Theoretical Throughput	15,625 M ops/s

2048-bit multiply throughput

Low latency design:	2048 bit redundant representation
High throughput design:	256 bit redundant representation

Performance Impacts - CPU Example

Scalar



mul-based instruction sequence	mulx-based instruction sequence	mulx/adcx/adox based instruction sequence
mov OP, [pB+8*0]	mov OP, [pB+8*0]	xor rax, rax
		mov rdx, [pB+8*0]
mov rax, [pA+8*0]		
mul OP	mulx TMP1,rax, [pA+8*0]	mulx T1, T2, [pA+8*0]
add R0, rax	add R0, rax	adox R0, T2
adc rdx, 0	adc TMP1, 0	adcx R1, T1
mov TMP, rdx	mov pDst, R0	mov pDst, R0
mov pDst, R0		
mov rax, [pA+8*1]		
mul OP	mulx TMP2,R’0, [pA+8*1]	mulx T1, R’0, [pA+8*1]
mov R0, rdx	add R’0, R1	adox R’0, R1
add R1, rax	adc TMP2, 0	adcx R2, T1
adc R0, 0	add R’0, TMP1	
add R1, TMP	adc TMP2, 0	
adc R0, 0		
mov rax, [pA+8*2]		
mul OP	mulx TMP1,R’1, [pA+8*2]	mulx T1, R’1, [pA+8*2]
mov TMP, rdx	add R’1, R2	adox R’1, R2
add R2, rax	adc TMP1, 0	adcx R3, T1
adc TMP, 0	add R’1, TMP2	…
add R2, R0	adc TMP1, 0	
adc TMP, 0	…	
…		

Vector

New instructions:
512-bit Integer Fused Multiply Add (IFMA)

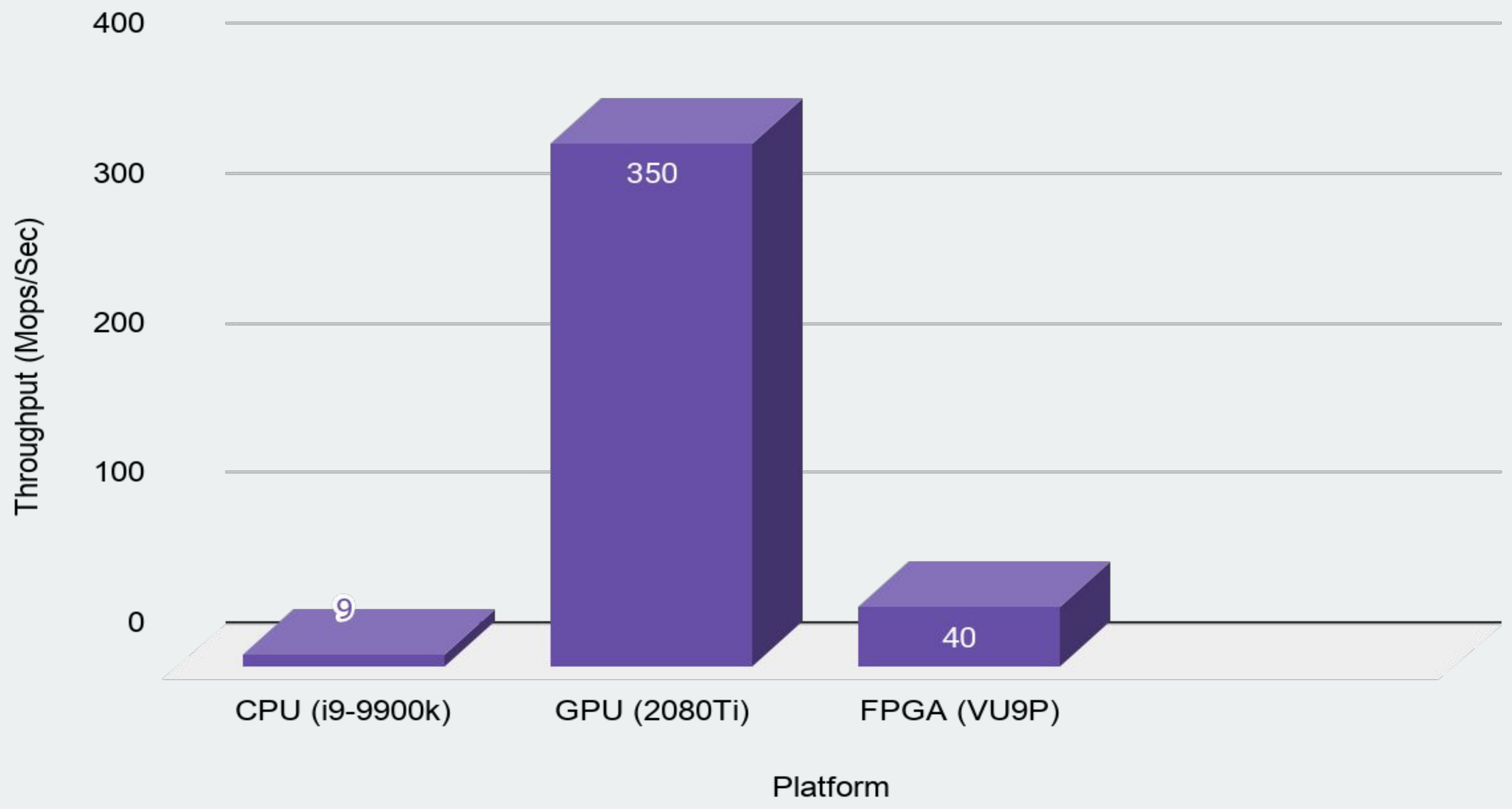
```
VPMADD52LUQ zmm1 {k1}{z},
zmm2,zmm3/m512/m64bcst

VPMADD52HUQ zmm1 {k1}{z},
zmm2,zmm3/m512/m64bcst
```

8 lanes of 52x52 + 64
First instruction does low result accumulate
Second instruction does high result accumulate

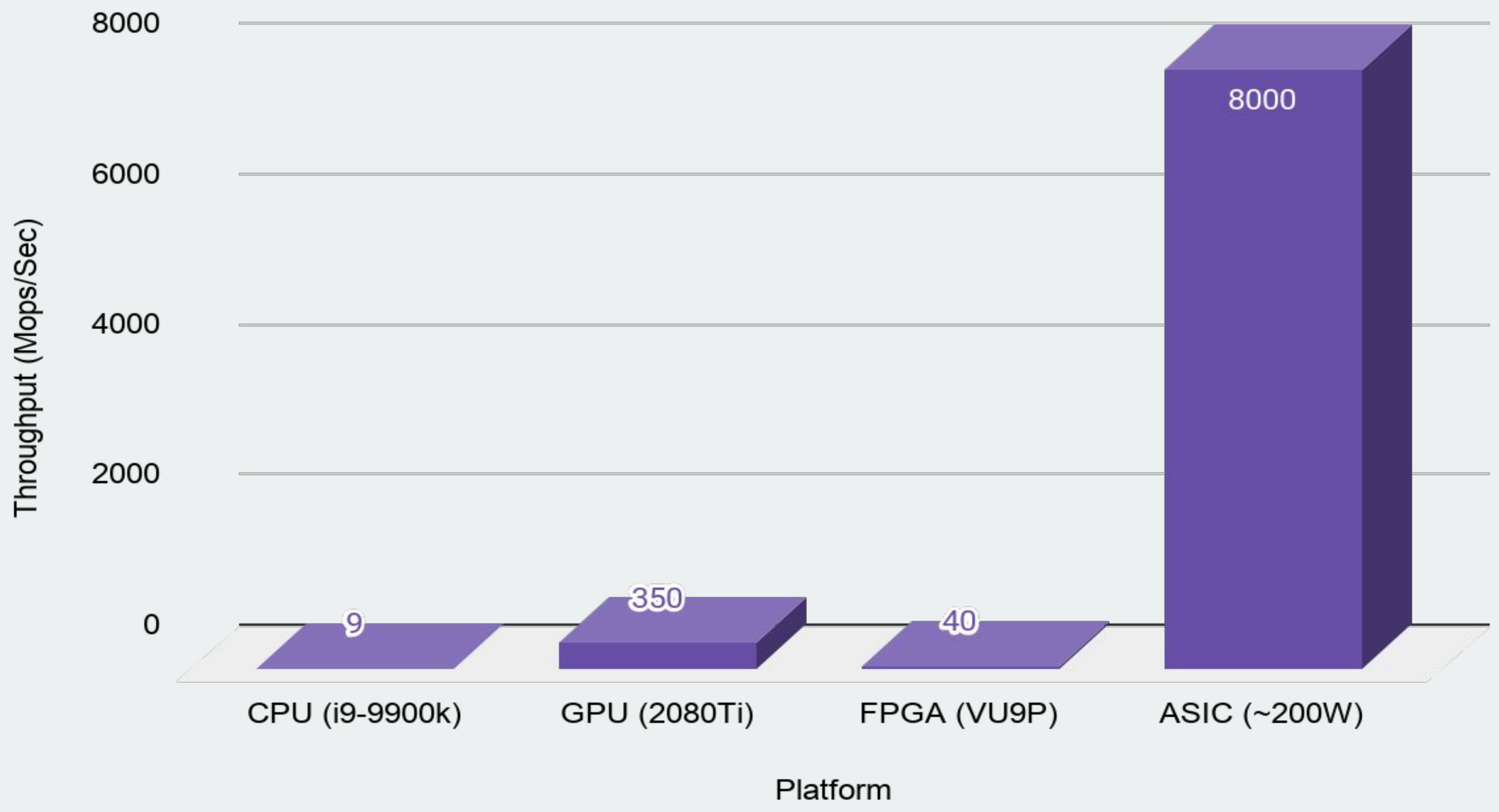
New Instructions Supporting Large Integer Arithmetic on Intel® Architecture Processors
https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ia-large-integer-arithmetic-paper.pdf

Throughput

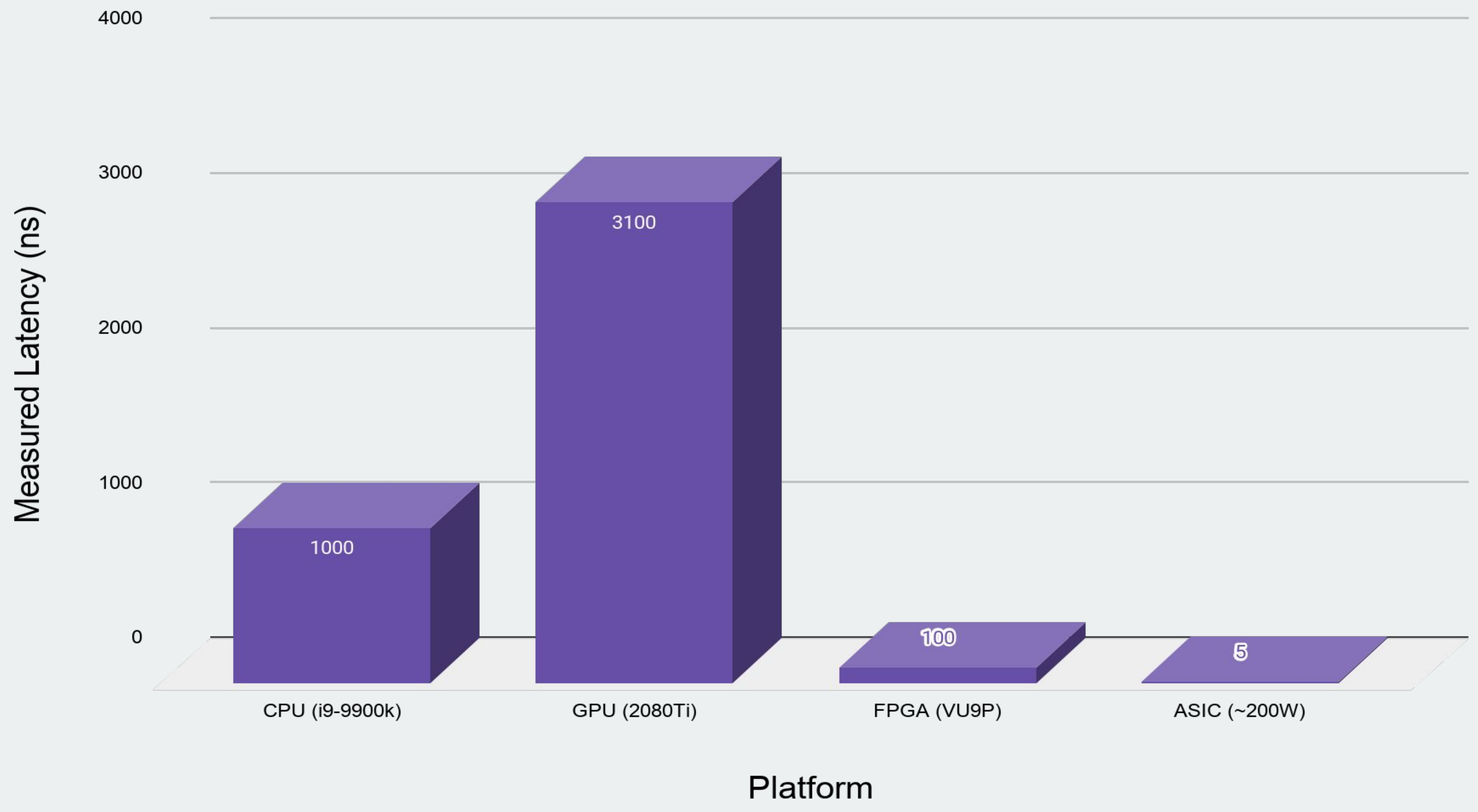


Measure 2k Montgomery Multiplication

Throughput


8000
6000
4000
2000
0
Throughput (Mops/Sec)
9
350
40
8000
CPU (i9-9900k)
GPU (2080Ti)
FPGA (VU9P)
ASIC (~200W)
Platform

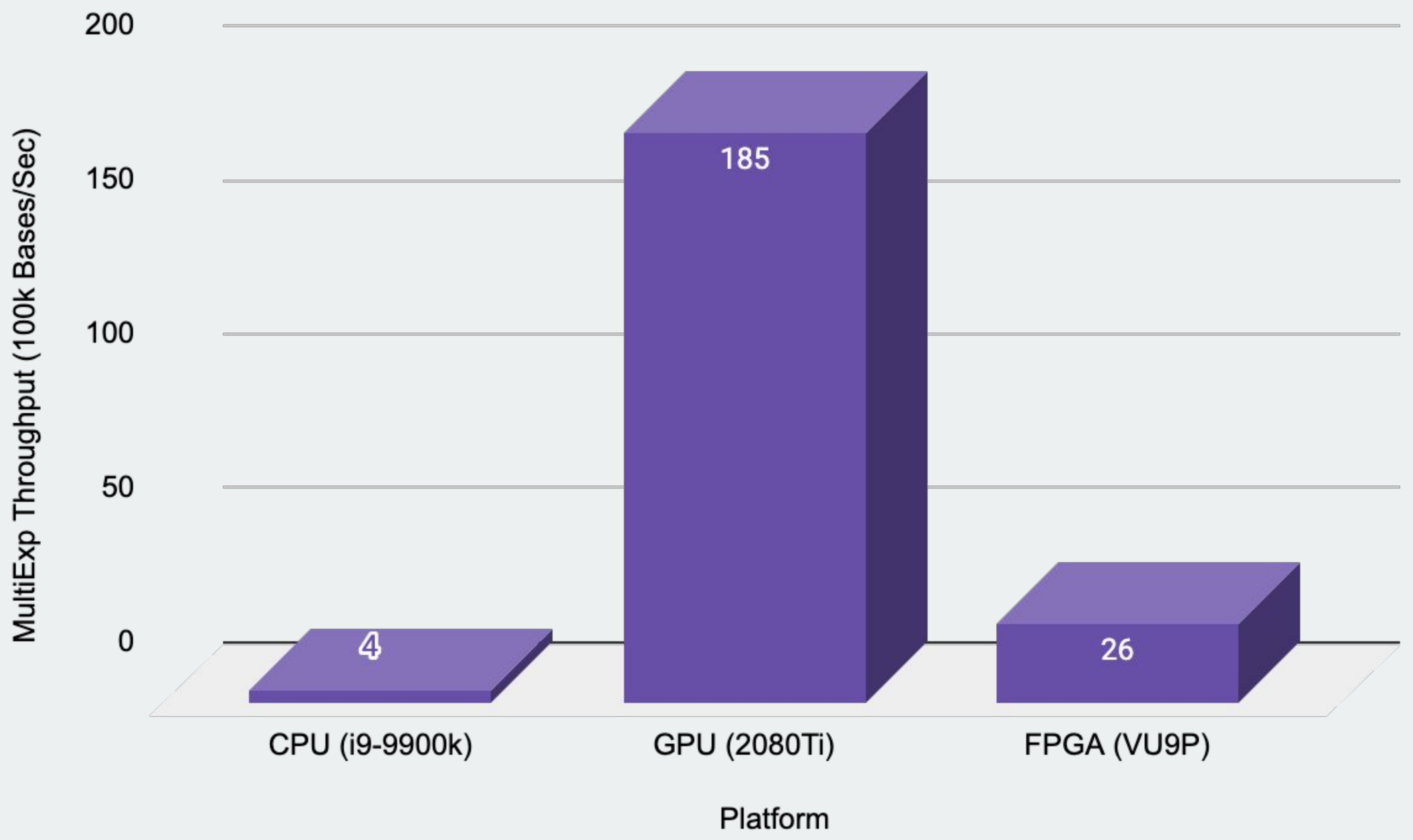
Latency



4000
3000
2000
1000
0
Measured Latency (ns)
1000
3100
100
5
CPU (i9-9900k)
GPU (2080Ti)
FPGA (VU9P)
ASIC (~200W)
Platform

BLS-381 MultiExp Platform Comparison

BLS-381 MultiExp Throughput



SNARK ASIC

SNARK ASIC Strawman Proposal (technical)

- **multiexponentations only**—no FFTs, no witness generation

SNARK ASIC Strawman Proposal (technical)

- **multiexponentations only**—no FFTs, no witness generation
- **BLS12-381 only**—"blockchain standard", secure, efficient recursion (~1m gates overhead)

SNARK ASIC Strawman Proposal (technical)

- **multiexponentations only**—no FFTs, no witness generation
- **BLS12-381 only**—"blockchain standard", secure, efficient recursion (~1m gates overhead)
- **28nm process**—no costly FinFET process (sufficient for PCIe4, GDDR5)

SNARK ASIC Strawman Proposal (technical)

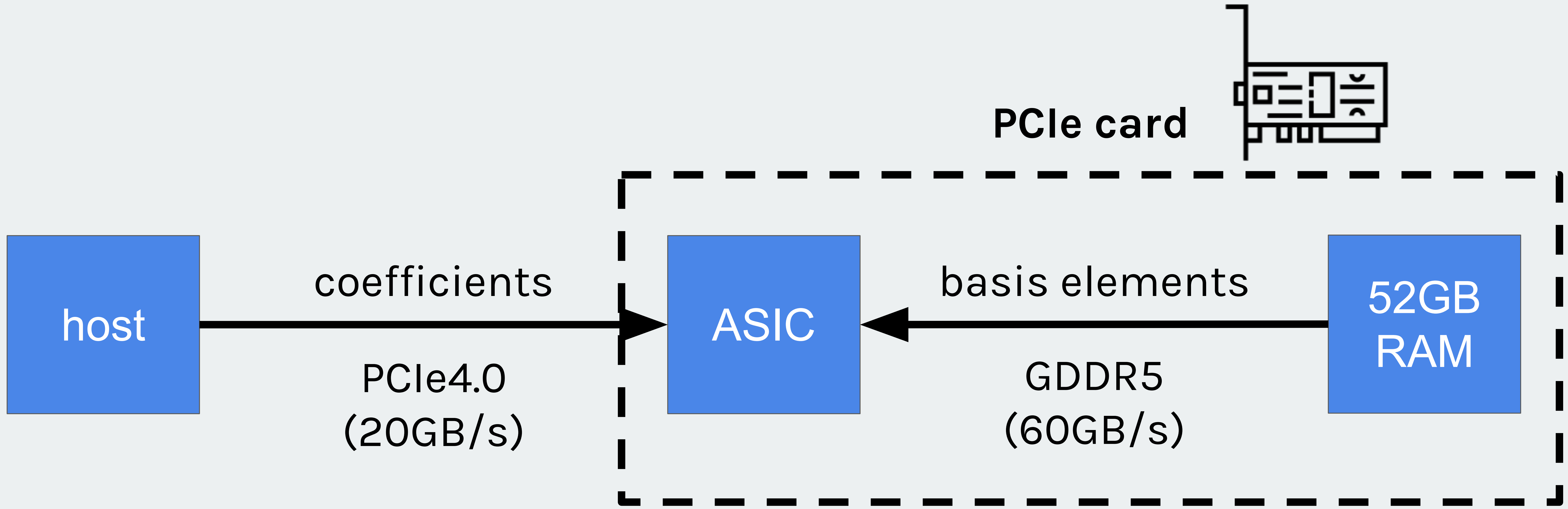
- **multiexponentations only**—no FFTs, no witness generation
- **BLS12-381 only**—"blockchain standard", secure, efficient recursion (~1m gates overhead)
- **28nm process**—no costly FinFET process (sufficient for PCIe4, GDDR5)
- **PCIe4.0**—cost effective, high bandwidth, convenient

SNARK ASIC Strawman Proposal (technical)

- **multiexponentations only**—no FFTs, no witness generation
- **BLS12-381 only**—"blockchain standard", secure, efficient recursion (~1m gates overhead)
- **28nm process**—no costly FinFET process (sufficient for PCIe4, GDDR5)
- **PCIe4.0**—cost effective, high bandwidth, convenient
- **GDDR5 RAM**—4x bandwidth boost versus PCIe alone, less host RAM

SNARK ASIC Strawman Proposal (technical)

- **multiexponentations only**—no FFTs, no witness generation
- **BLS12-381 only**—"blockchain standard", secure, efficient recursion (~1m gates overhead)
- **28nm process**—no costly FinFET process (sufficient for PCIe4, GDDR5)
- **PCIe4.0**—cost effective, high bandwidth, convenient
- **GDDR5 RAM**—4x bandwidth boost versus PCIe alone, less host RAM



SNARK ASIC Strawman Proposal (non-technical)

- **Supranational**—trusted to deliver

SNARK ASIC Strawman Proposal (non-technical)

- **Supranational**—trusted to deliver
- **public good**—subsidized NRE (~$5m), PCIe cards sold at cost (~$1K)

SNARK ASIC Strawman Proposal (non-technical)

- **Supranational**—trusted to deliver
- **public good**—subsidized NRE (~$5m), PCIe cards sold at cost (~$1K)
- **open source**—open source RTL

SNARK ASIC Strawman Proposal (non-technical)

- **Supranational**—trusted to deliver
- **public good**—subsidized NRE (~$5m), PCIe cards sold at cost (~$1K)
- **open source**—open source RTL
- **2023**—immediately after the VDF ASIC

SNARK ASIC Strawman Proposal (non-technical)

- **Supranational**—trusted to deliver
- **public good**—subsidized NRE (~$5m), PCIe cards sold at cost (~$1K)
- **open source**—open source RTL
- **2023**—immediately after the VDF ASIC
- **funding**—Ethereum Foundation and others

SNARK ASIC Strawman Proposal (non-technical)

- **Supranational**—trusted to deliver
- **public good**—subsidized NRE (~$5m), PCIe cards sold at cost (~$1K)
- **open source**—open source RTL
- **2023**—immediately after the VDF ASIC
- **funding**—Ethereum Foundation and others

FUNDING

ethereum

COSMOS

Filecoin

NSF

Tezos

TOOLING

aws

SYNOPSYS®

XILINX

VDF Alliance partners

Open Questions for Discussion

technical

- **witness generation**—can we optimise to 0.1%? dial-down fancy non-determinism?
- **FFTs**—optional or essential? is 5x incremental speed-up valuable?
- **BLS12-381**—can it become a winner-take-most?
- **bottleneck**—bandwidth or compute? fast multi-point decompression?
- **recursion**—impact to hardware acceleration?

Open Questions for Discussion

technical

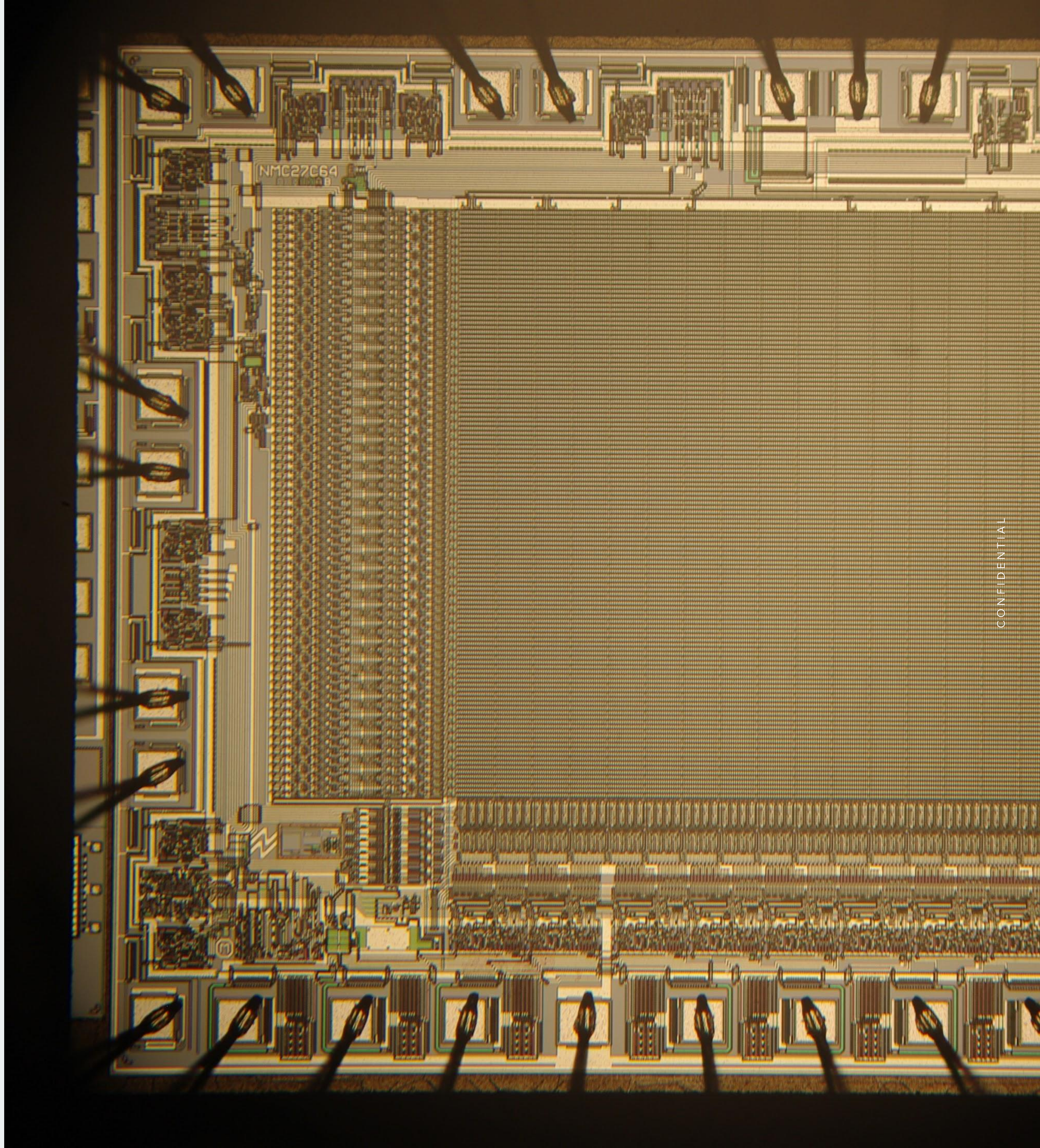
- **witness generation**—can we optimise to 0.1%? dial-down fancy non-determinism?
- **FFTs**—optional or essential? is 5x incremental speed-up valuable?
- **BLS12-381**—can it become a winner-take-most?
- **bottleneck**—bandwidth or compute? fast multi-point decompression?
- **recursion**—impact to hardware acceleration?

non-technical

- **adoption**—is 2023 good timing? what are burning use cases?
- **funding**—who's in? (Protocol Labs, Tezos, Interchain, Web3, NSF)

Questions?

hello@supranational.net



*"64kbit EPROM National Semiconductor NMC2" (CC By 2.0) by yellowcloud