

Logarithmic Numbers and Asynchronous Accumulators

The Future of DL Chips

April 5, 2021

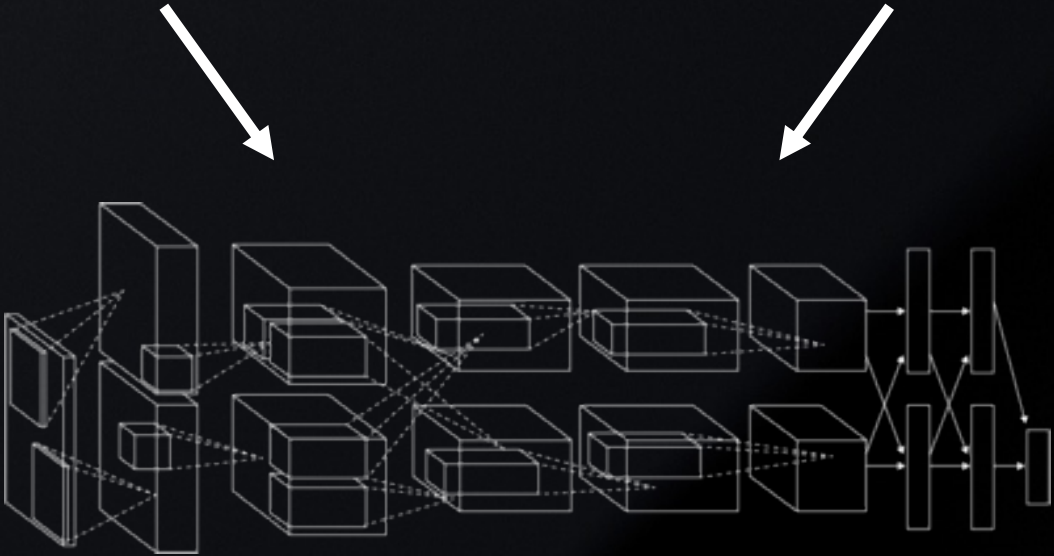
Bill Dally

Chief Scientist and SVP of Research, NVIDIA Corporation

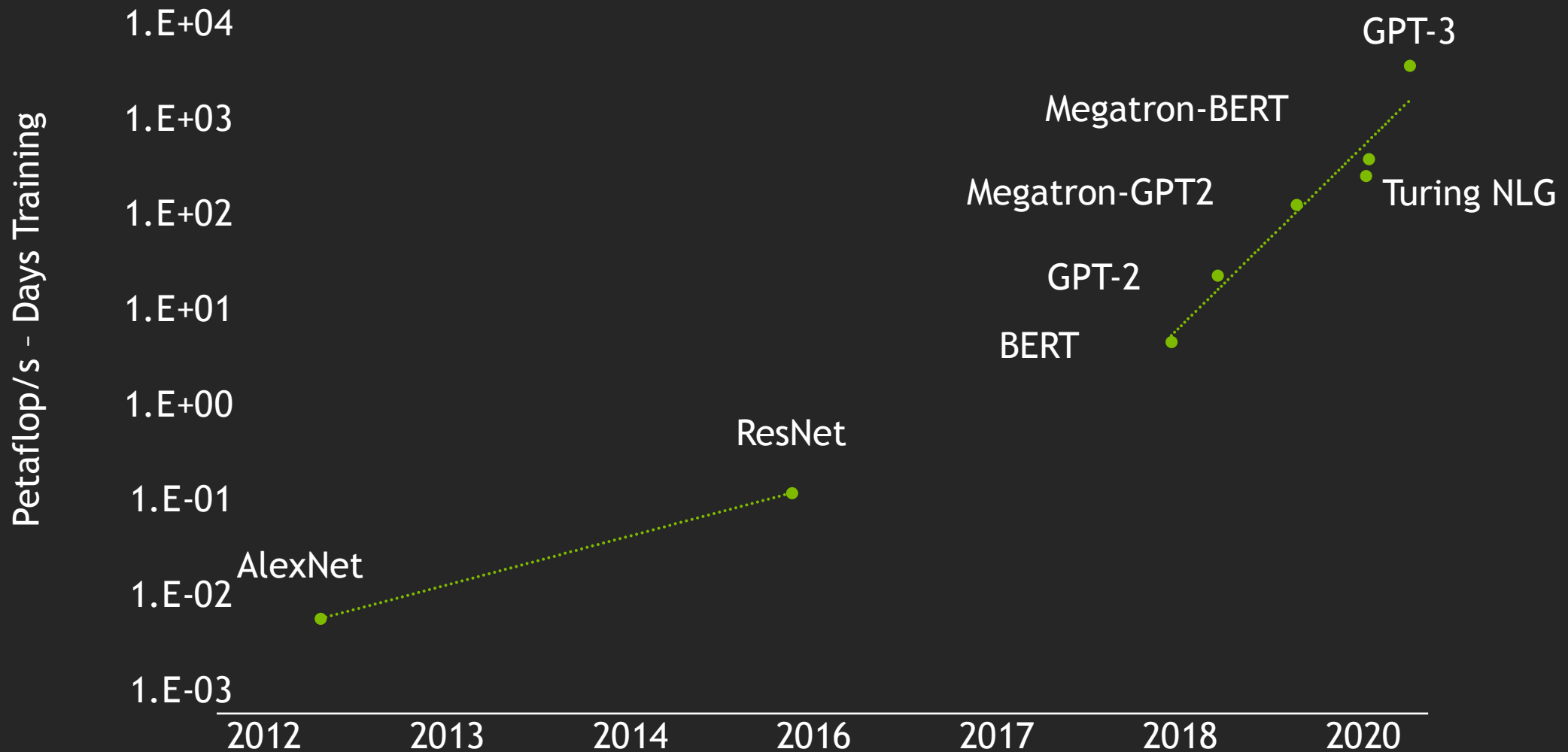
Adjunct Professor of CS and EE, Stanford

Motivation

Deep Learning was Enabled by Hardware



Deep Learning is Gated by Hardware

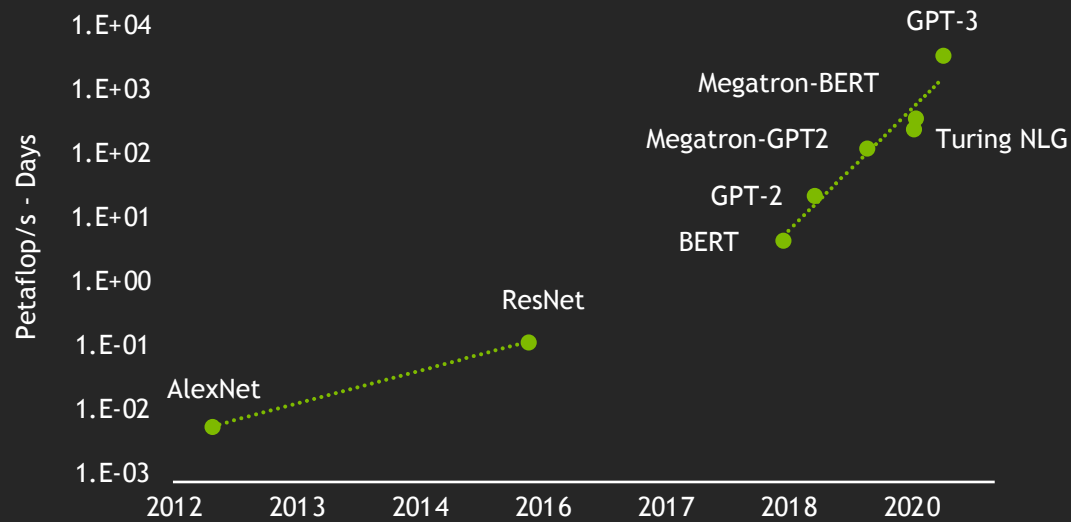


CHALLENGES: ACCELERATING BIG AND SMALL

Exploding Model Complexity

Distributed Pervasive Acceleration

Doubling Every 2 Months

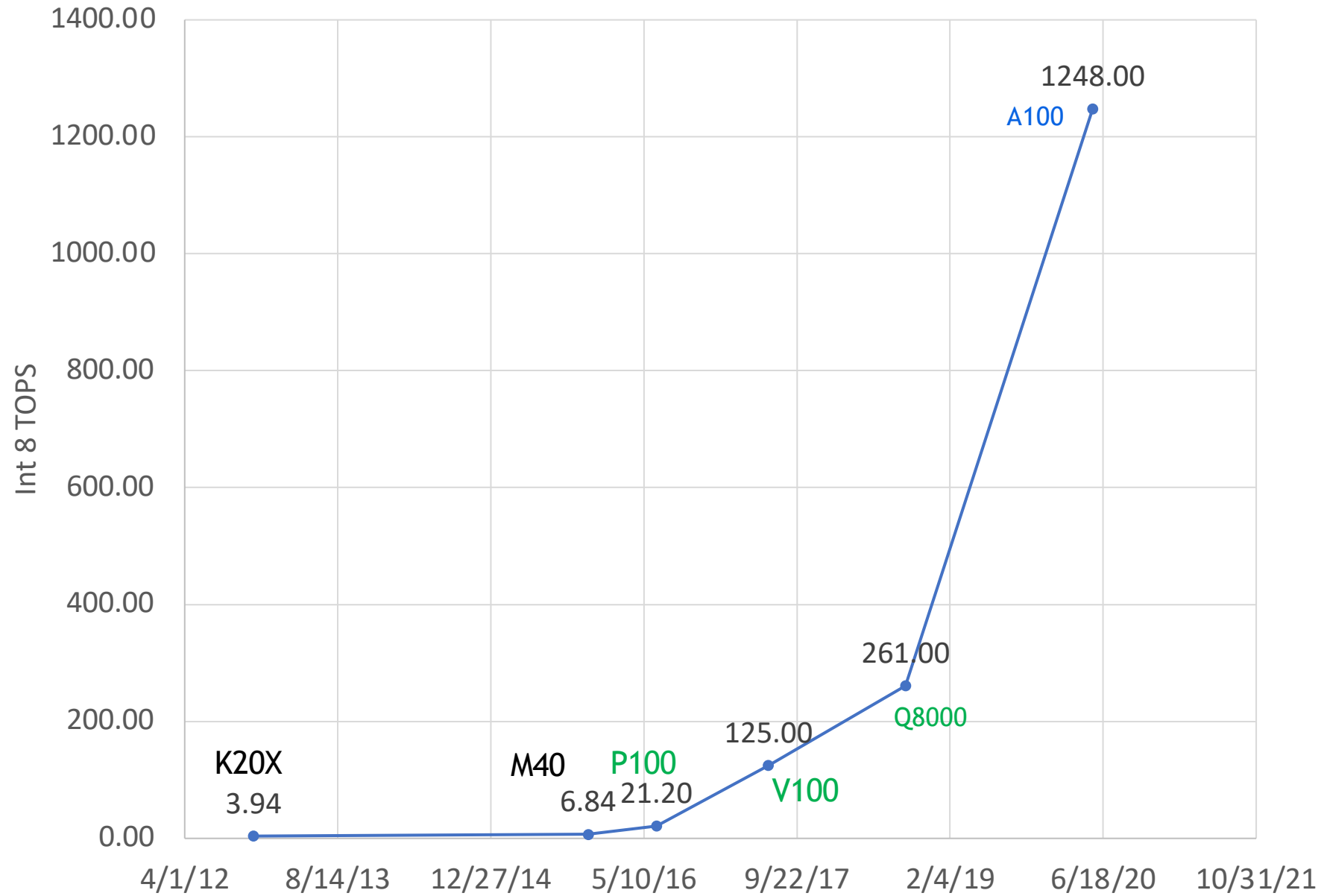


AI Interactions Per Day



Some History

Single-Chip Inference Performance - 317X in 8 years



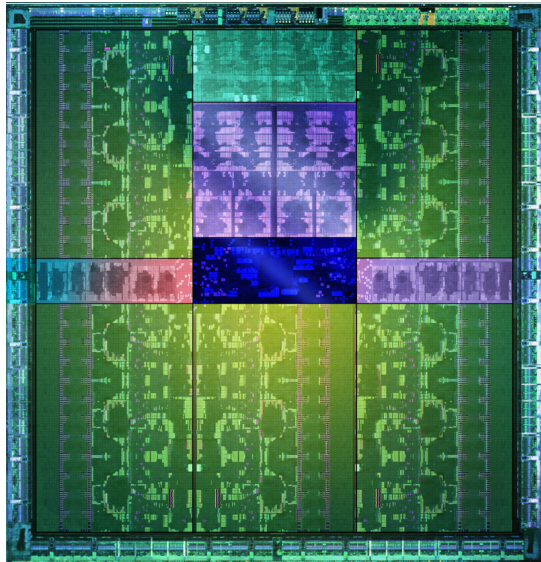
Kepler (2012)

3.95 TFLOPS (FP32)

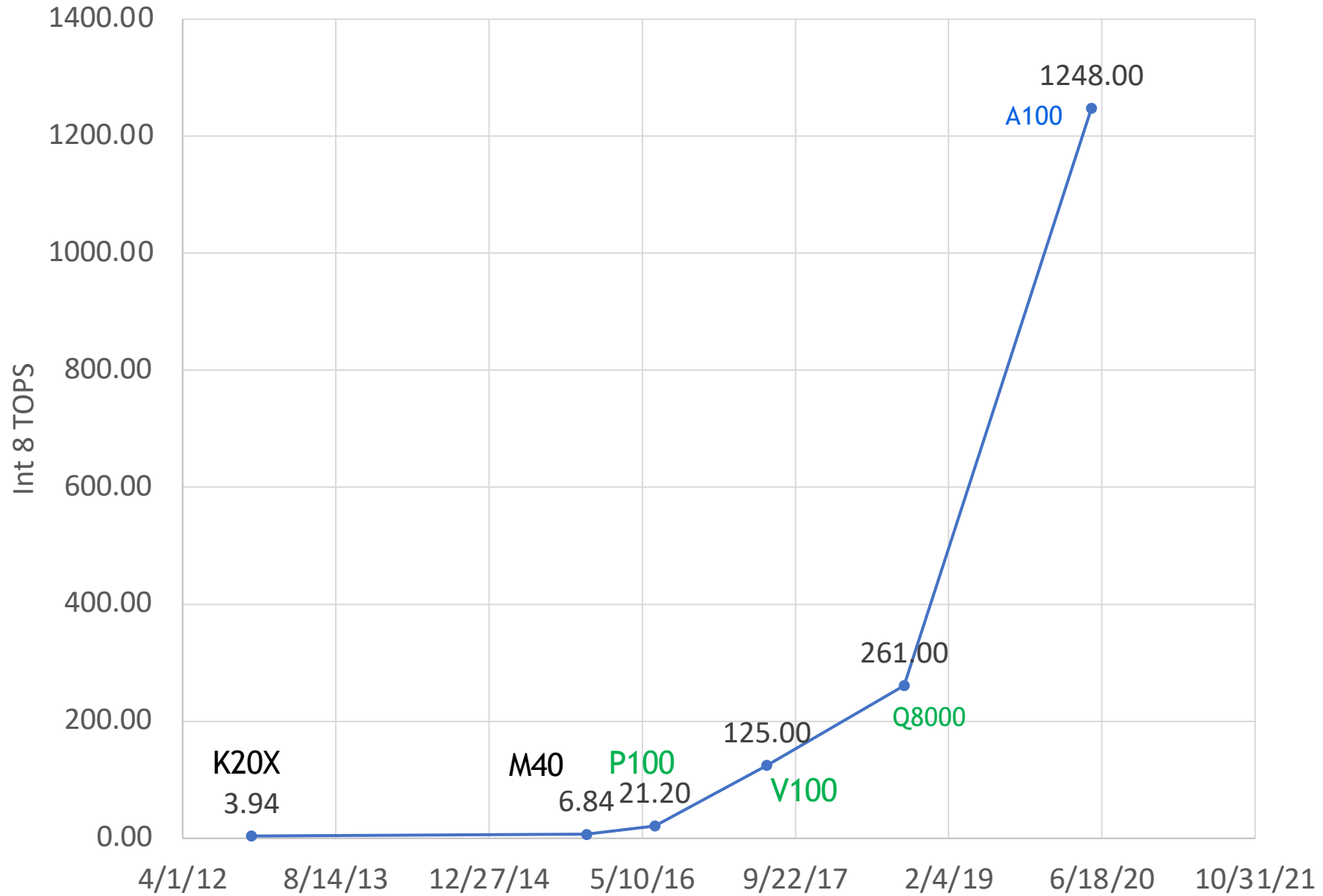
250 GB/s

300W

28nm



Single-Chip Inference Performance - 317X in 8 years



Pascal (2016)

10.6 TFLOPS (FP32)

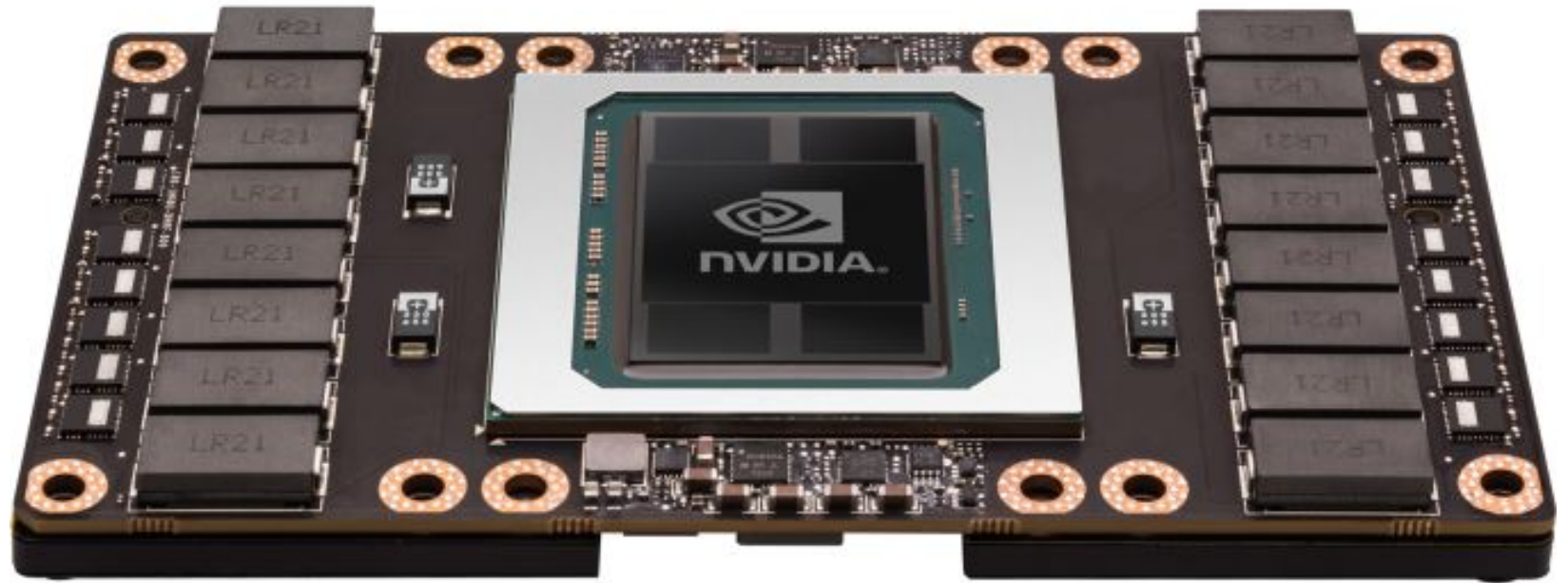
21.3 TFLOPS (FP16)

FDP4

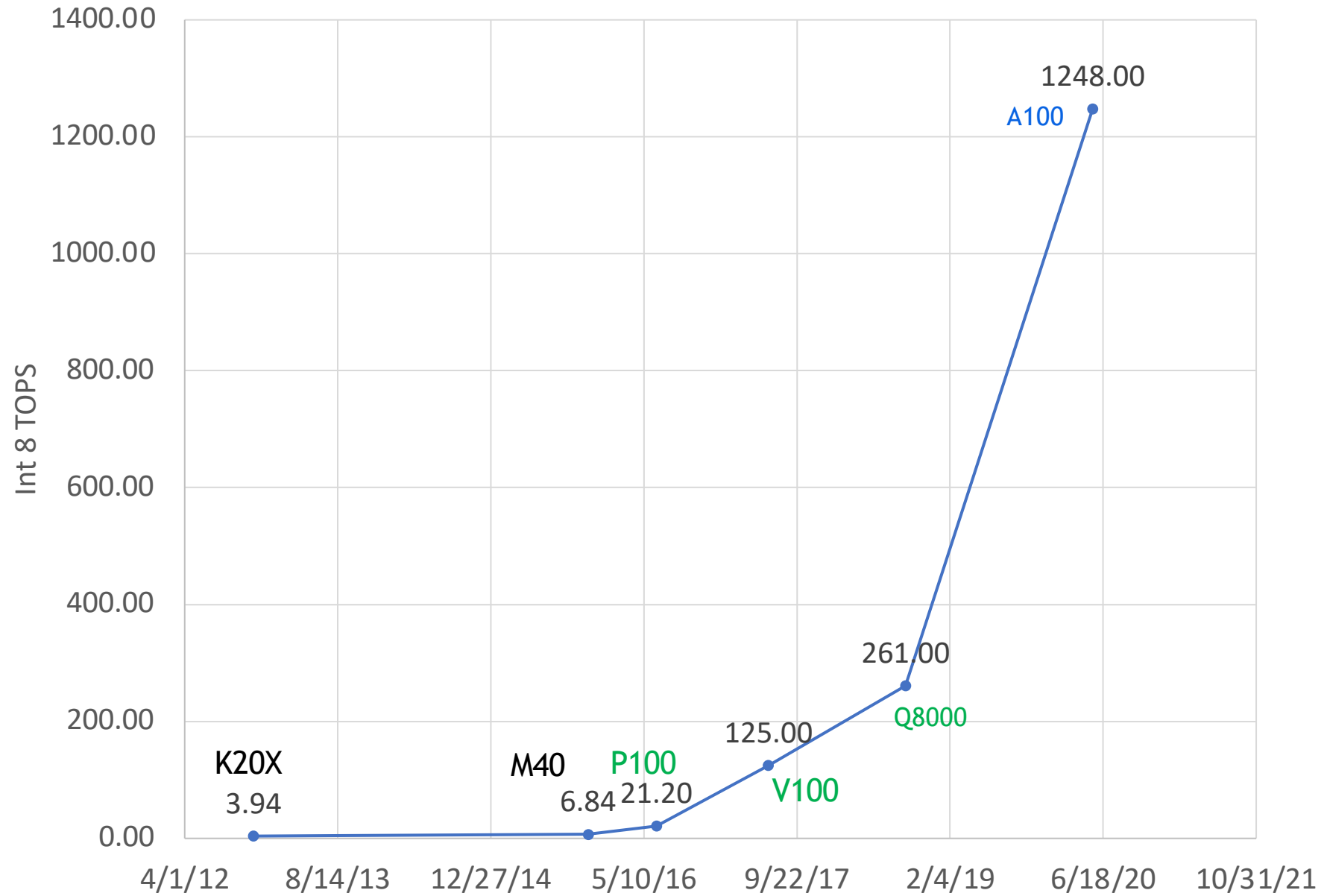
732 GB/s (HBM)

NVLink

300W



Single-Chip Inference Performance - 317X in 8 years

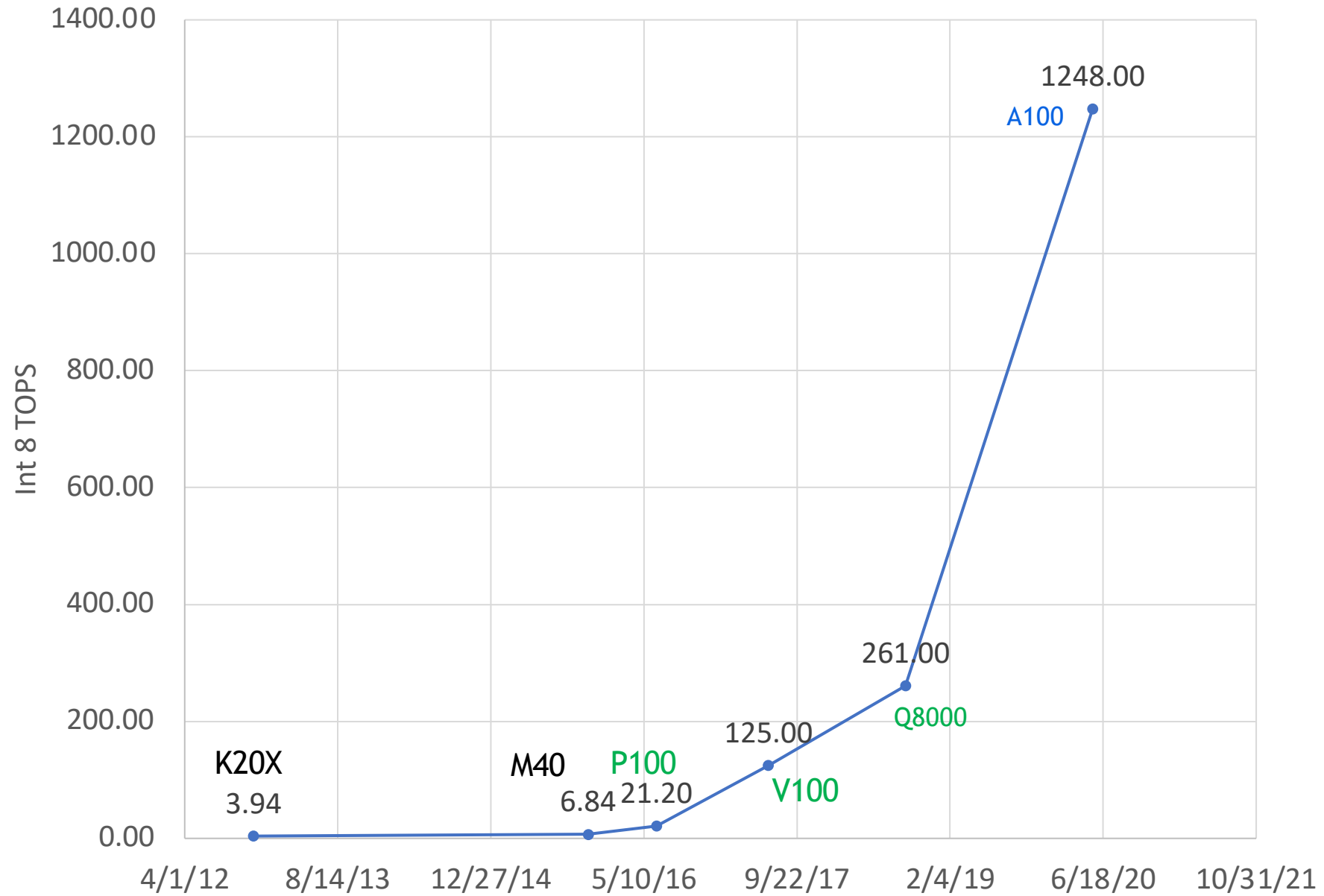


Volta (2017)

Tensor Cores!
15 TFLOPS (FP32)
125 TFLOPS (FP16)
HMMA
900 GB/s (HBM)
300 GB/s NVLink
300W

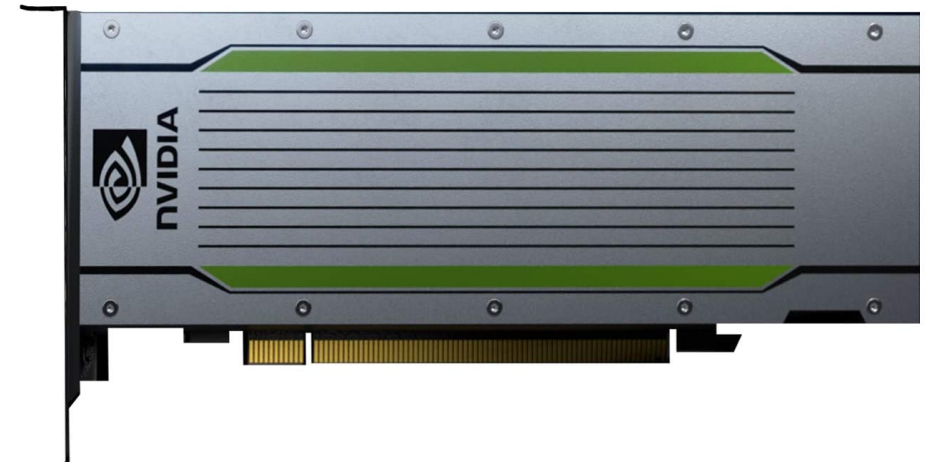
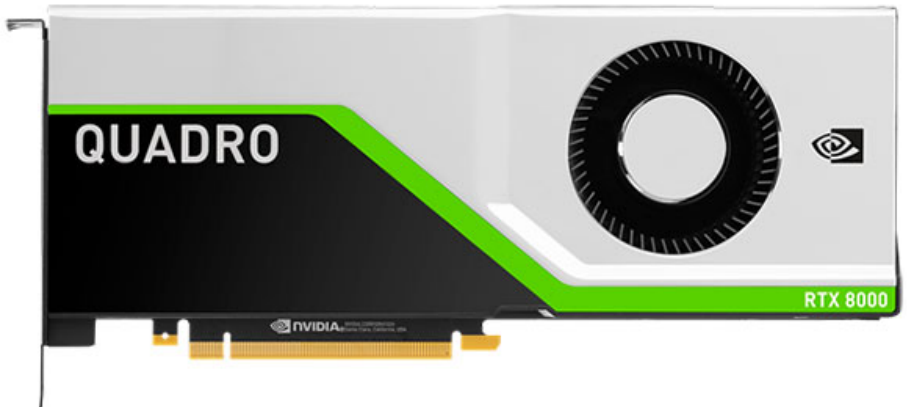
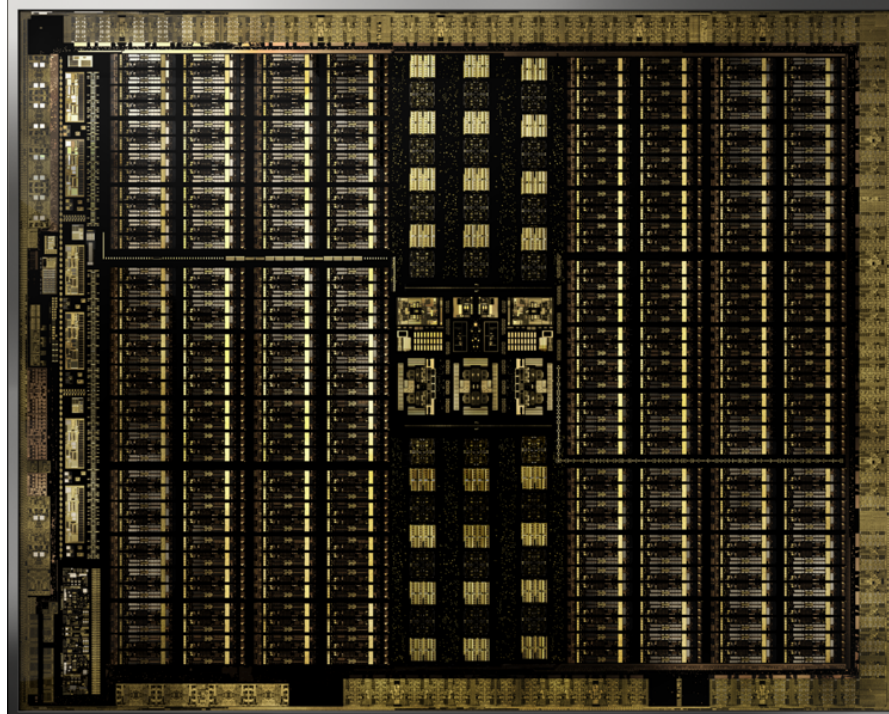


Single-Chip Inference Performance - 317X in 8 years

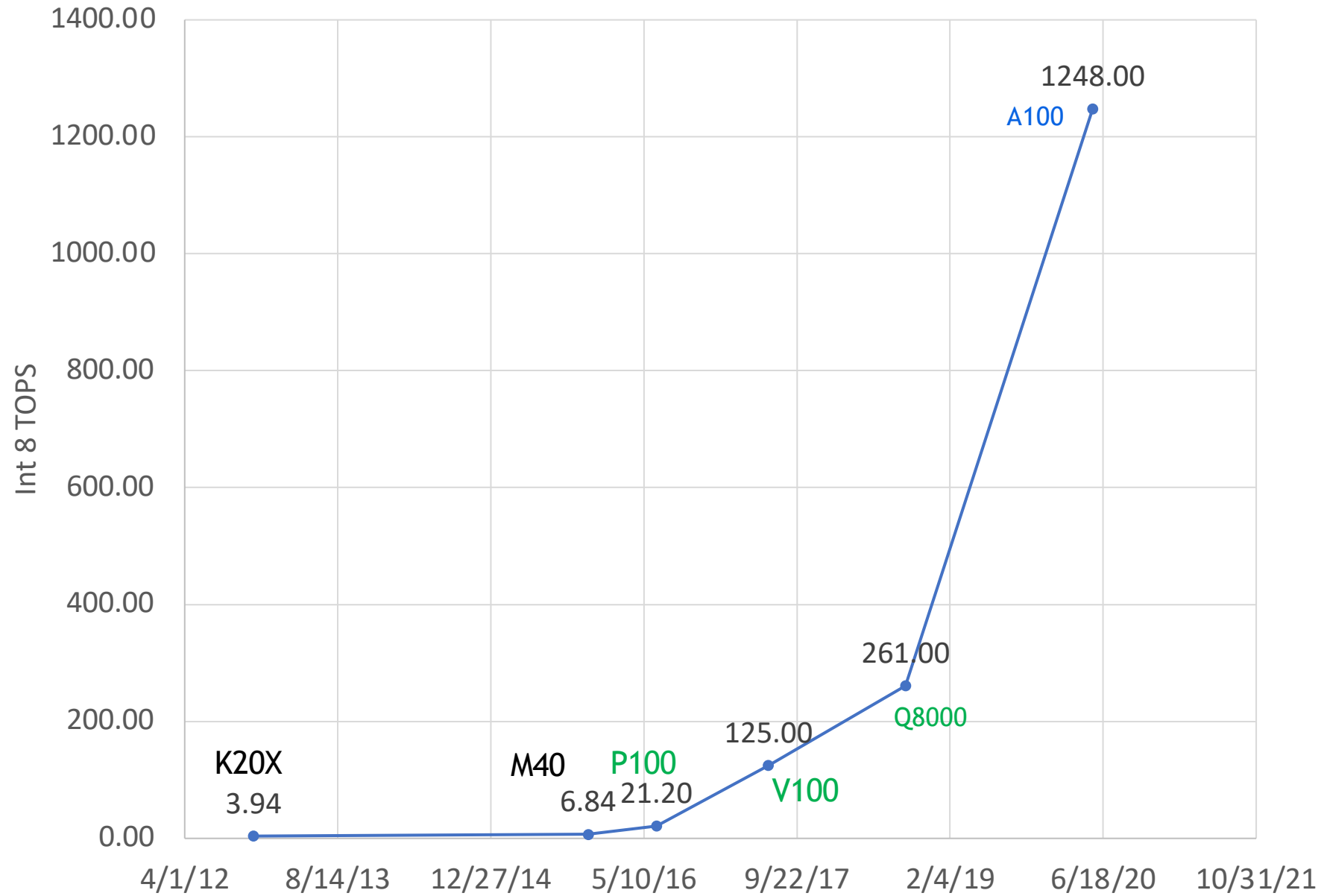


Turing (2018)

Integer Tensor Cores!
65 TFLOPS (FP32)
130 TFLOPS (FP16)
261 TOPs (Int8)
IMMA
672 GB/s (G5)
Ray Tracing!



Single-Chip Inference Performance - 317X in 8 years



Ampere (2020)

Sparsity!

BF16 & TF32!

156 / 312 TFLOPS (TF32) (dense/sparse)

312 / 624 TFLOPS (FP16 or BF16)

624 / 1,248 TOPS (Int 8)

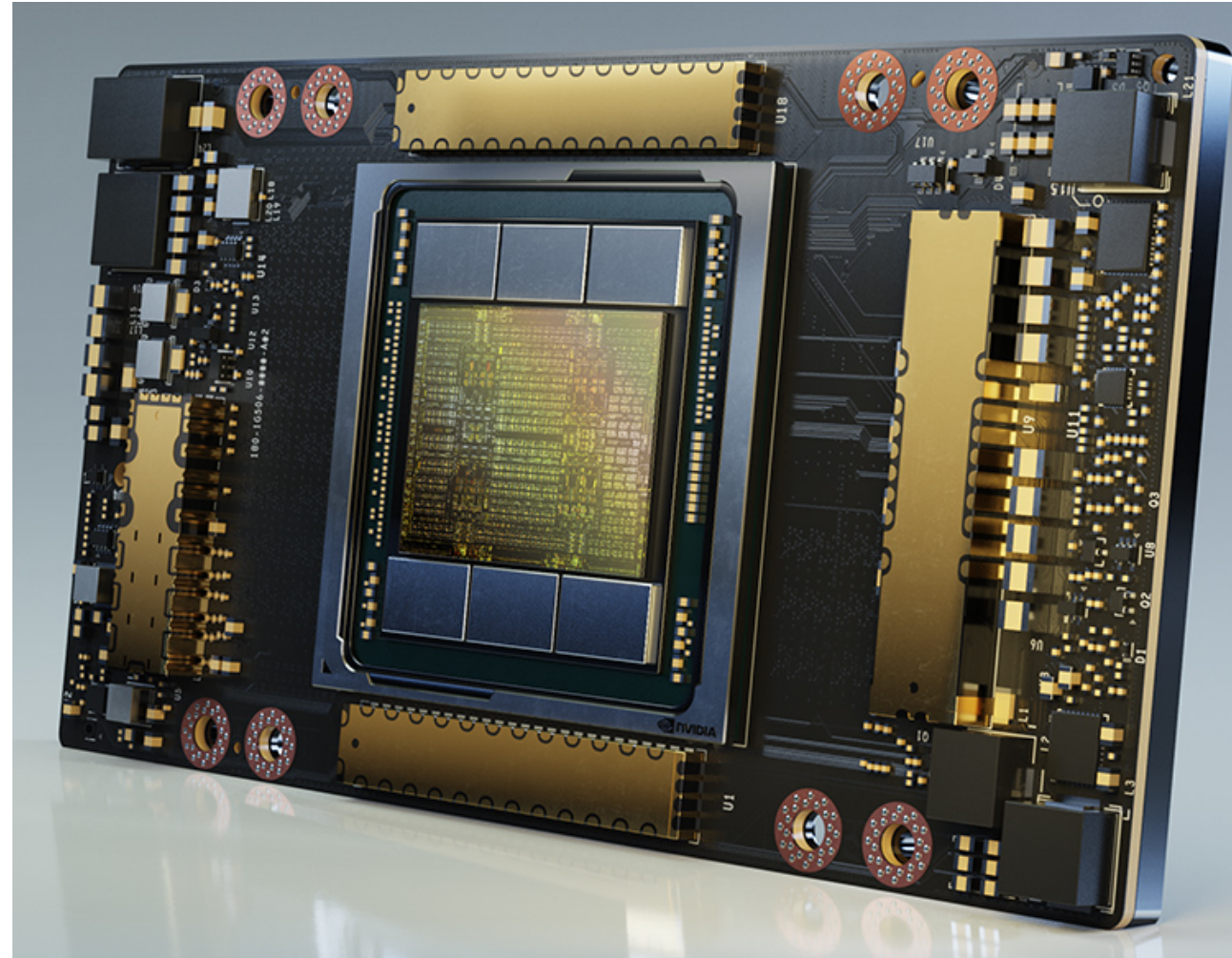
1,248 / 2,496 TOPS (Int 4)

2TB/s (HBM)

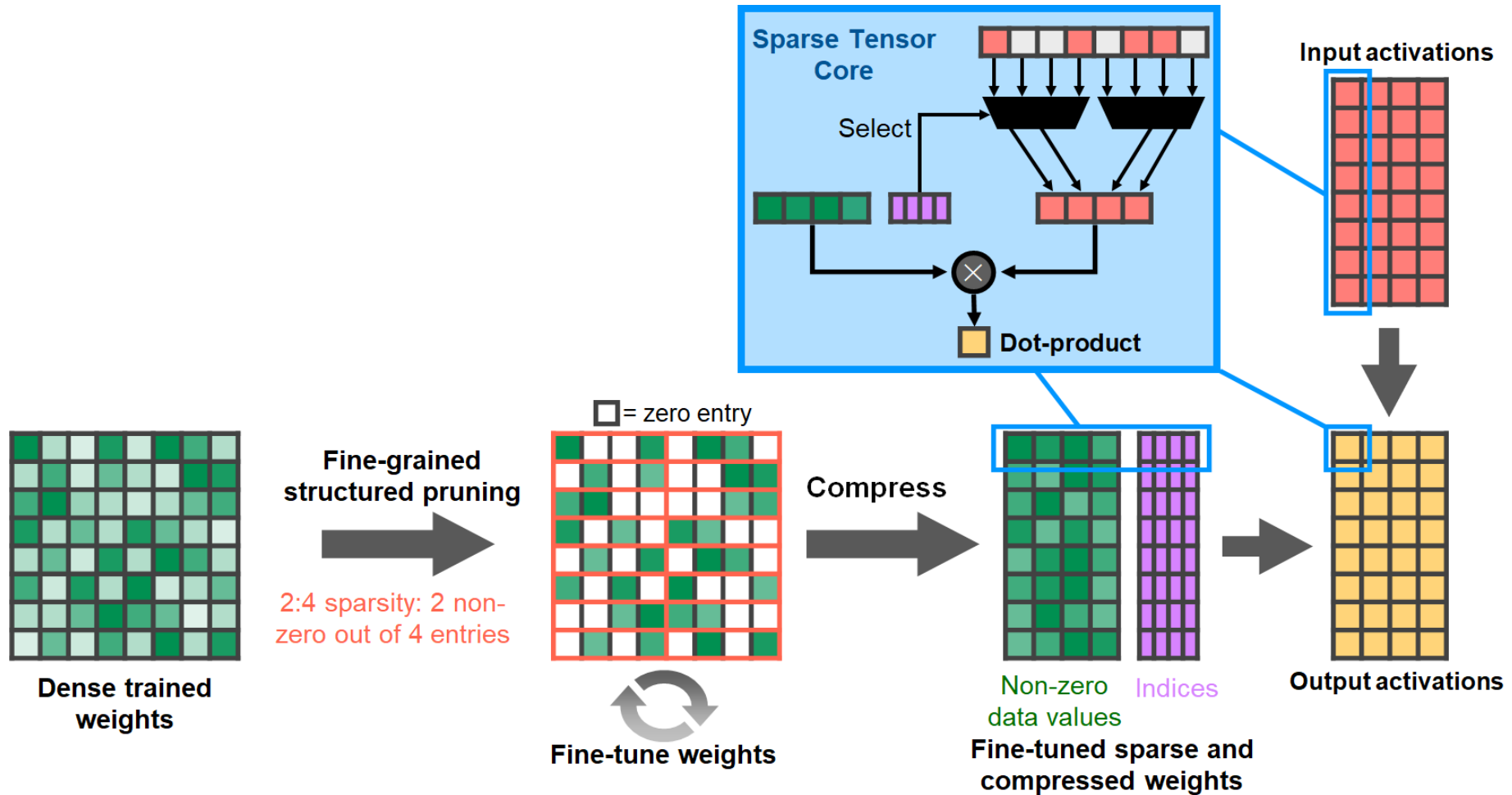
400W

3.12 TOPS/W (Int 8)

6.24 TOPS/W (Int 4)



Structured Sparsity



Gains from

Number representation

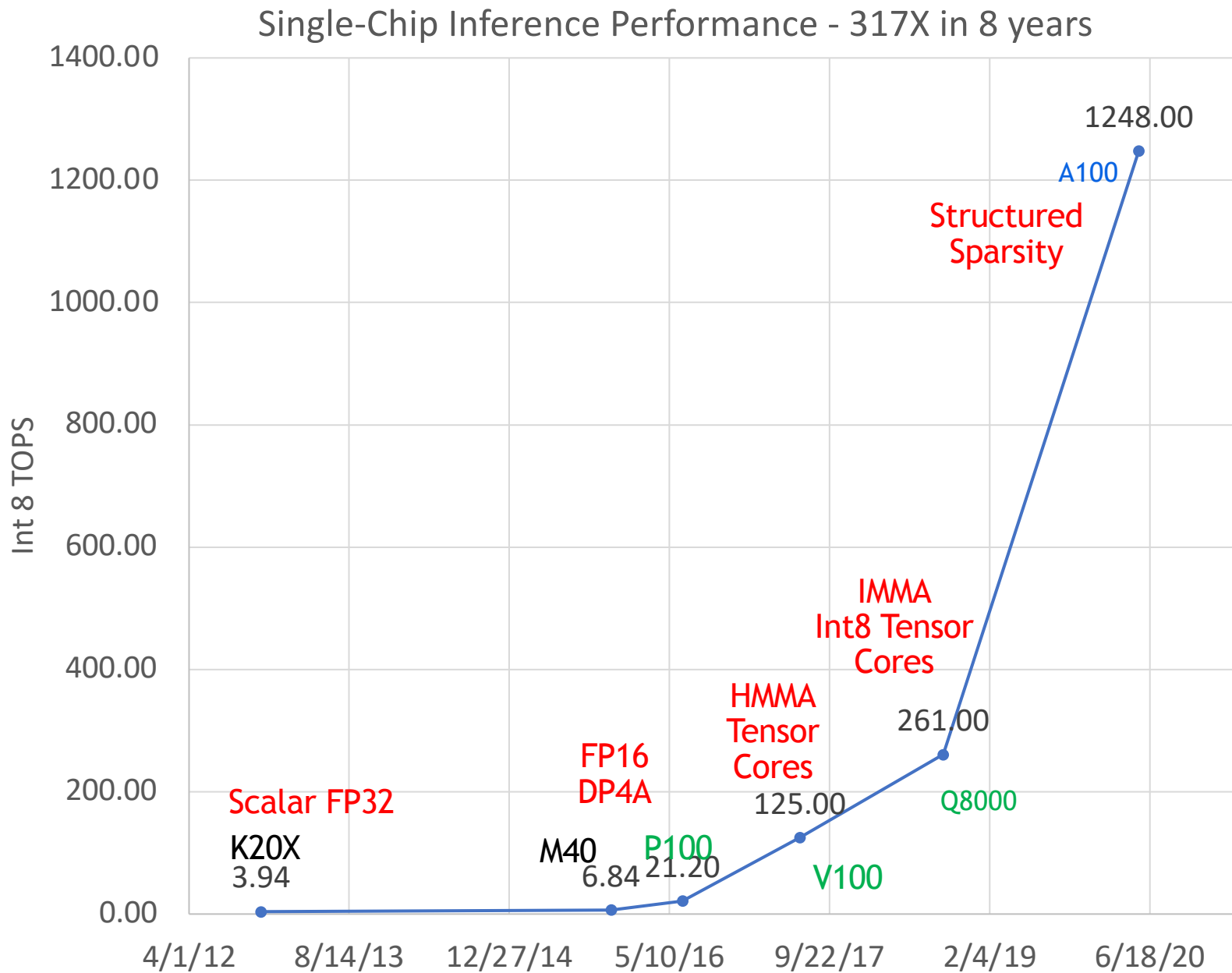
FP32, FP16, Int8
(TF32, BF16)

Complex instructions

DP4, HMMA, IMMA

Process

28nm, 16nm, 7nm



Specialized Instructions Amortize Overhead

Operation	Energy**	Overhead*
HFMA	1.5pJ	2000%
HDP4A	6.0pJ	500%
HMMA	110pJ	22%
IMMA	160pJ	16%

*Overhead is instruction fetch, decode, and operand fetch – 30pJ

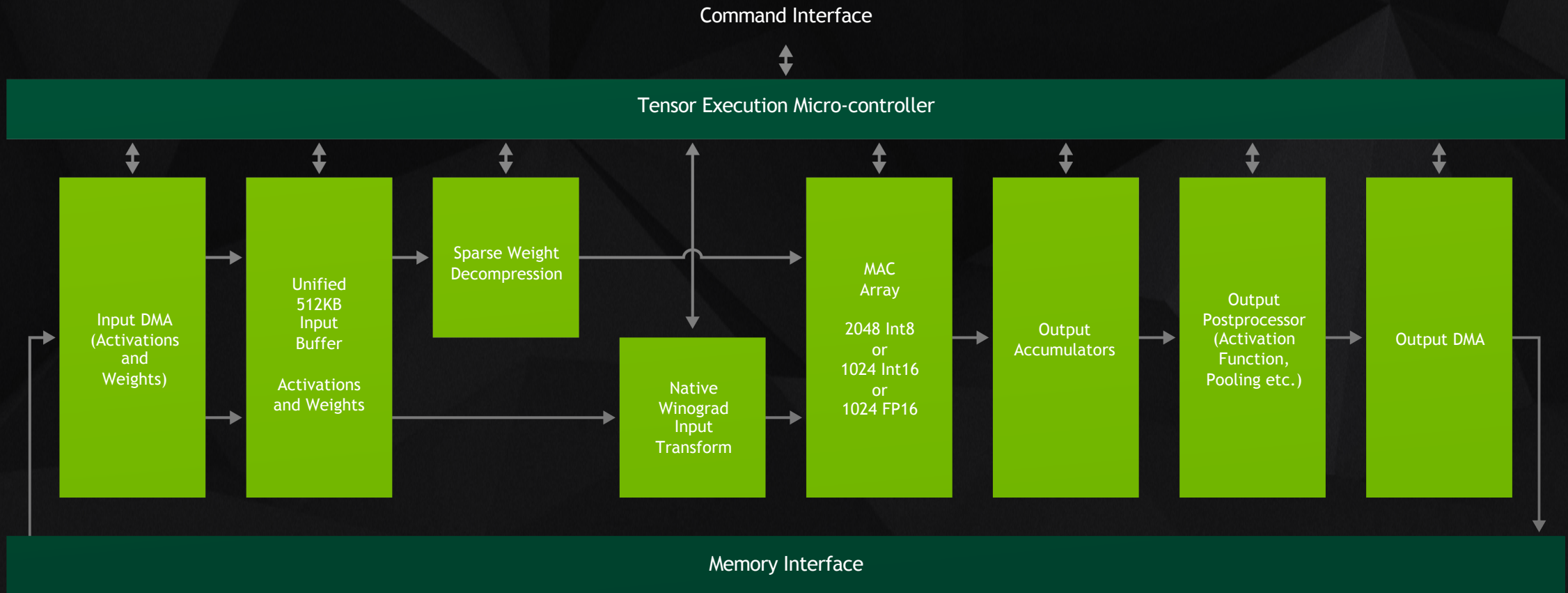
**Energy numbers from 45nm process

Accelerators

All have a matrix-multiply unit fed by a memory hierarchy.

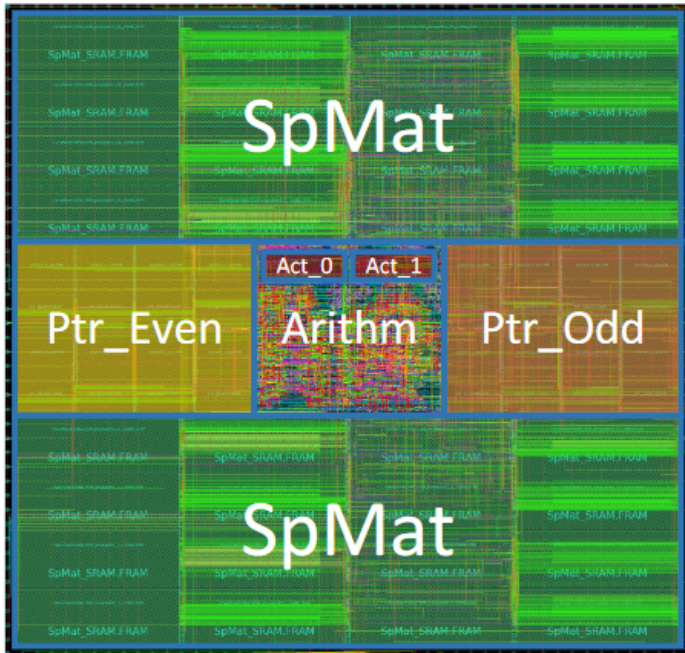
NVIDIA DLA

Sparsity
Compression
Data gating
Winograd



Open-sourced at nvdla.org

EIE (2016)

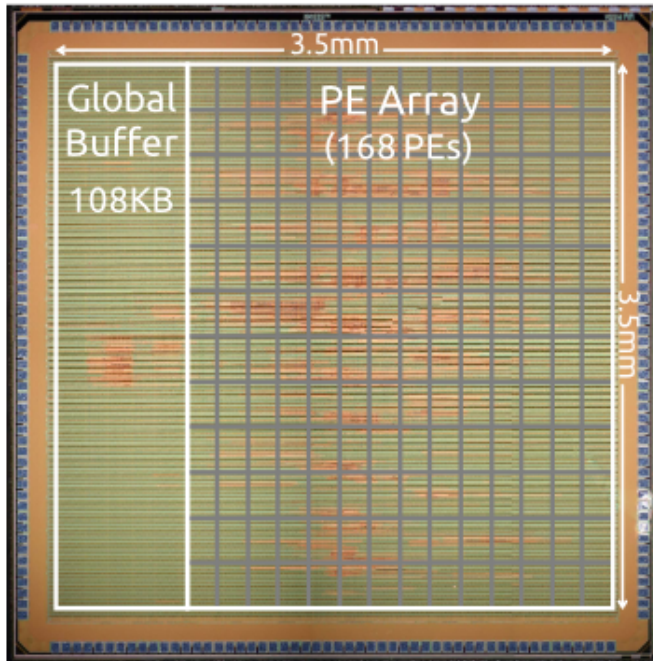


Efficient Inference Engine
for compressed
fully connected layers

Sparsity
Hardware CSR
Coding
Scalar Quantization



Eyeriss (2016)

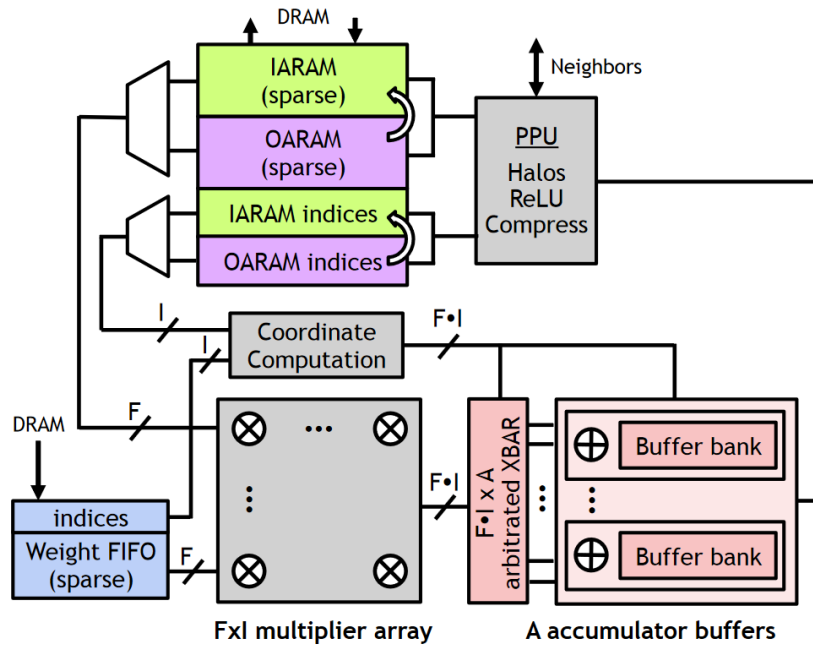


Tiling (dataflows)
Weight stationary
Row stationary

Spatial tiling with
optimized dataflows
for CNNs



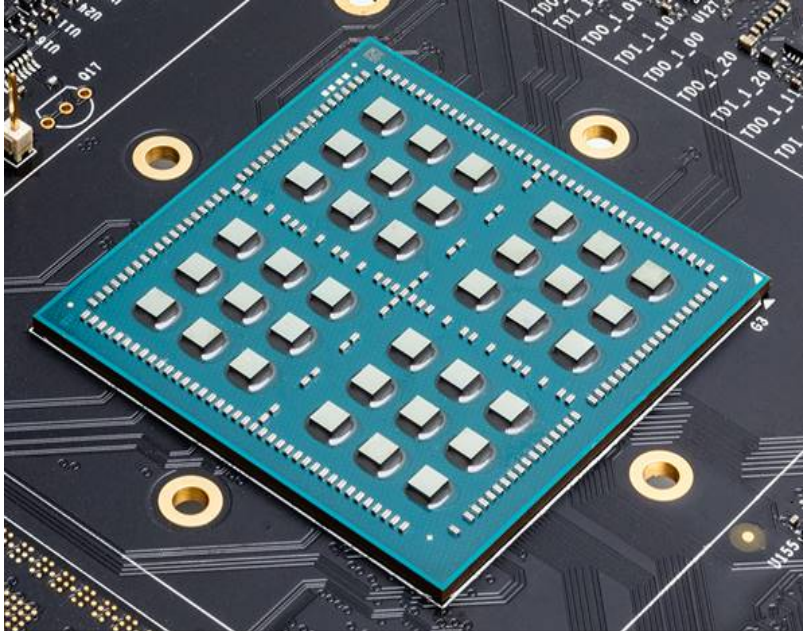
SCNN (2017)



Sparsity
Outer product
Scatter-Add

Optimized PE for
accelerating compressed
Sparse CNNs

SIMBA (RC18) (2019)



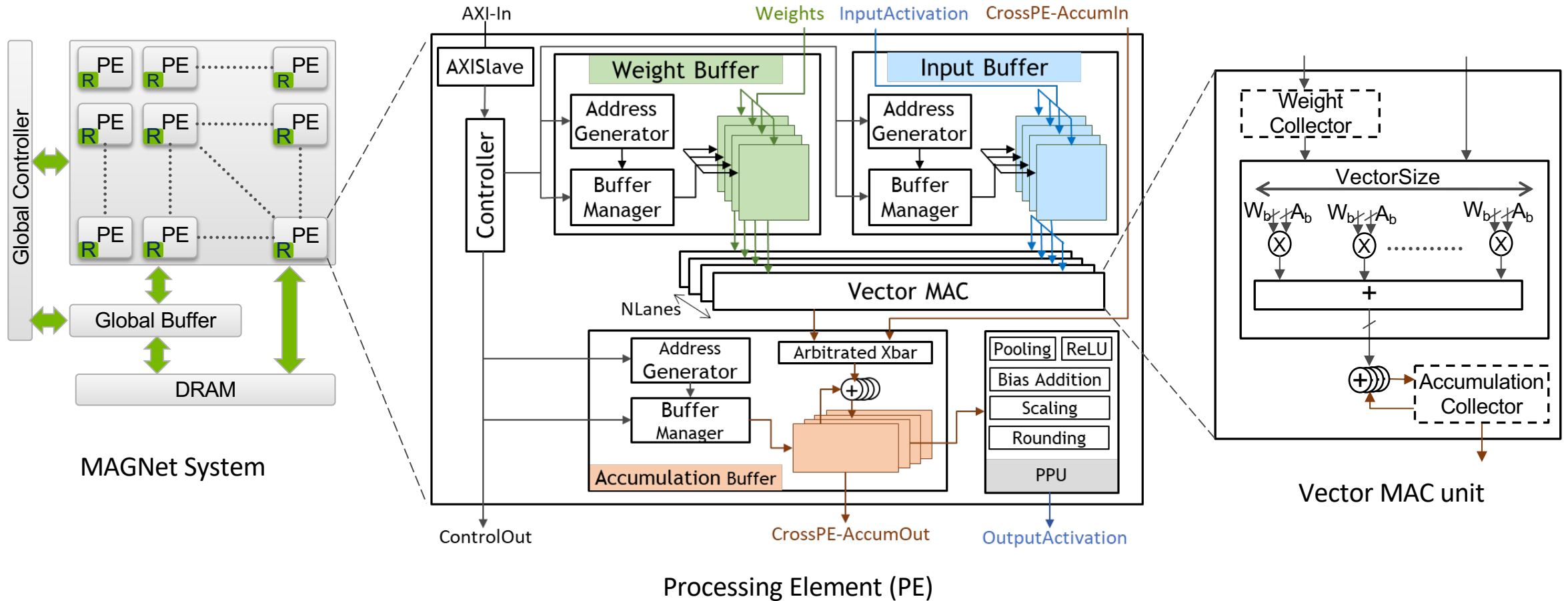
Scalable
MCM
Hierarchical Mesh

Tiled PEs in a scalable MCM
128 TOPS
0.11 pJ/Op



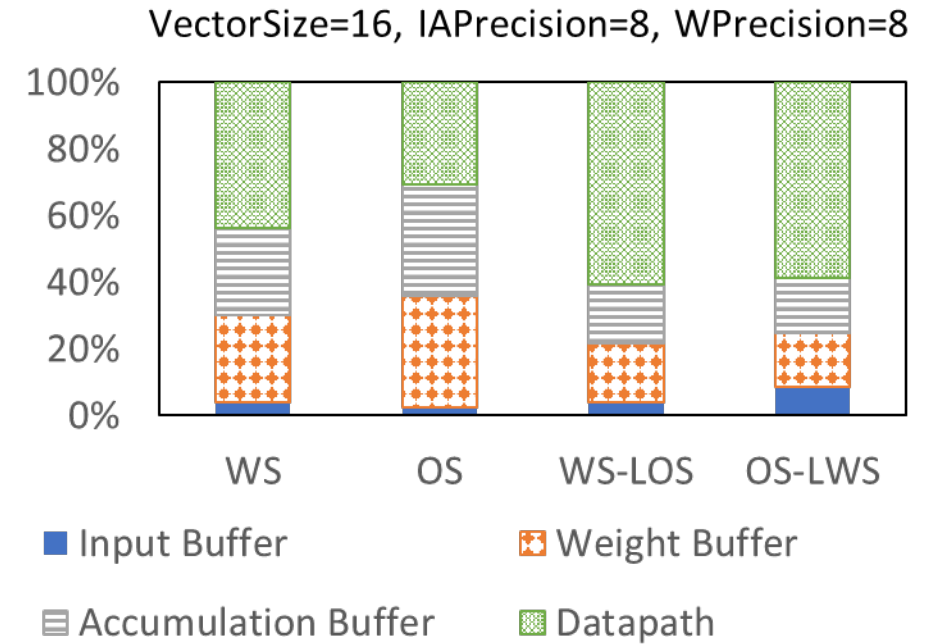
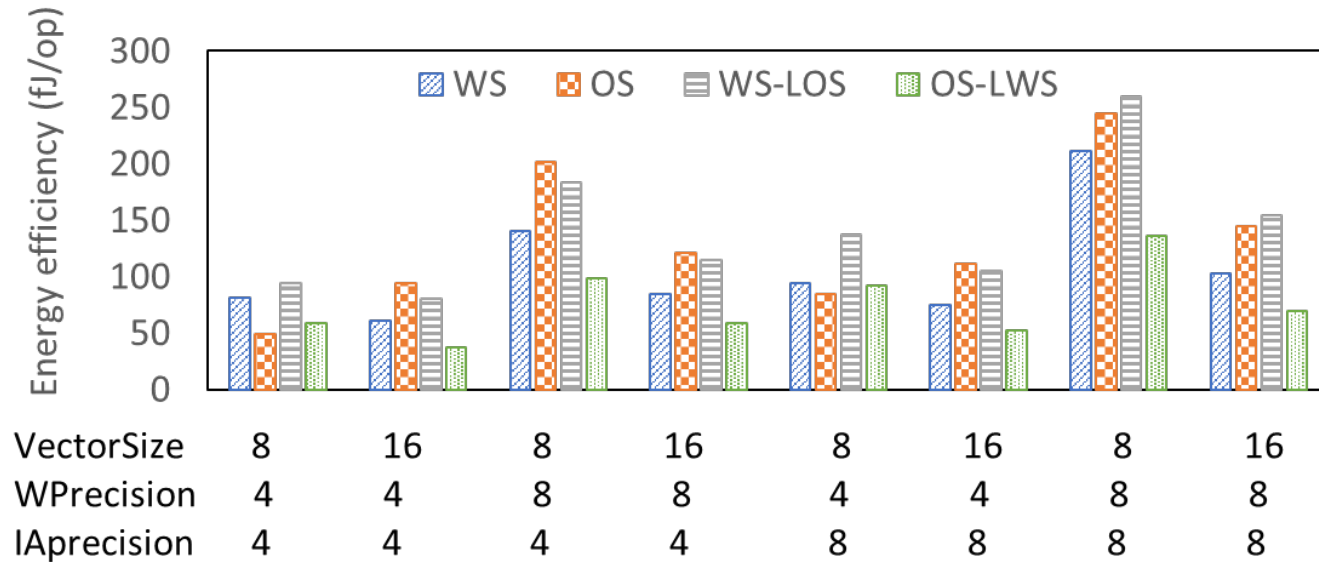
MAGNET

Configurable using synthesizable SystemC, HW generated using HLS tools



MAGNET RESULTS

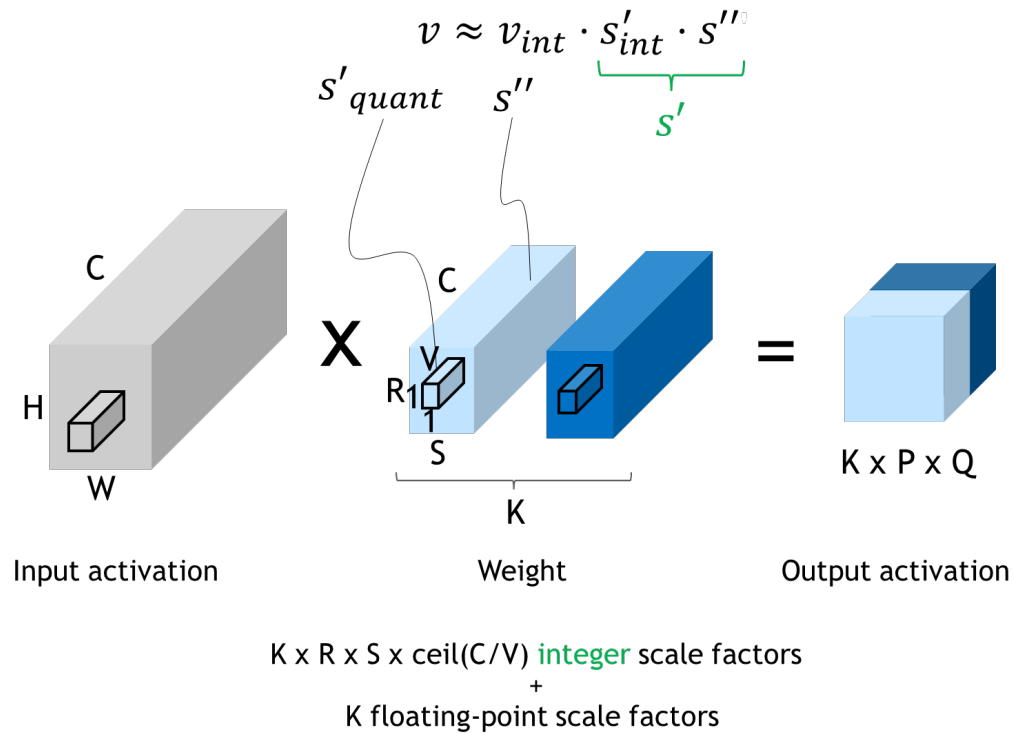
Design Space Exploration for ResNet-50



43% Energy Efficiency Improvement from Multi-Level Dataflows

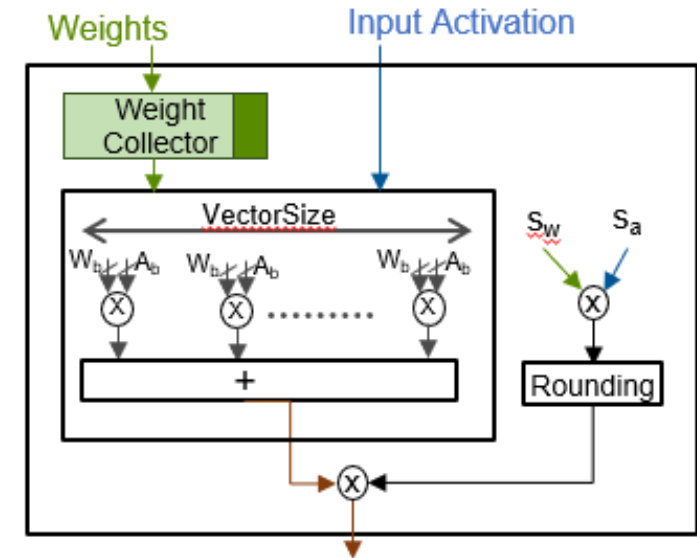
VS-Quant

Per-Vector Scaled Quantization for Low-Precision Inference



Fine-grained scale factors per vector

$$y_q(j) = \left(\sum_{i=0}^{vecsize-1} w_q(i) a_q(i) \right) s_w(j) s_a(j)$$

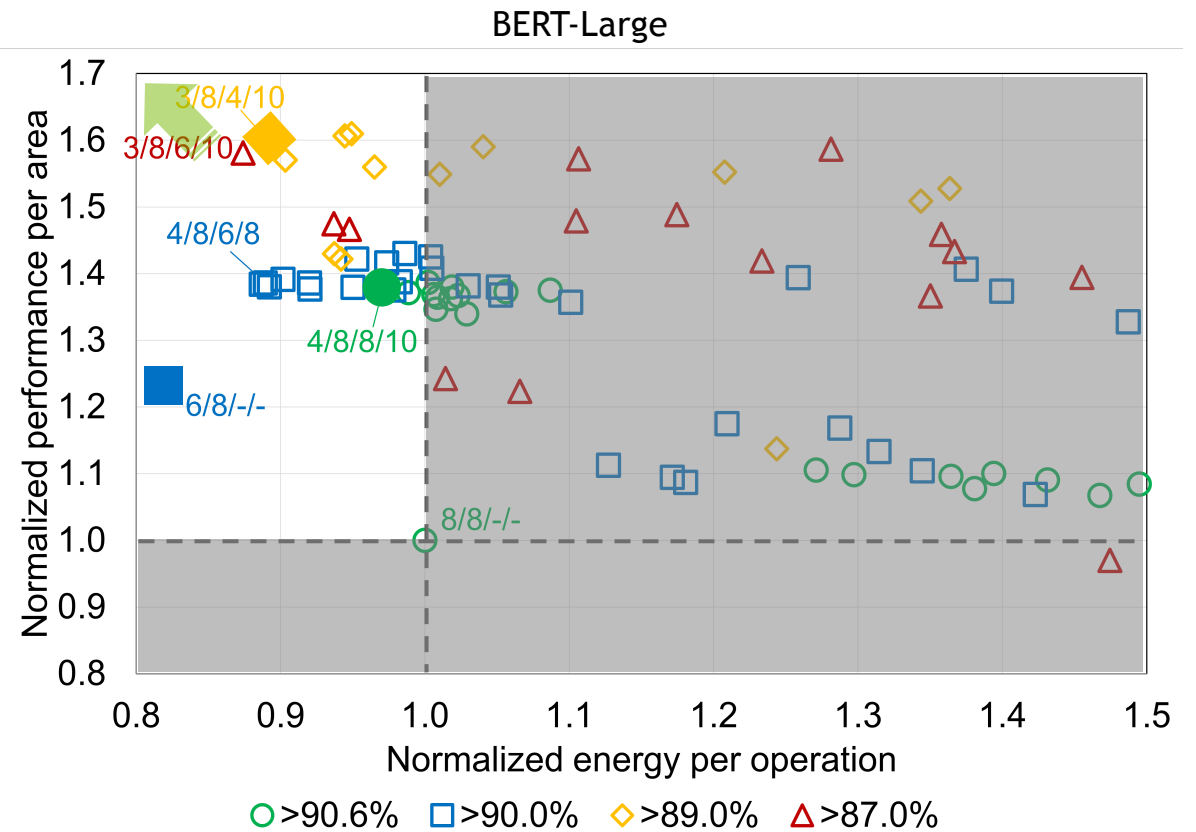
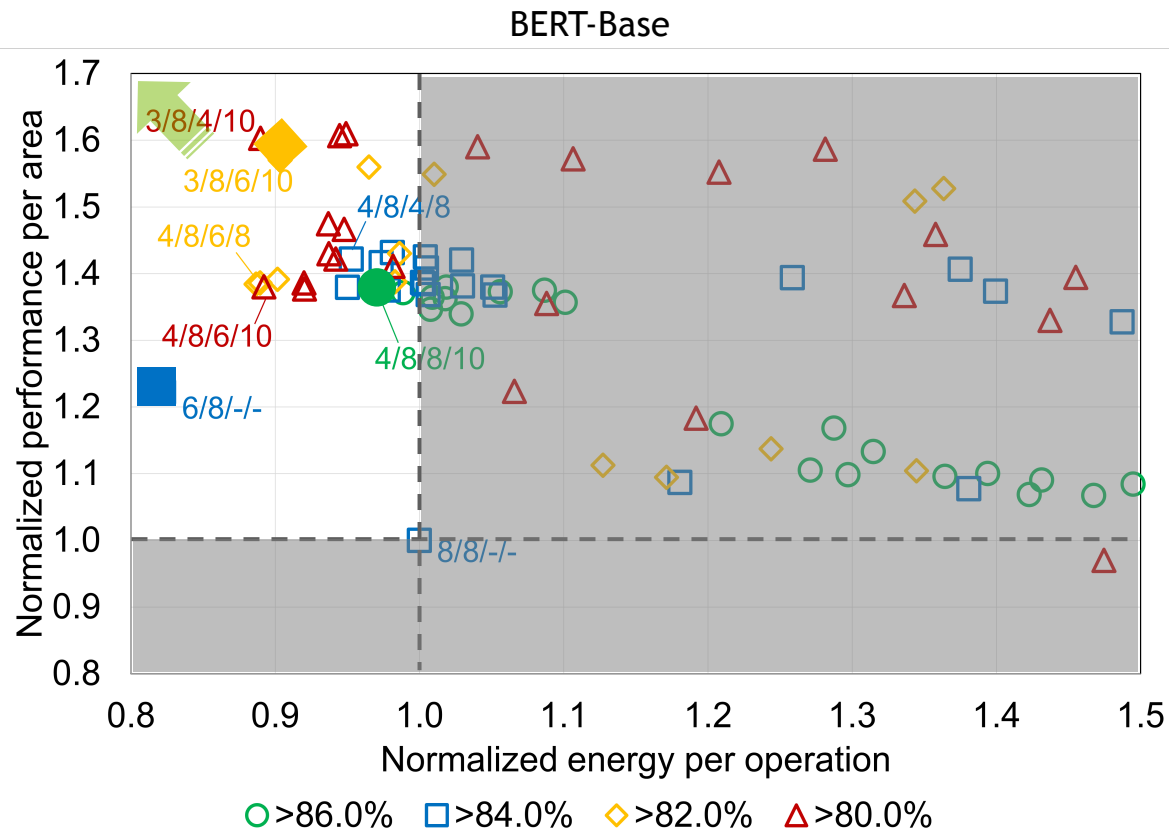


Modified vector MAC unit for VS-Quant

Works with either post-training quantization or quantization-aware retraining!

Energy, Area, and Accuracy Tradeoff

BERT-base and BERT-large on SQuAD



* Amount of scale rounding varies among design points

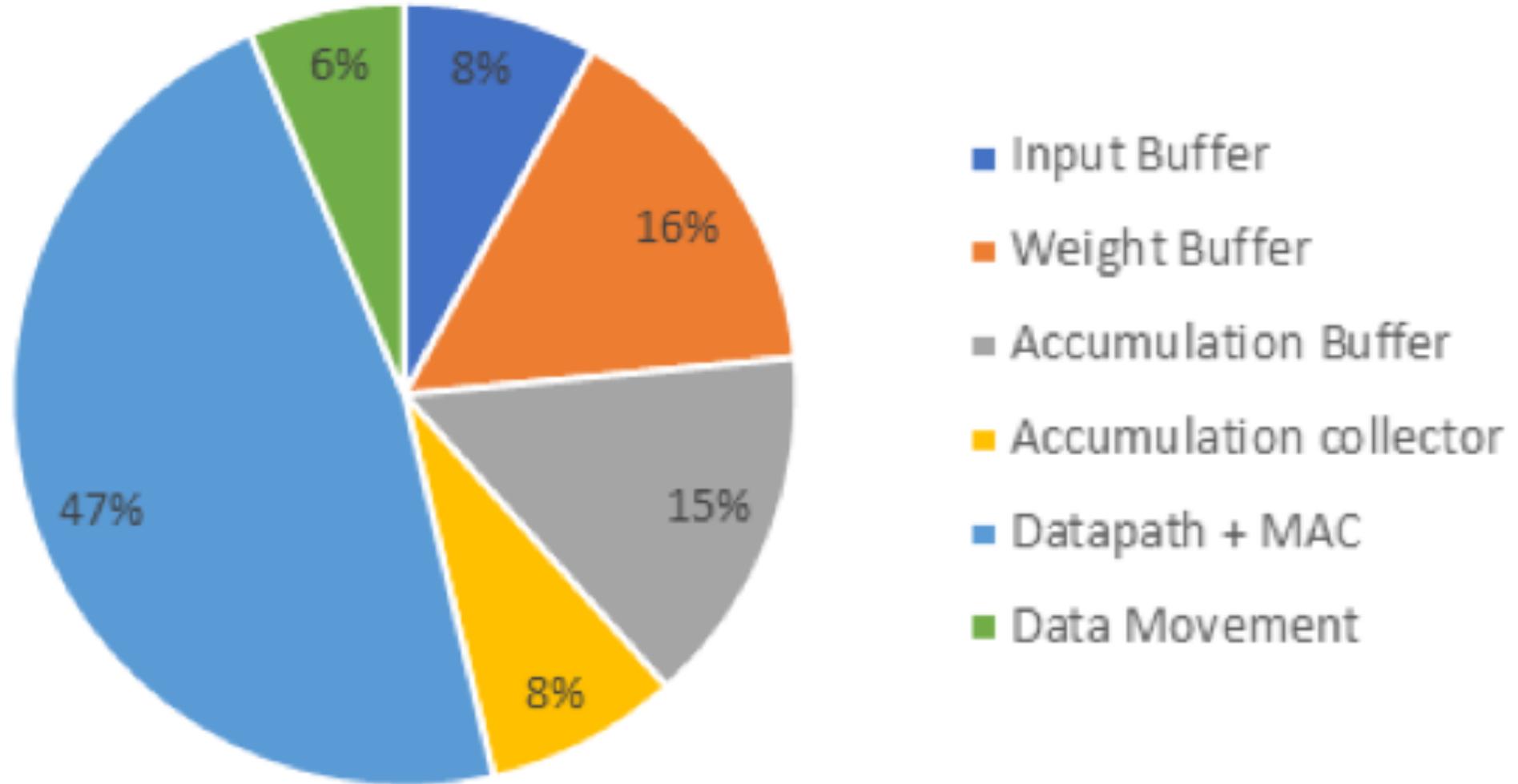
Weight Width / Activation Width / Weight Scale Width / Activation Scale Width
 “-” indicates per-channel scaling

Accelerators




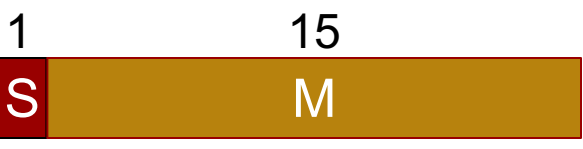

- Start with a matrix multiplier
- Tiling (dataflow)
 - Maximize re-use from memory hierarchy
 - Number of levels and size of each level are free variables
- Sparsity
 - Compression (memory and communication)
 - Data gating
 - Sparse computation
- Number representation
 - Coding (makes math expensive)
 - Scaling (put the bits where they do the most good)
 - Scale by the vector

Logarithmic Numbers

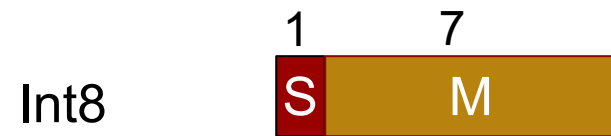
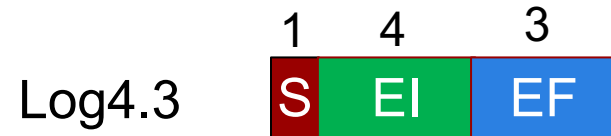
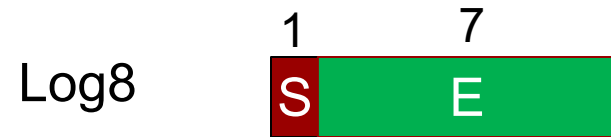
Energy Breakdown



Number Representation

		Range	Accuracy
FP32		$10^{-38} - 10^{38}$.000006%
FP16		$6 \times 10^{-5} - 6 \times 10^4$.05%
Int32		$0 - 2 \times 10^9$	33%
Int16		$0 - 6 \times 10^4$	33%
Int8		$0 - 127$	33%

Logarithmic Numbers



Range

Accuracy

$10^{-38} - 10^{38}$

33%

$6 \times 10^{-5} - 6 \times 10^5$

4%

0-127

33%

Optimum but expensive

$$v = -1^s 2^{ei.ef}$$



Dynamic Range 10^5

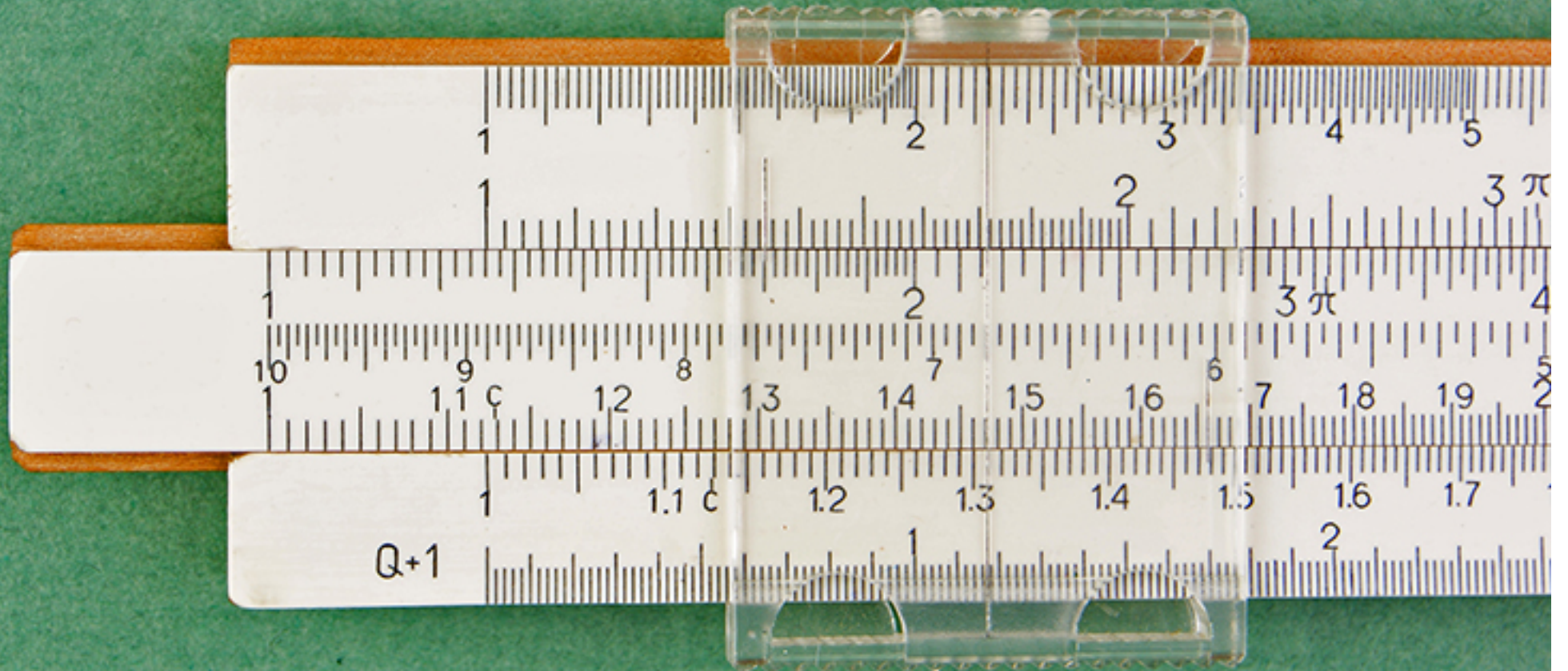
WC Accuracy 4%

Vs Int8 – DR 10^2

WC Accuracy 33%

Can apply offset to EI to represent any range of 16 integers, e.g., -8 to 7 (scaling)

Numbers near zero need special treatment



Computer Multiplication and Division Using Binary Logarithms*

JOHN N. MITCHELL, JR.,[†] ASSOCIATE, IRE

Summary—A method of computer multiplication and division is proposed which uses binary logarithms. The logarithm of a binary number may be determined approximately from the number itself by simple shifting and counting. A simple add or subtract and shift operation is all that is required to multiply or divide. Since the logarithms used are approximate there can be errors in the result. An error analysis is given and a means of reducing the error for the multiply operation is shown.

I. INTRODUCTION

MULTIPLICATION and division operations in computers are usually accomplished by a series of additions and subtractions, and shifts. Con-

be binary logarithms (to the base two). Since $\log_{10} N$ is usually written $\log N$ and $\log_e N$ is written $\ln N$, to avoid ambiguity and the necessity of writing the subscript a similar notation will be adopted in this paper to imply $\log_2 N$:

$$\lg N \equiv \log_2 N.$$

A table of binary logarithms is shown in Fig. 1, and the familiar logarithmic curve is plotted in Fig. 2. Suppose the points where $\lg N$ is an integer are connected by straight lines. The dashed lines in Fig. 2 describe the

Convolutional Neural Networks using Logarithmic Data Representation

Daisuke Miyashita

Stanford University, Stanford, CA 94305 USA
Toshiba, Kawasaki, Japan

DAISUKEM@STANFORD.EDU

Edward H. Lee

Stanford University, Stanford, CA 94305 USA

EDHLEE@STANFORD.EDU

Boris Murmann

Stanford University, Stanford, CA 94305 USA

MURMANN@STANFORD.EDU

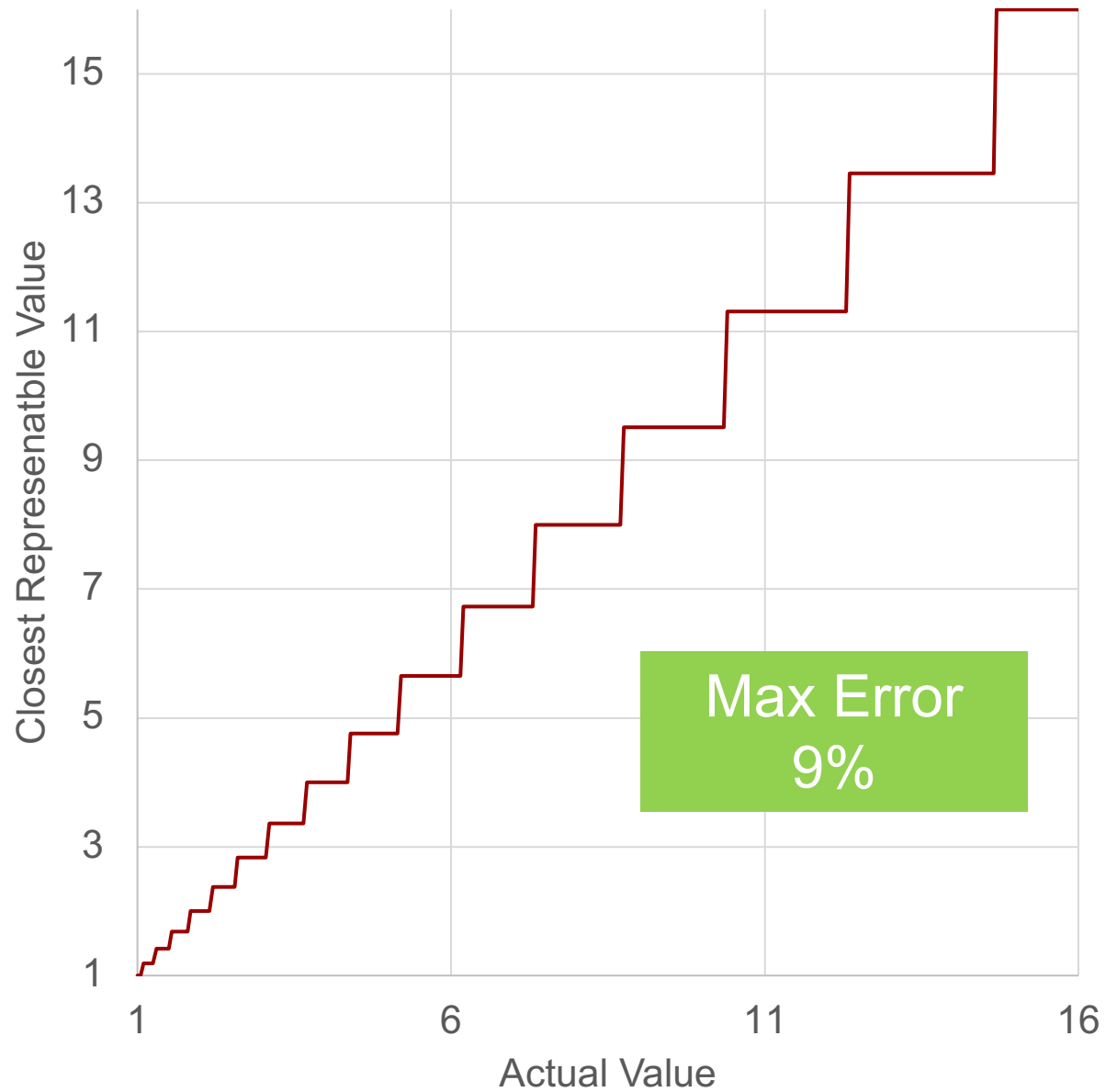
Abstract

Recent advances in convolutional neural networks have considered model complexity and

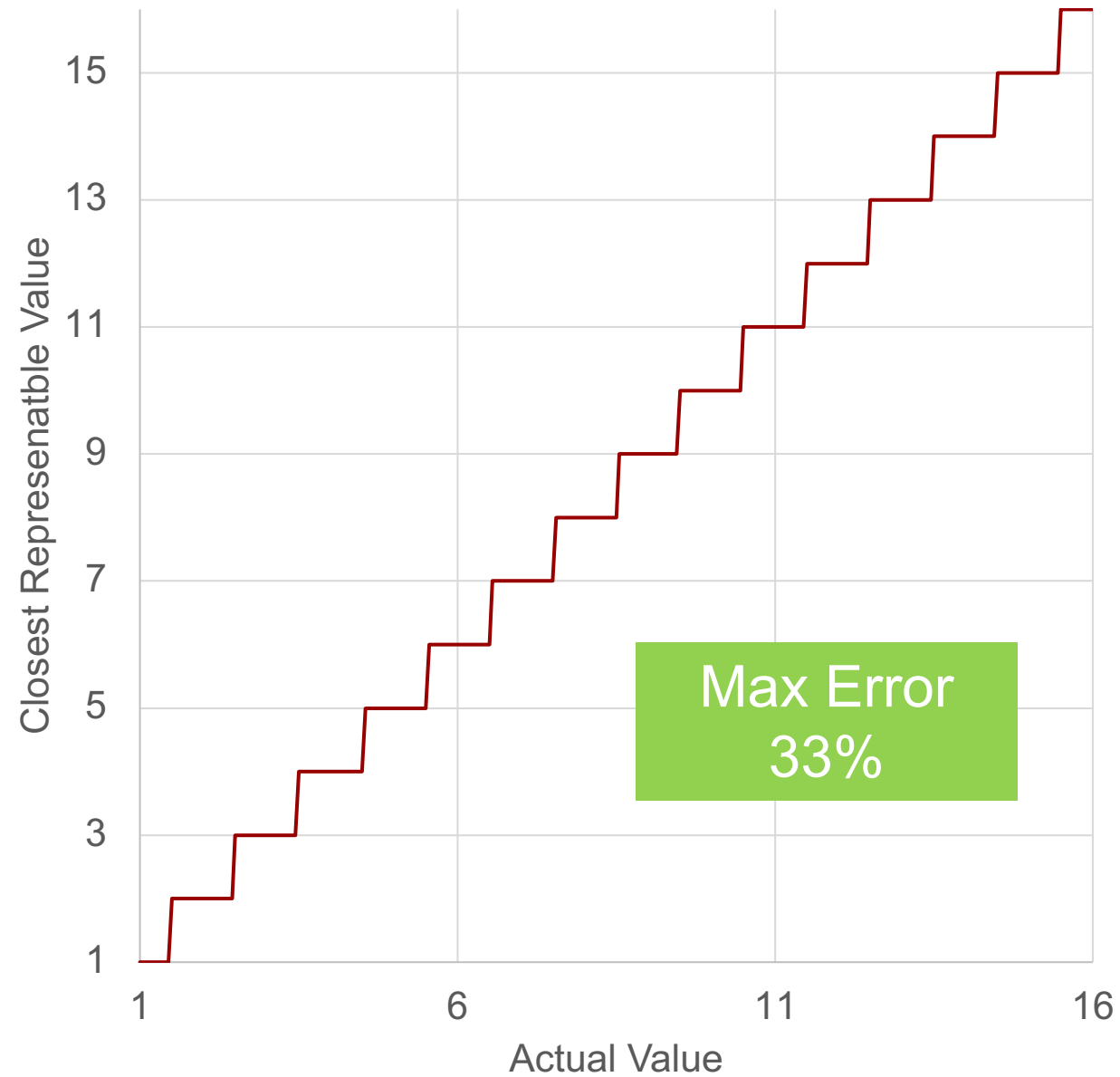
(Krizhevsky et al., 2012; Simonyan & Zisserman, 2014; He et al., 2015) but have steadily grown in computational complexity. For example, the Deep Residual Learning (He

Integer log only – $2^{a.0}$

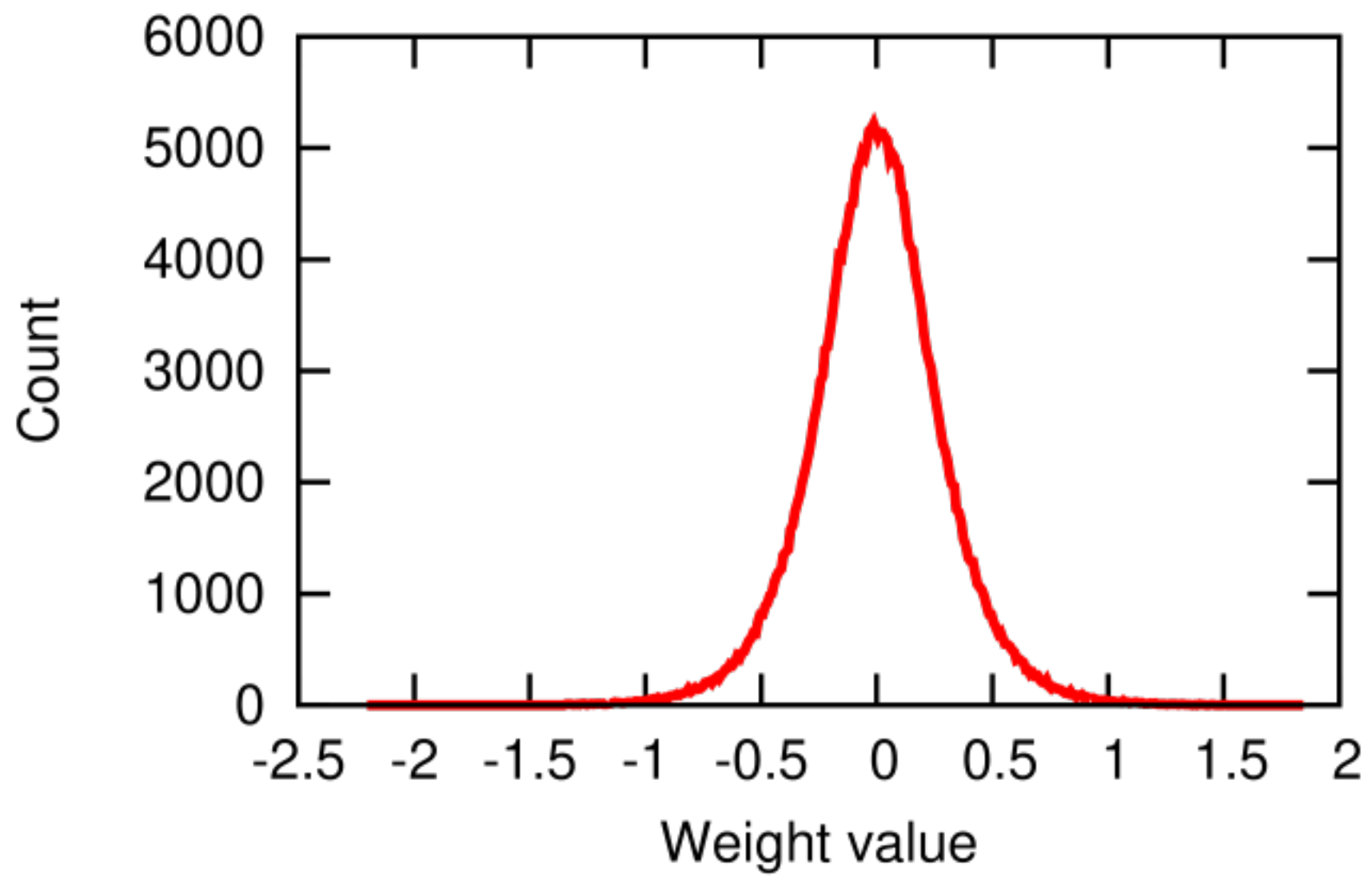
4-bit Log Representation (L2.2)



4-bit Integer Representation (Int4)



Weight distribution of layer 1 (PTB small)



Why Log

- Lower error where it matters
- Same accuracy with fewer bits
- Multiplies become adds

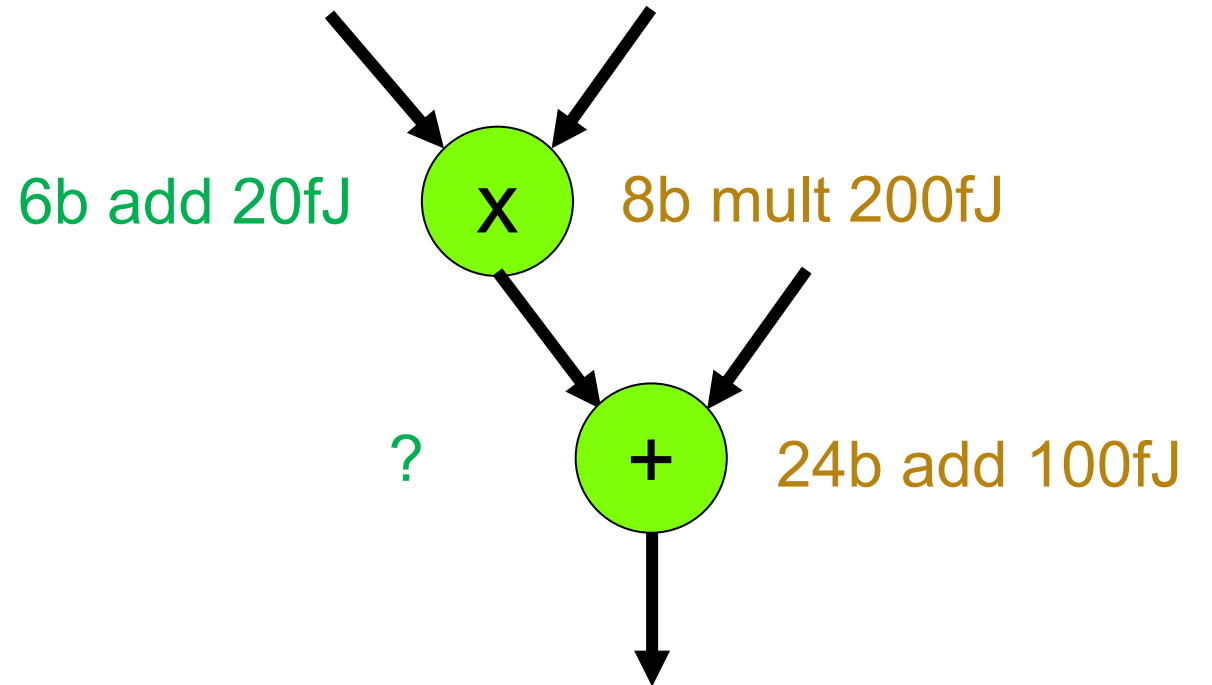
Multiply energy reduced by 10x

What about the add?

Log adds are expensive
Shift a constant and add

Log6

Int8



(19) **United States**

(12) **Patent Application Publication**
Dally et al.

(10) **Pub. No.: US 2021/0056446 A1**
(43) **Pub. Date: Feb. 25, 2021**

(54) **INFERENCE ACCELERATOR USING LOGARITHMIC-BASED ARITHMETIC**

(52) **U.S. Cl.**
CPC **G06N 5/04** (2013.01); **G06N 20/00** (2019.01)

(71) Applicant: **NVIDIA Corporation**, Santa Clara, CA (US)

(57) **ABSTRACT**

(72) Inventors: **William James Dally**, Incline Village, NV (US); **Rangharajan Venkatesan**, San Jose, CA (US); **Brucek Kurdo Khailany**, Austin, TX (US)

Neural networks, in many cases, include convolution layers that are configured to perform many convolution operations that require multiplication and addition operations. Compared with performing multiplication on integer, fixed-point, or floating-point format values, performing multiplication on logarithmic format values is straightforward and energy efficient as the exponents are simply added. However, performing addition on logarithmic format values is more complex. Conventionally, addition is performed by converting the logarithmic format values to integers, computing the sum, and then converting the sum back into the logarithmic format. Instead, logarithmic format values may be added by decomposing the exponents into separate quotient and remainder components, sorting the quotient components based on the remainder components, summing the sorted quotient components using an asynchronous accumulator to produce partial sums, and multiplying the partial sums by the remainder components to produce a sum. The sum may then be converted back into the logarithmic format.

(21) Appl. No.: **16/750,823**

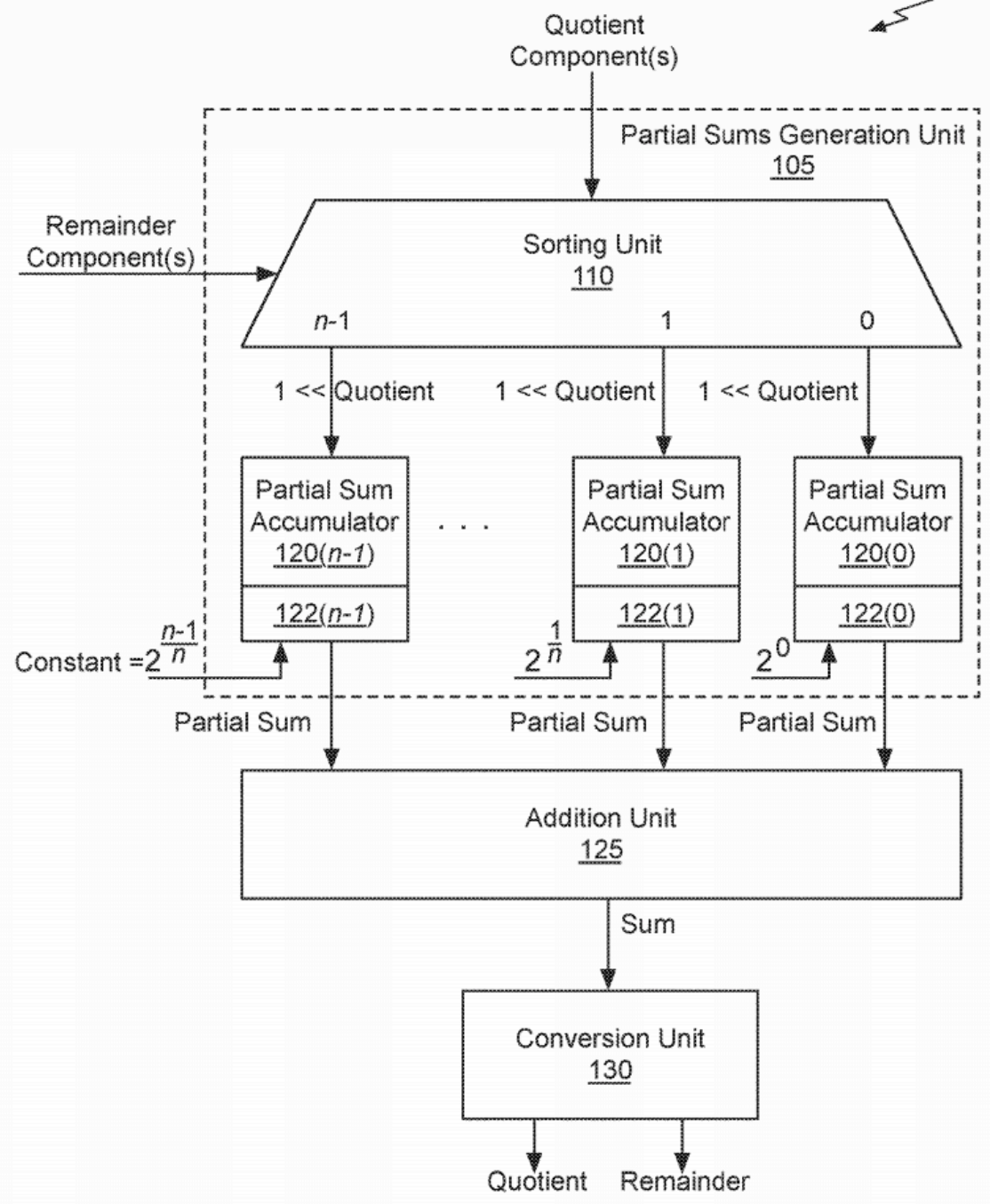
(22) Filed: **Jan. 23, 2020**

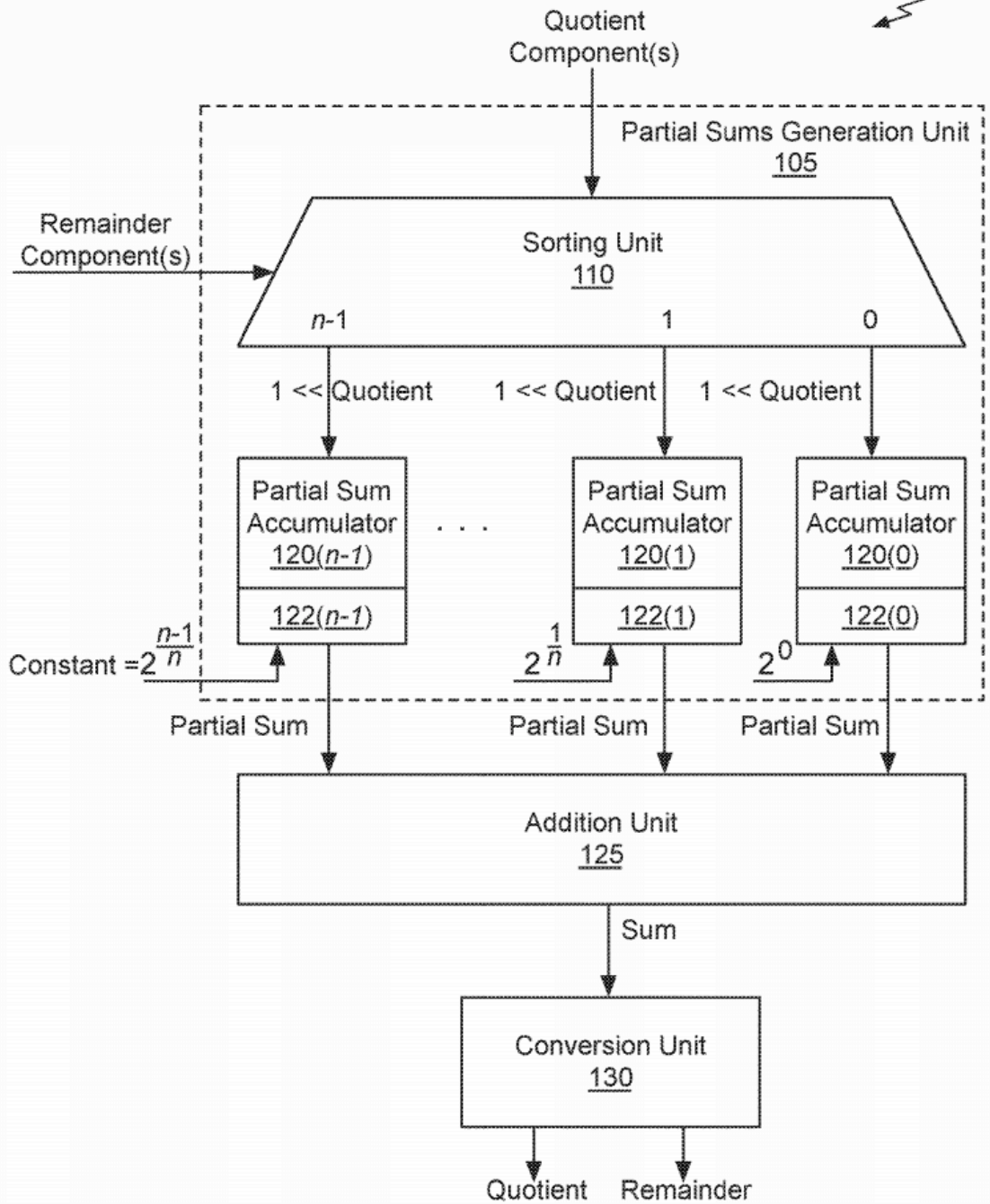
Related U.S. Application Data

(63) Continuation-in-part of application No. 16/549,683, filed on Aug. 23, 2019.

Publication Classification

(51) **Int. Cl.**
G06N 5/04 (2006.01)





Numbers being summed are one hot

Two bits of accumulator toggle on average

vs half of bits toggling for normal add

Wasteful to clock a 24b register

(19) **United States**

(12) **Patent Application Publication**

Dally et al.

(10) **Pub. No.: US 2021/0056399 A1**

(43) **Pub. Date: Feb. 25, 2021**

(54) **ASYNCHRONOUS ACCUMULATOR USING LOGARITHMIC-BASED ARITHMETIC**

(71) Applicant: **NVIDIA Corporation**, Santa Clara, CA (US)

(72) Inventors: **William James Dally**, Incline Village, NV (US); **Rangharajan Venkatesan**, San Jose, CA (US); **Brucek Kurdo Khailany**, Austin, TX (US); **Stephen G. Tell**, Chapel Hill, NC (US)

(21) Appl. No.: **16/750,917**

(22) Filed: **Jan. 23, 2020**

Related U.S. Application Data

(63) Continuation-in-part of application No. 16/549,683, filed on Aug. 23, 2019.

Publication Classification

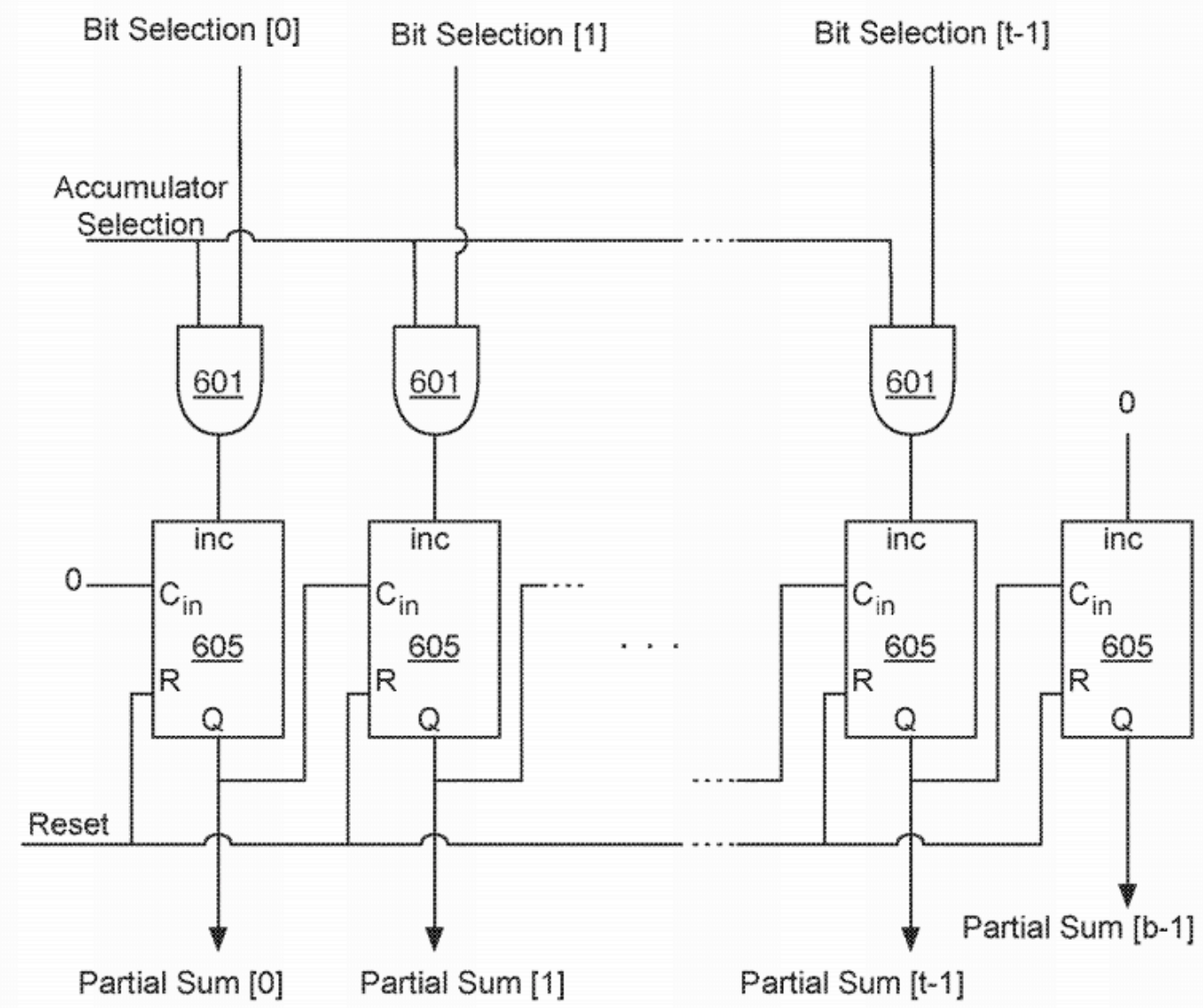
(51) **Int. Cl.**
G06N 3/063 (2006.01)
G06F 17/16 (2006.01)

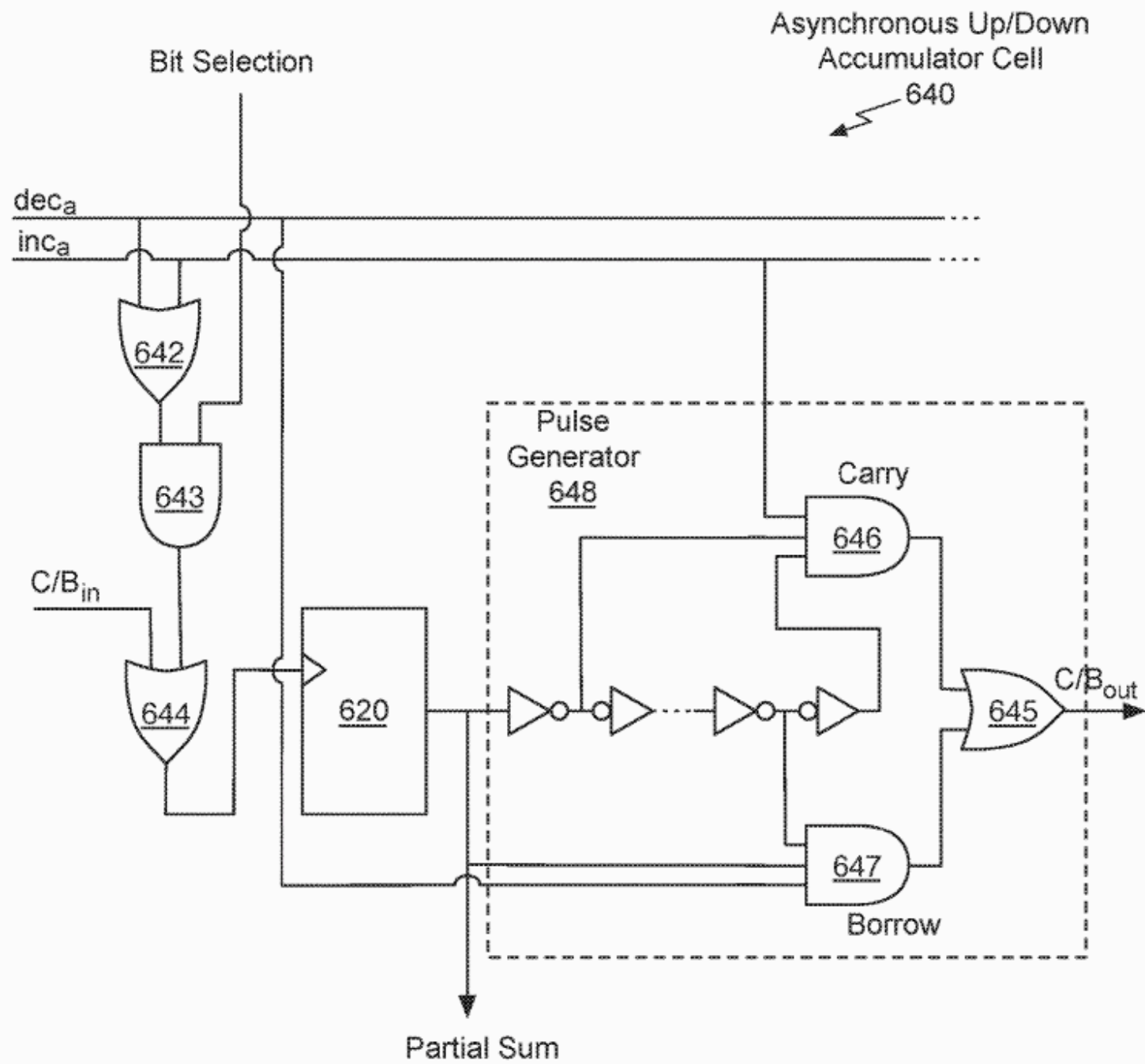
(52) **U.S. Cl.**
CPC **G06N 3/063** (2013.01); **G06F 17/16** (2013.01)

(57) **ABSTRACT**

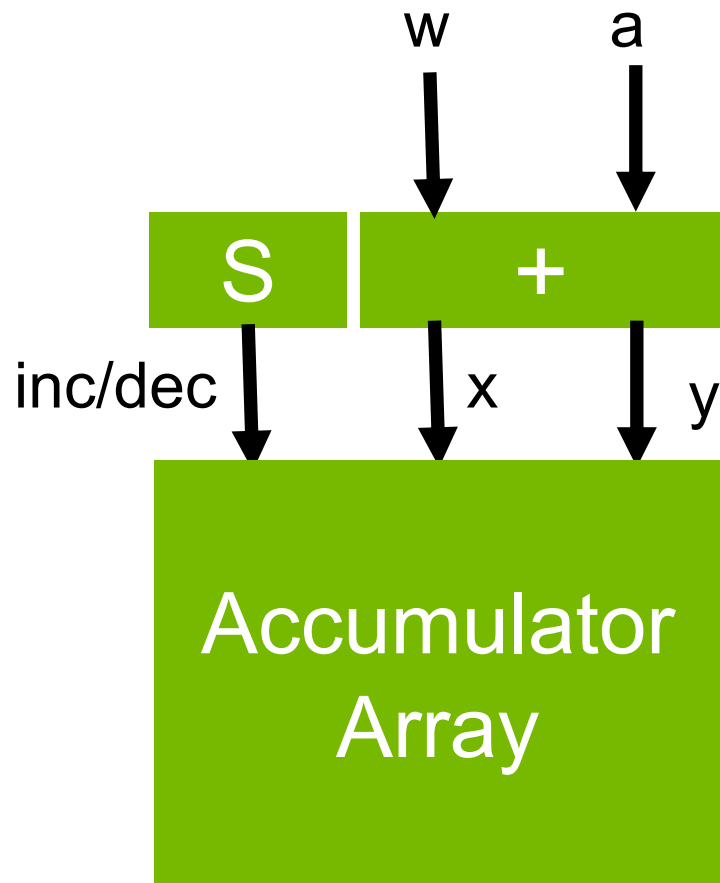
Neural networks, in many cases, include convolution layers that are configured to perform many convolution operations that require multiplication and addition operations. Compared with performing multiplication on integer, fixed-point, or floating-point format values, performing multiplication on logarithmic format values is straightforward and energy efficient as the exponents are simply added. However, performing addition on logarithmic format values is more complex. Conventionally, addition is performed by converting the logarithmic format values to integers, computing the sum, and then converting the sum back into the logarithmic format. Instead, logarithmic format values may be added by decomposing the exponents into separate quotient and remainder components, sorting the quotient components based on the remainder components, summing the sorted quotient components using an asynchronous accumulator to produce partial sums, and multiplying the partial sums by the remainder components to produce a sum. The sum may then be converted back into the logarithmic format.

Asynchronous
Accumulator
600





Big Picture



XOR of sign bits selects inc/dec (0/1)

Integer bits of sum (x) select bit position to increment

Fraction bits of sum (y) select which accumulator to increment

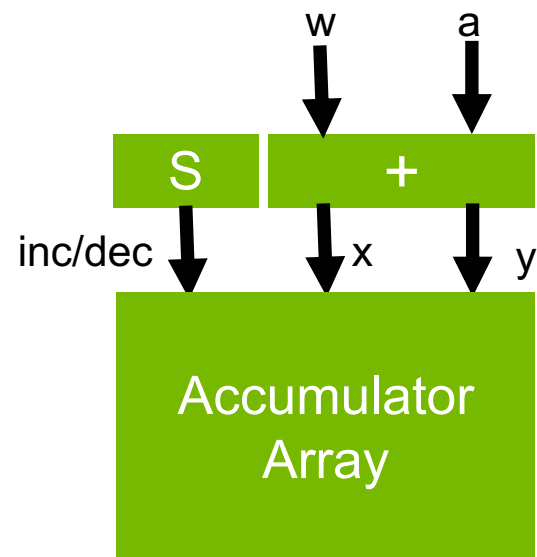
Energy Relative to Full-Adder Bit

Symbol	FA Equiv	Description
C	2	Carry-Lookahead adder bit
M	1.6	Multiplier partial product bit (b^2 of these in a b -bit mult)
R	2	Register bit
W	0.1	Wire width of full-adder bit

Energy Comparison

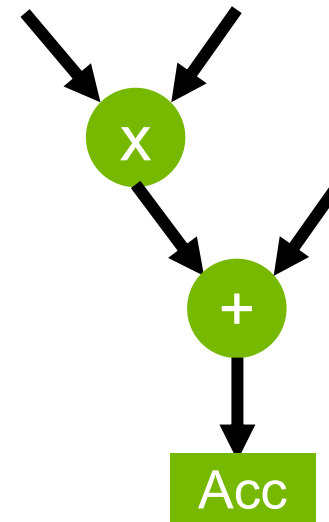
Log6 MAC Unit

Element	FA equiv	
6b CL Adder	6C	12
2 Acc Bits Toggle	4R	8
Select Wires	$2(32+32)W$	13
TOTAL	$6C+4R+128W$	33

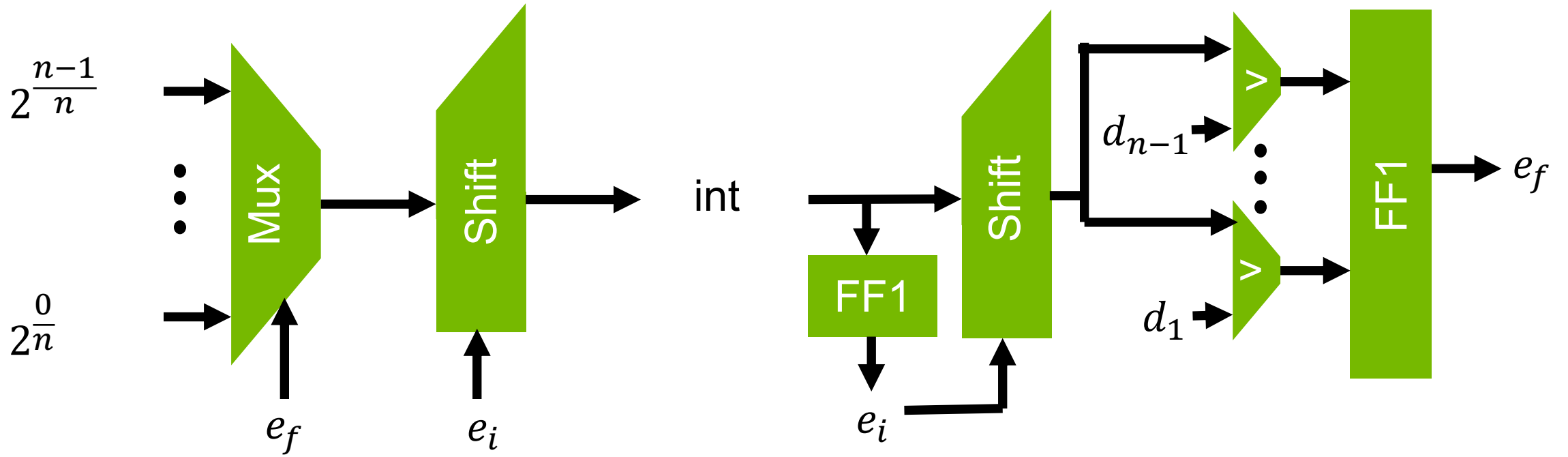


Int8 MAC Unit

Element	FA equiv	
8b Multiplier	64M	102
24b CL Adder	24C	48
24b Reg	24R	48
TOTAL	$64M+24(R+C)$	198



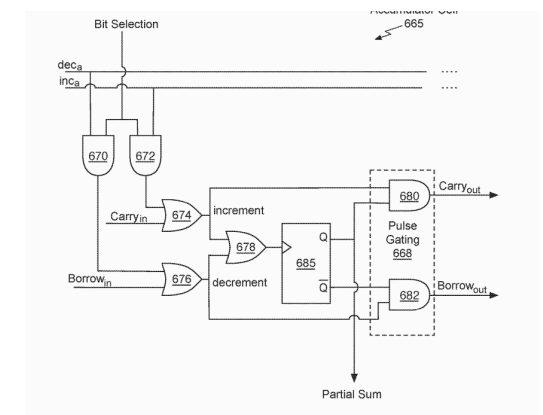
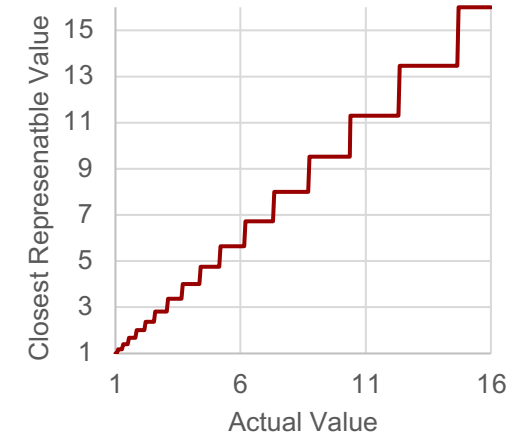
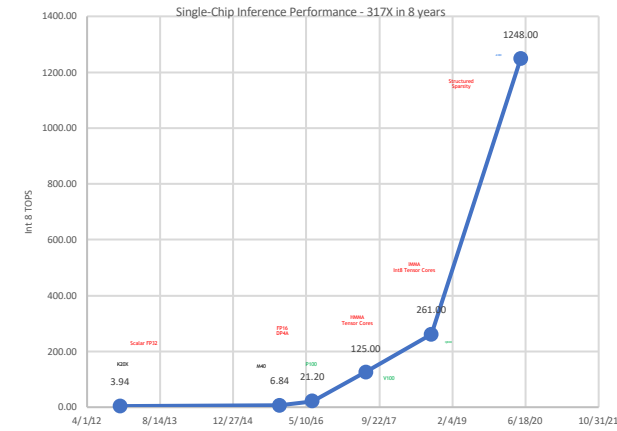
Conversion



Conclusion

Conclusion

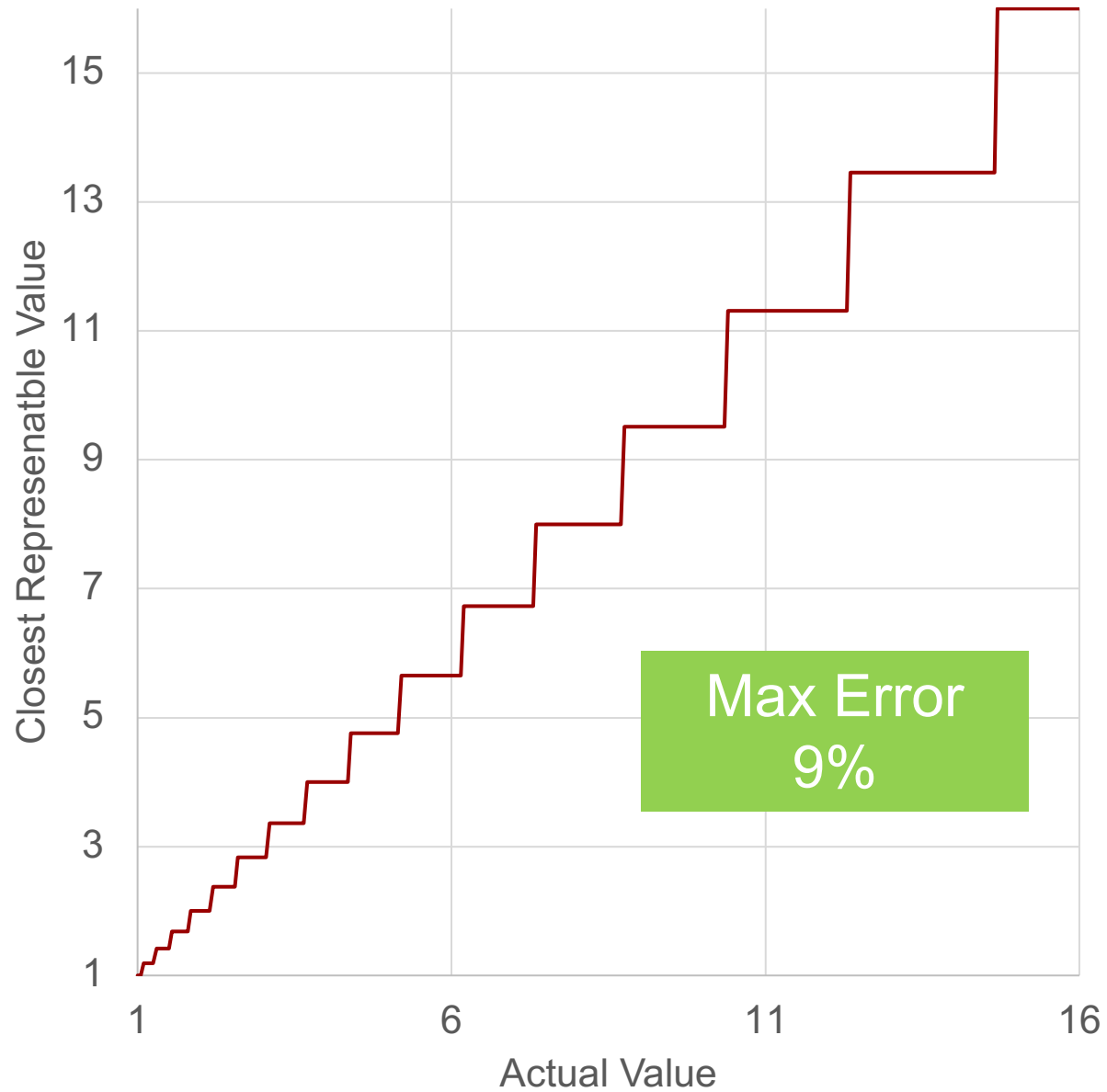
- GPU inference performance doubling every year
 - Better number representation, FP16, Int8, Int4, ...
 - Complex instructions, DP4A, HMMA, IMMA
 - Sparsity
 - Plumbing
- Accelerators experiment with new techniques
 - Sparsity, Tiling (data flows), Number Representation
- Log Numbers give more "bang per bit"
 - Same accuracy with fewer bits (less memory area, energy)
 - Very low energy arithmetic
- Asynchronous accumulators
 - Log results in one-hot add into accumulator
 - Only clock the bits that toggle



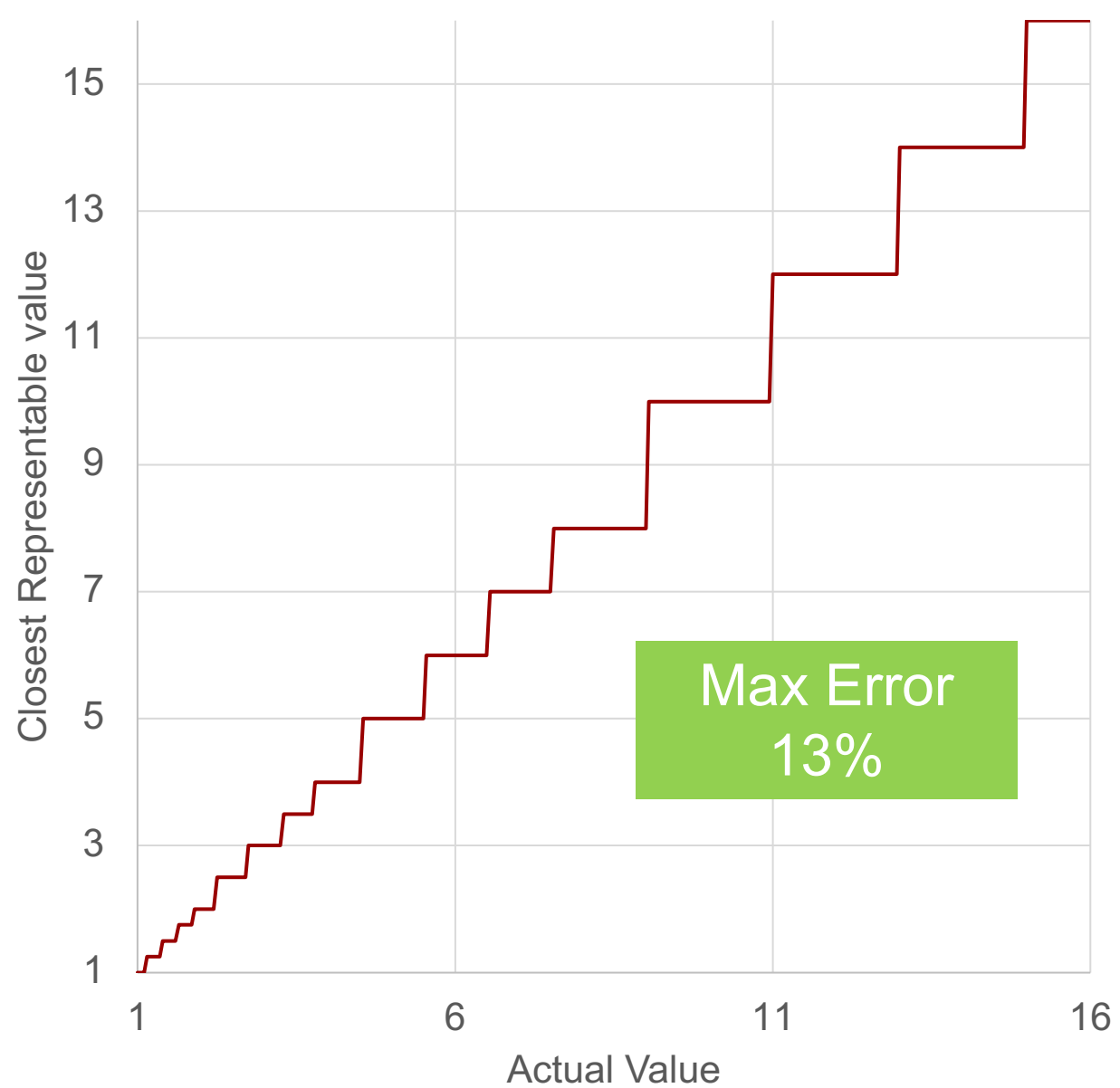


Backup

4-bit Log Representation (L2.2)



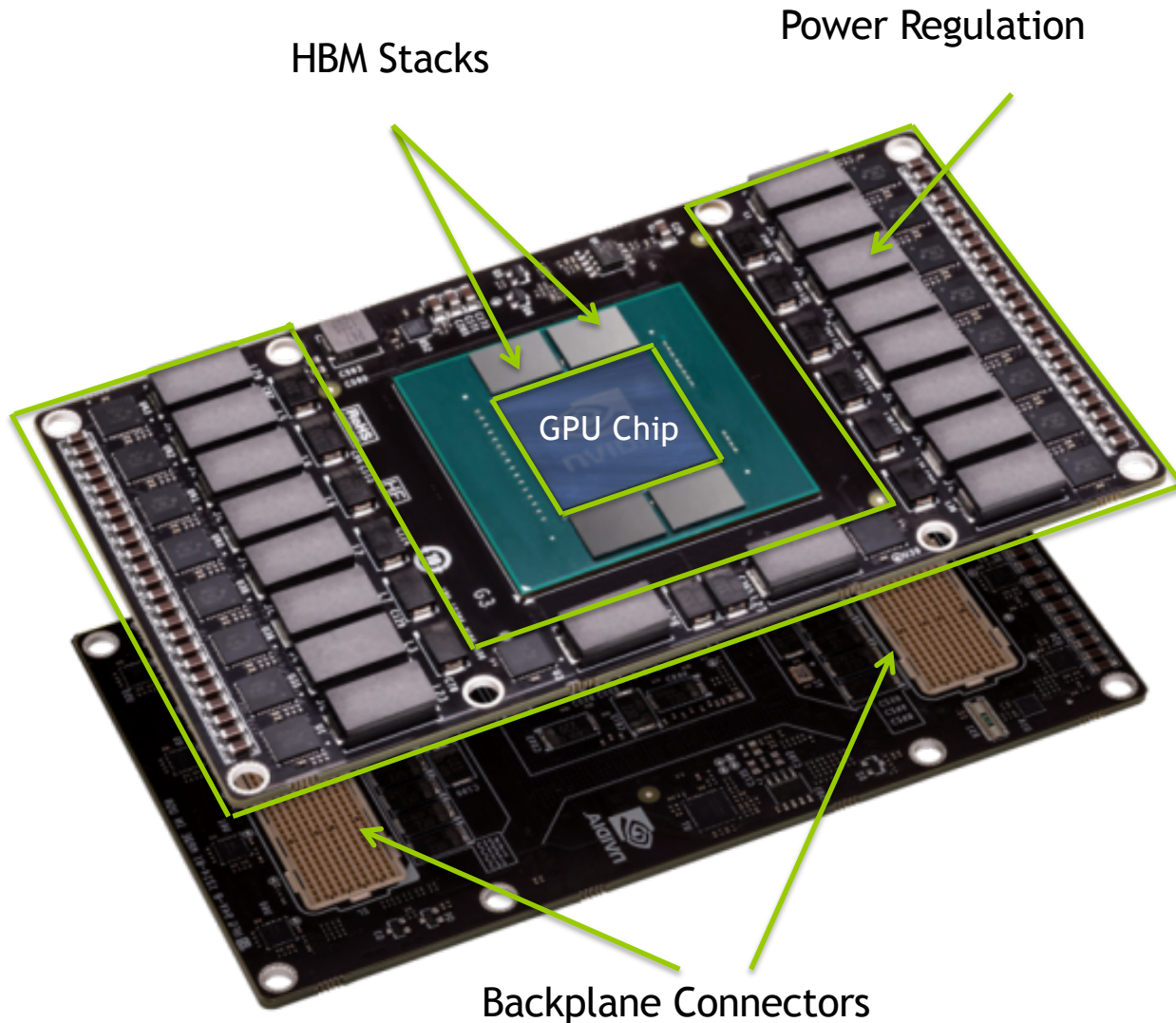
4-bit Floating Point (FP2.2)



Log vs FP

- Slightly better accuracy
 - Constant maximum error across range
 - FP error maximum at start of each subrange
- Much simpler arithmetic
 - FP still needs a small multiplier
 - FP needs normalization
 - Have to do a real add – not just an increment/decrement

PASCAL GP100



- 10 TeraFLOPS FP32
- 20 TeraFLOPS FP16
- 16GB HBM - 750GB/s
- 300W TDP
- 67GFLOPS/W (FP16)
- 16nm process
- 160GB/s NV Link