Accelerating Everything

Jeremy Kepner, Chansup Byun, Piotr Luszczek, LaToya Anderson, William Arcand, David Bestor, William Bergeron, Alex Bonn, Daniel Burrill, Vijay Gadepally, Michael Houle, Matthew Hubbell, Hayden Jananthan, Michael Jones, Lauren Milechin, Julie Mullen, Andrew Prout, Albert Reuther, Antonio Rosa, Charles Yee, Peter Michaleas

2026 June

Research was sponsored by the Department of the Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Department of the Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

Levels of Communication

Words

"How do I … "

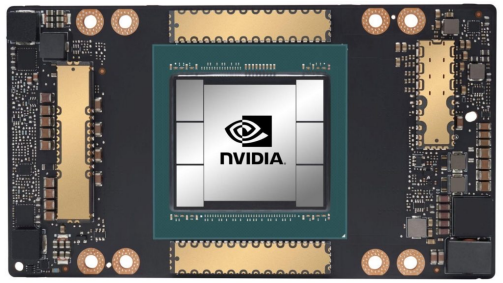
Code

```
for i=1:N
 for j=1:M
  for k=1:L
   C(i,j)=C(i,j)+A(i,k)*B(k,j);
```

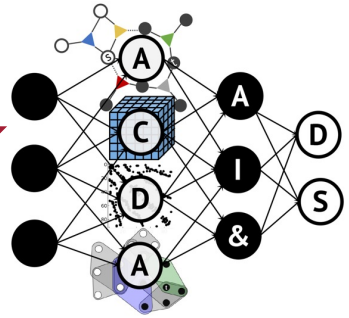
Math

$$C = A \times B$$

01101101



20% more productive

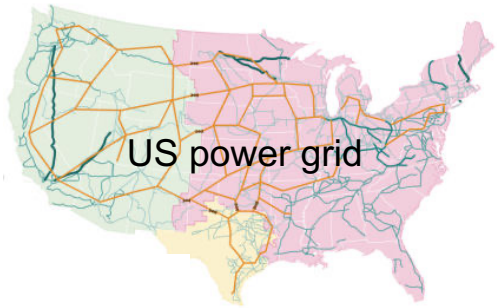


100x faster

One Motivation: AI Compute Energy

all gov't
edge data

total
energy
required



all gov't human
readable text

standard
processing

100,000x

standard
AI

1,000,000x

agentic
AI

relative energy per byte of input

Outline

- Hardware, Precision, Bandwidth
- Data & Math
- Benchmarks, Applications, Vectorization
- Results
- Summary

Numerical Precision Overview

Common Integer Representations

int8_t:

Range: -128 (-2^7) to 127 ($2^7 - 1$)

`+ b b b b b b b`

char, uint8_t:

Range: 0 to 255 ($2^8 - 1$)

`b b b b b b b b`

short, int16_t:

Range: -32,768 (-2^{15}) to 32,767 ($2^{15} - 1$)

`+ b b b b b b b b b b b b b b b`

unsigned short, uint16_t:

Range: 0 to 65,535 ($2^{16} - 1$)

`b b b b b b b b b b b b b b b b`

int, int32_t:

Range: -2,147,483,648 (-2^{31}) to 2,147,483,647 ($2^{31} - 1$)

`+ b b b b b b b b b b b b b b b b b b b b b b b b b b b b b b b`

unsigned int, uint32_t:

Range: 0 to 4,294,967,295 ($2^{32} - 1$)

`b b b b b b b b b b b b b b b b b b b b b b b b b b b b b b b b`

Common Floating-Point Representations

bfloat16: Brain Floating Point Format

Range: $\sim 1\times10^{-38}$ to $\sim 3\times10^{38}$ Number of Decimal Digits: ~2.3

Exponent: 8 bits Mantissa (Significand): 7 bits

`+ E E E E E E E E M M M M M M M`

fp16: Half-precision IEEE Floating Point Format

Range: $\sim 5.96\times10^{-8}$ to $\sim 6.50\times10^{-4}$ Number of Decimal Digits: ~3.3

Exponent: 5 bits Mantissa (Significand): 10 bits

`+ E E E E E M M M M M M M M M M`

fp32: Single-precision IEEE Floating Point Format

Range: $\sim 1\times10^{-38}$ to $\sim 3\times10^{38}$ Number of Decimal Digits: ~7.2

Exponent: 8 bits Mantissa (Significand): 23 bits

`+ E E E E E E E E M M M M M M M M M M M M M M M M M M M M M M M`

fp64: Double-precision IEEE Floating Point Format

Range: $\sim 1\times10^{-308}$ to $\sim 3\times10^{308}$ Number of Decimal Digits: ~15.9

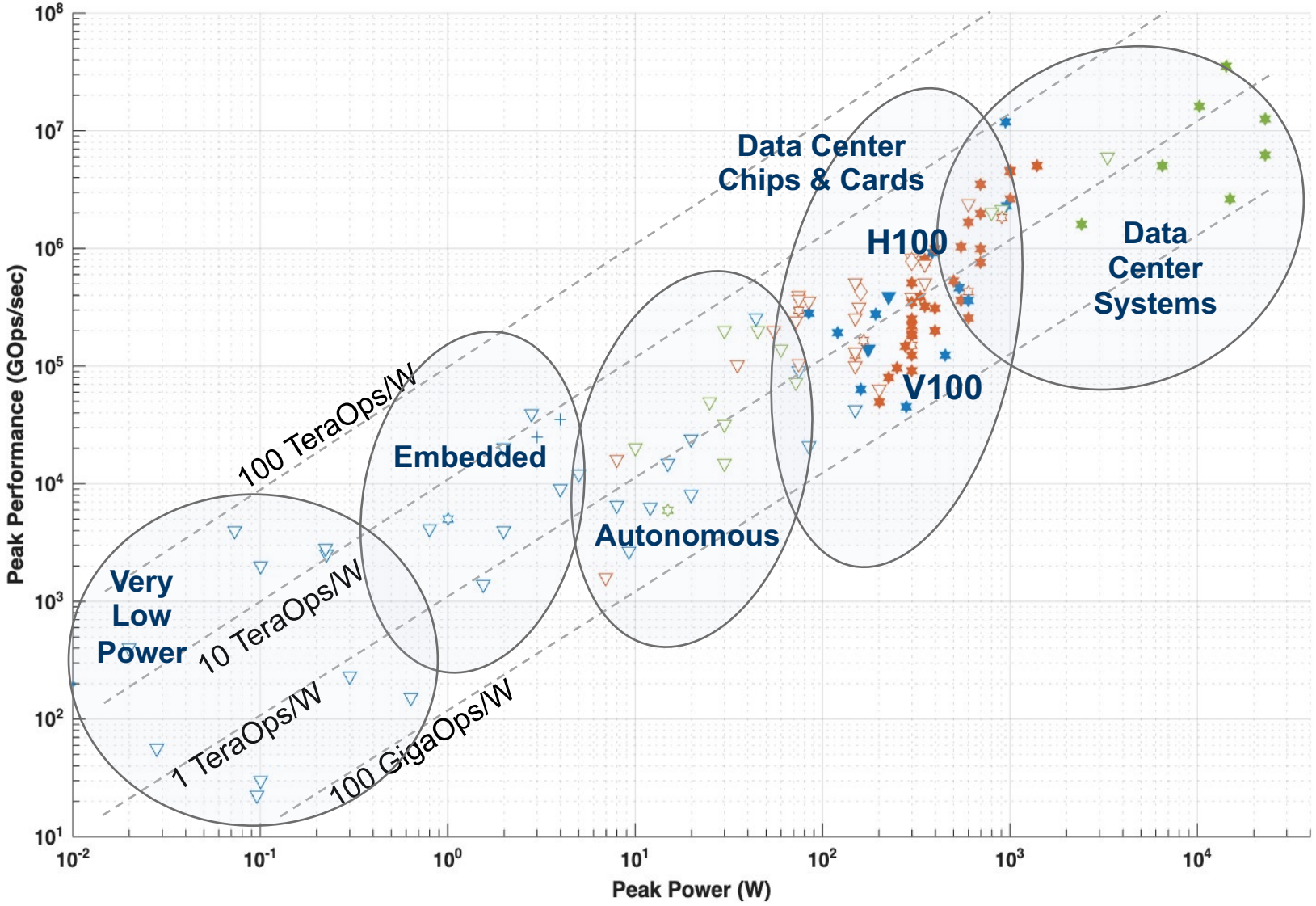
Exponent: 11 bits Mantissa (Significand): 52 bits

`+ E E E E E E E E E E E M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M`

Example: $123.45 = 1.2345 \times 10^2 = 0.12345 \times 10^3 =$ `0100 0010 1111 0110 1110 0110 0110 0110` (fp32)

`+` Sign bit of floating-point number
`E` Exponent bit of floating-point number
`M` Mantissa bit of floating-point number
`b` Bit of a fixed-point number

Lincoln AI Computing Survey (LAICS)



Legend

Computation Precision

- analog
- int1
- int2
- int4
- int8
- fp8
- int16
- int32
- fp16
- bf16
- fp32
- fp64

Form Factor

- Chip
- Card
- System

Computation Type

- Inference
- Training

Slide courtesy of Albert Reuther, MIT Lincoln Laboratory Supercomputing Center

Mixed Precision Performance Across Industry

Microsoft amazon Google

Peak specifications	Azure Maia 200	AWS Trainium3	Google TPU v7
Process Node	**3nm**	3nm	3nm
FP4 TFLOPS + E E M	**10,145**	2,517	n/a
FP8 TFLOPS + E E E E M M M	**5,072**	2,517	4,614
BF16 TFLOPS + E E E E E E E E M M M M M M M	**1,268**	671	2,307

HPCwire

Genesis Mission Will Lean Heavily on Ozaki Scheme for FP64 Capability

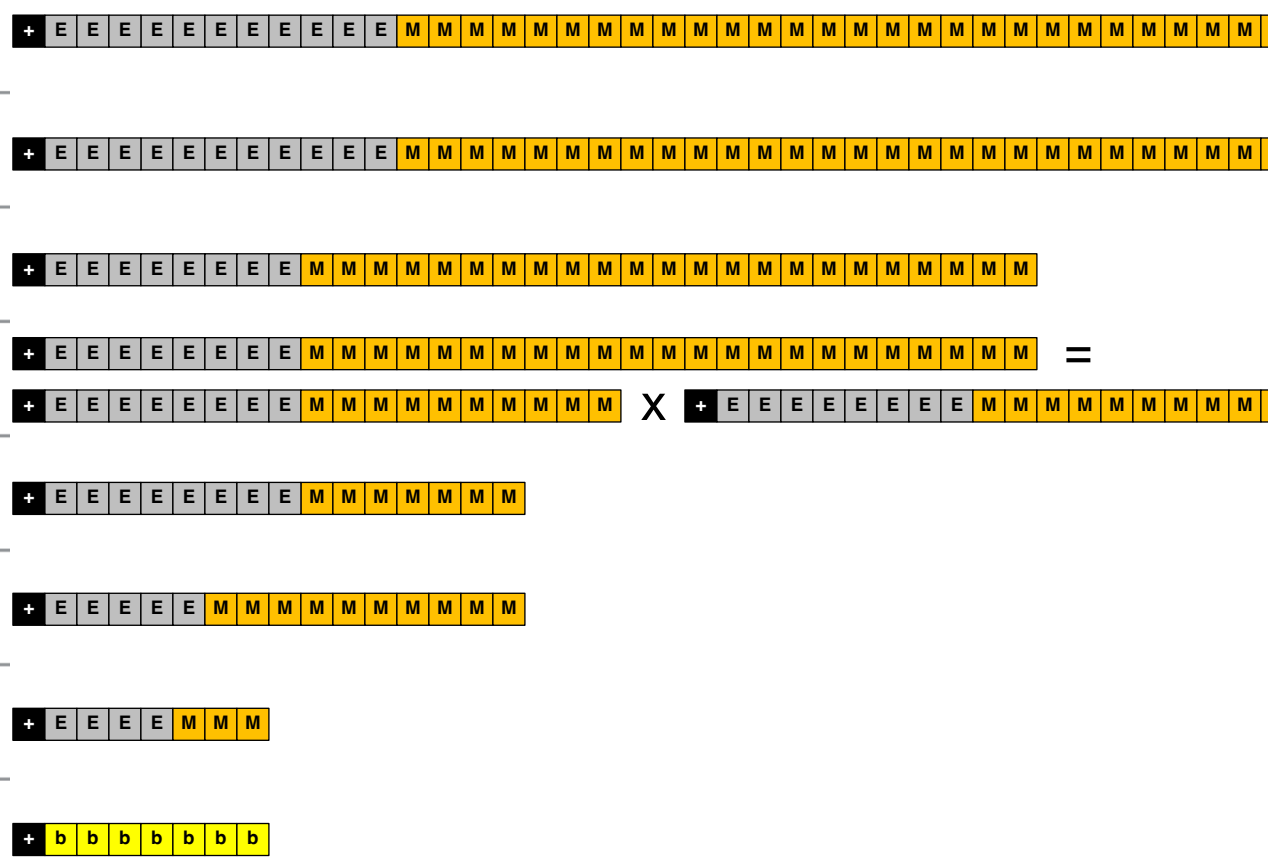
While the latest generation of GPUs have emphasized lower precision performance that are favored for AI workloads,

- **Many leading accelerators only have low-precision**
- **Rely on novel methods for emulating high-precision with low-precision**

https://www.hpcwire.com/aiwire/2026/01/27/microsoft-raises-the-ai-inference-bar-with-maia-200/
https://www.hpcwire.com/2026/02/17/genesis-mission-will-lean-heavily-on-ozaki-scheme-for-fp64-capability/

Example: Nvidia H100

Technical Specifications	H100 NVL
FP64	30 teraFLOPS
FP64 Tensor Core	60 teraFLOPS
FP32	60 teraFLOPS
TF32 Tensor Core*	835 teraFLOPS
BFLOAT16 Tensor Core*	1,671 teraFLOPS
FP16 Tensor Core*	1,671 teraFLOPS
FP8 Tensor Core*	3,341 teraFLOPS
INT8 Tensor Core*	3,341 TOPS



Slide 8

*With sparsity
NVL = NVLink

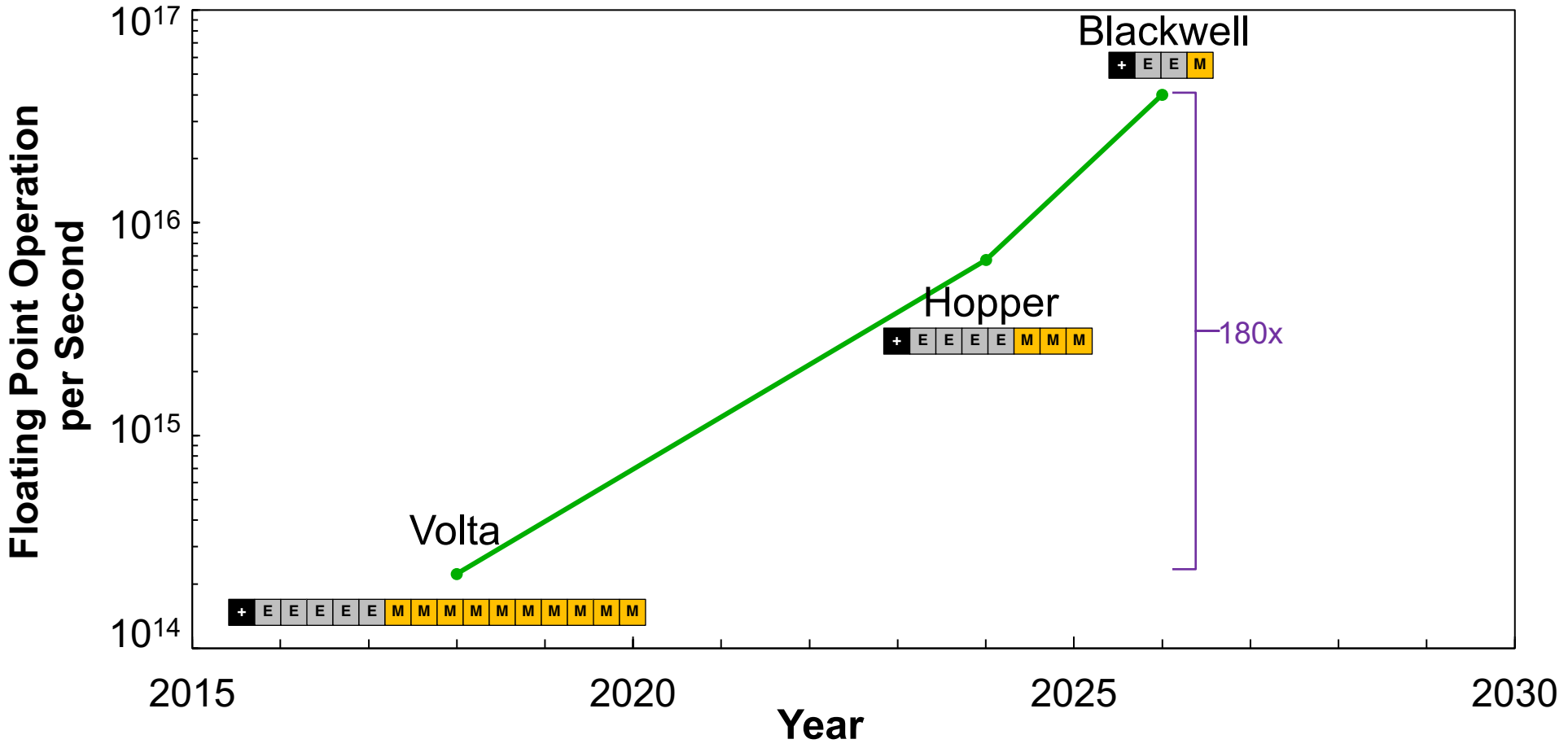
https://resources.nvidia.com/en-us-hopper-architecture/nvidia-tensor-core-gpu-datasheet

[1] *Decompositional Factorizations with FP64 Floating-Point Emulation in INT8*, Luszczek et al, IEEE HPEC

[2] *Parallel Hypersparse, Matrix Based Graph Streaming, and Complex-Index Matrices*, Jananthan et al, IEEE HPEC

Precision Performance Trend

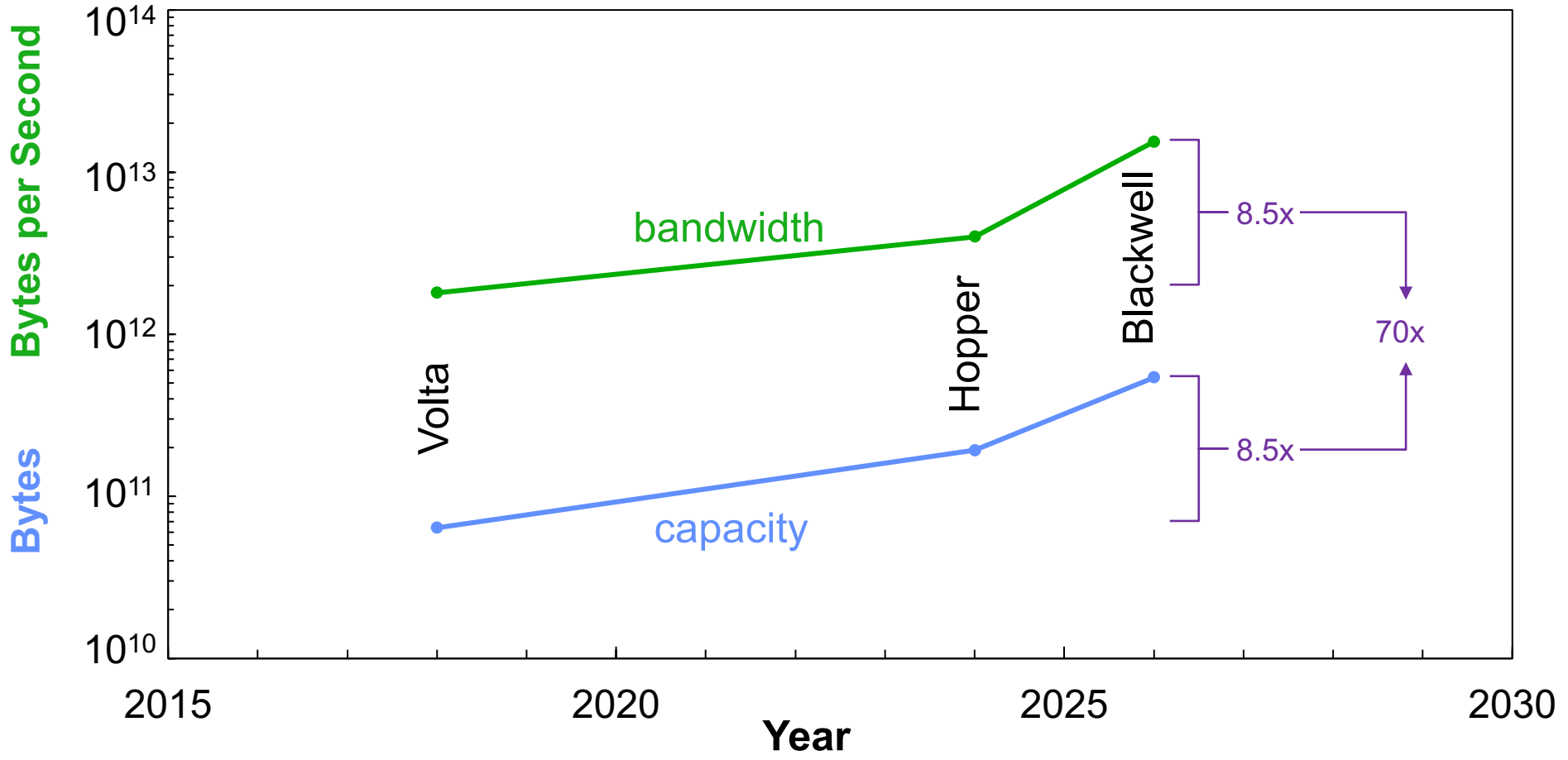
- Nvidia Dual GPU -



Significant improvements for reduced precision algorithms and emulation of high precision with low precision

Memory Bandwidth Capacity Trend

- Nvidia Dual GPU -



Significant improvements for sparse, hypersparse, associative (token) array algorithms and spectral methods

Outline

- Hardware, Precision, Bandwidth
- Data & Math
- Benchmarks, Applications, Vectorization
- Results
- Summary

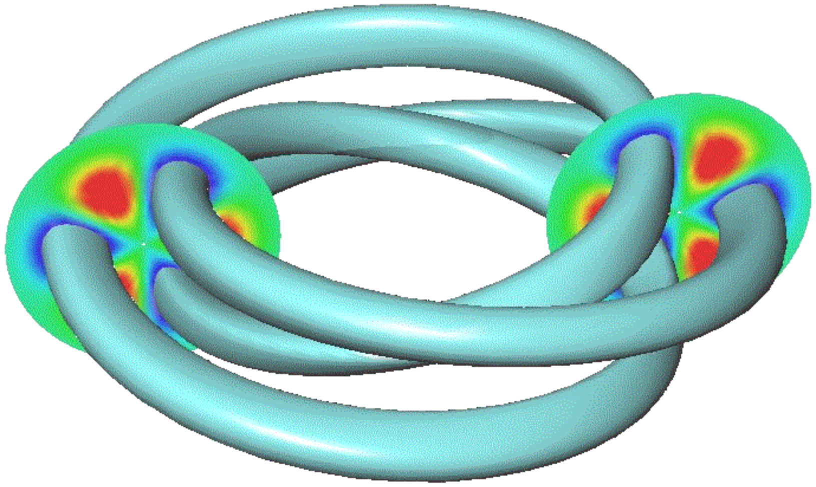
Accelerated Computing Data Foundations

Dense Double Precision (64 bits)

1 … … … … … … … … … … … … ~2^{15}

1	0.1234 567890 12345	0.1234 567890 12345	0.1234 567890 12345	0.1234 567890 12345	0.1234 567890 12345
… … …	0.1234 567890 12345	0.1234 567890 12345	0.1234 567890 12345	0.1234 567890 12345	0.1234 567890 12345
… … …	0.1234 567890 12345	0.1234 567890 12345	0.1234 567890 12345	0.1234 567890 12345	0.1234 567890 12345
… … …	0.1234 567890 12345	0.1234 567890 12345	0.1234 567890 12345	0.1234 567890 12345	0.1234 567890 12345
… ~2^{15}	0.1234 567890 12345	0.1234 567890 12345	0.1234 567890 12345	0.1234 567890 12345	0.1234 567890 12345

Example: Physical Simulation



+ E E E E E E E E E E E M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M M

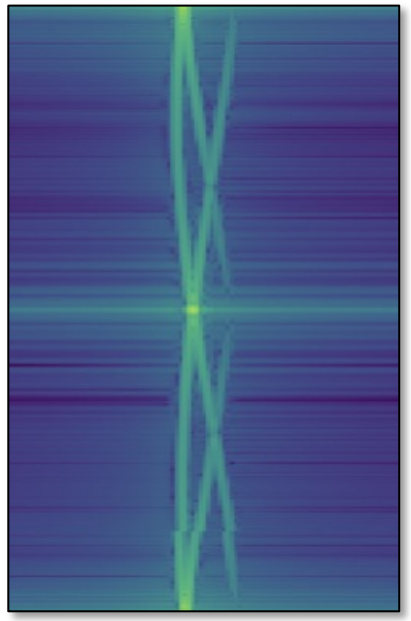
Accelerated Computing Data Foundations

Dense Half Precision (16 bits)

1 … … … … … … … … … … … … … ~2^{15}

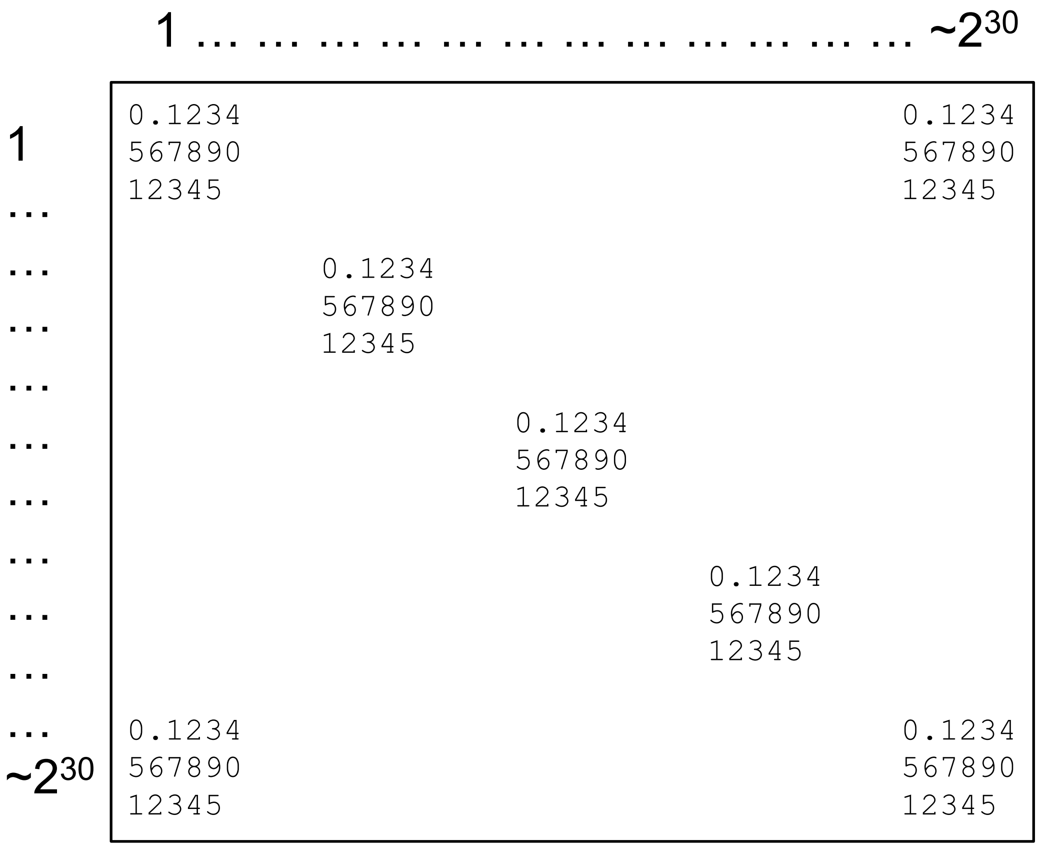
1 … … … … … … … … … … … … … ~2^{15}				
0.1234	0.1234	0.1234	0.1234	0.1234
0.1234	0.1234	0.1234	0.1234	0.1234
0.1234	0.1234	0.1234	0.1234	0.1234
0.1234	0.1234	0.1234	0.1234	0.1234
0.1234	0.1234	0.1234	0.1234	0.1234

Example: Sensor Processing

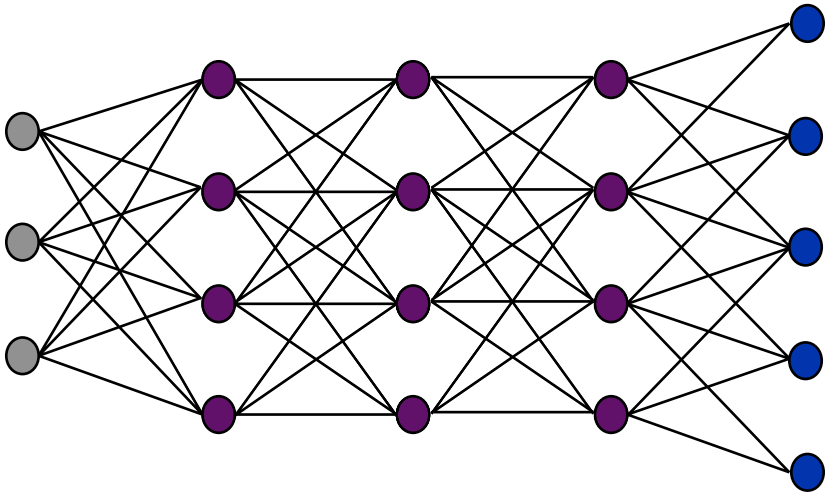


Accelerated Computing Data Foundations

Sparse Double Precision (64 bits)



Example: Networks



Accelerated Computing Data Foundations

Hypersparse Double Precision (64 bits)

1 … … … … … … … … … … … … $\sim 2^{60}$

1 … … … … … … … … … $\sim 2^{60}$

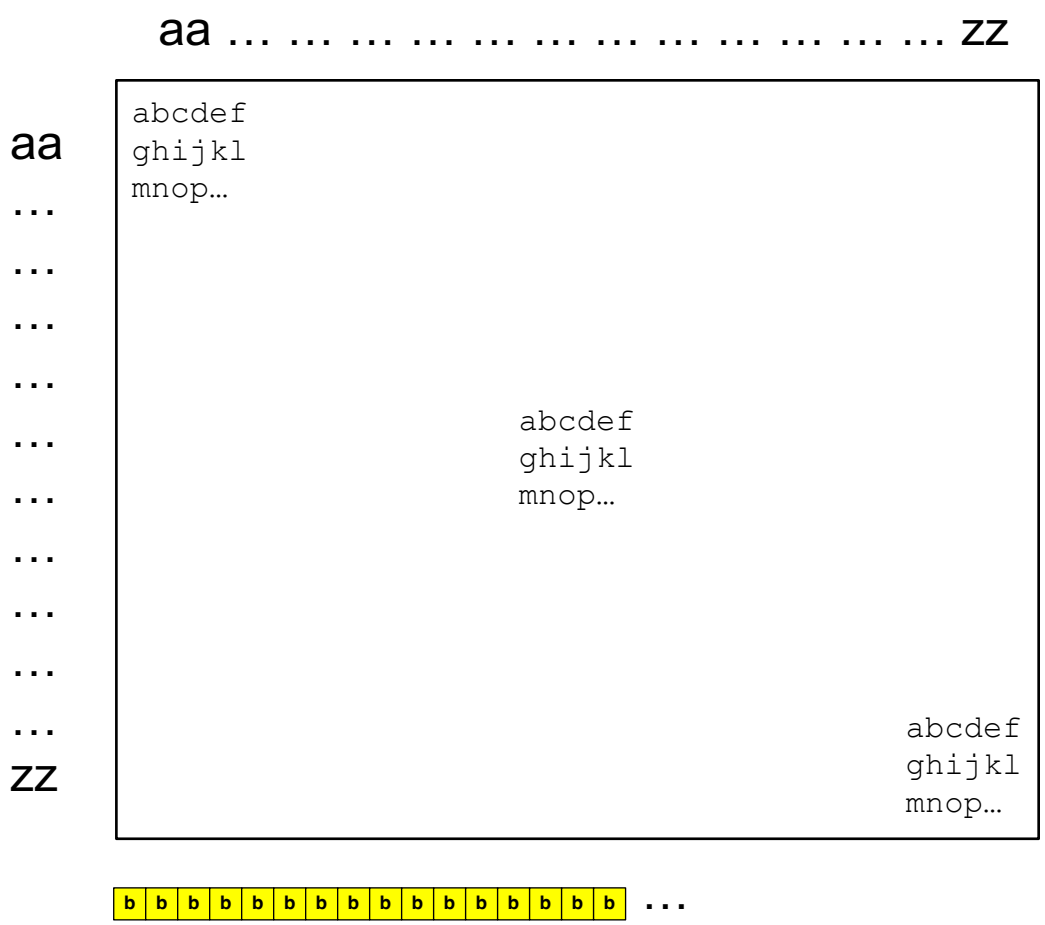
```
0.1234
567890
12345

                    0.1234
                    567890
                    12345

                                        0.1234
                                        567890
                                        12345
```

+	E	E	E	E	E	E	E	E	E	E	E	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M

Example: Entity Relationships

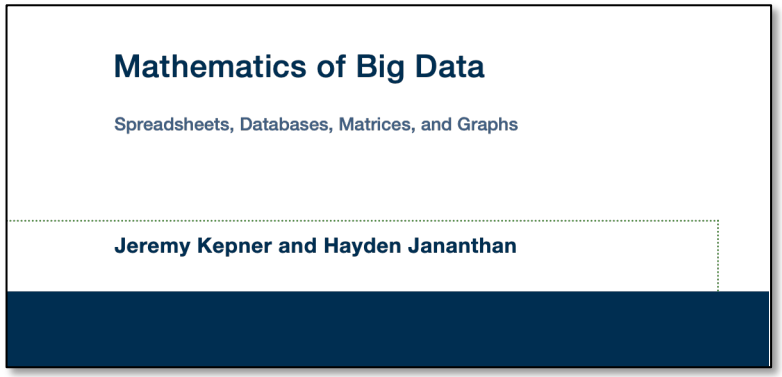
Accelerated Computing Data Foundations

Associative (Token) Arrays



Example: Arbitrary Data

	A	B	C	D	E	F	G	H	I	J	K
1	1	2	2	1		Code	Name	Job		Date	$ in bank
2	2	3	3	2		A0001	Alice	scientist		2000 Jan 01	$11,700
3	2	3	3	2		B0002	Bob	engineer		2001 Jan 01	$10,600
4	1	2	2	1		C0003	Carl	mathematician		2002 Jan 01	$10,200
5										2003 Jan 01	$8,600
6										2004 Jan 01	$10,400
7										2005 Jan 01	$10,600
8					y=10	6	12	18		2006 Jan 01	$10,900
9					y=8	5	10	15		2007 Jan 01	$12,300
10					y=6	4	8	12		2008 Jan 01	$12,600
11					y=4	3	6	9		2009 Jan 01	$9,000
12					y=2	2	4	6		2010 Jan 01	$10,600
13					y=0	1	2	3		2011 Jan 01	$11,700
14						x=0	x=5	x=10			



$$2 \times (3 + 4) = (2 \times 3) + (2 \times 4)$$

$$2 \otimes (3 \oplus 4) = (2 \otimes 3) \oplus (2 \otimes 4)$$

$\oplus = +$ $\quad\quad$ $\otimes = \times$

$\oplus = max$ $\quad\quad$ $\otimes = +$

$\oplus = \cup$ $\quad\quad$ $\otimes = \cap$

$$A \otimes (B \oplus C) = (A \otimes B) \oplus (A \otimes C)$$

$A = 2$ $B = 3$ $C = 4$

Associative Arrays

A B C	matrices
A B C	graphs/networks
A B C	database tables
A B C	spreadsheets
A B C	LLMs

$$A(i,j) = v$$

$$i \in I \qquad j \in J \qquad v \in V$$

matrices/graphs/networks: I,J integers V reals

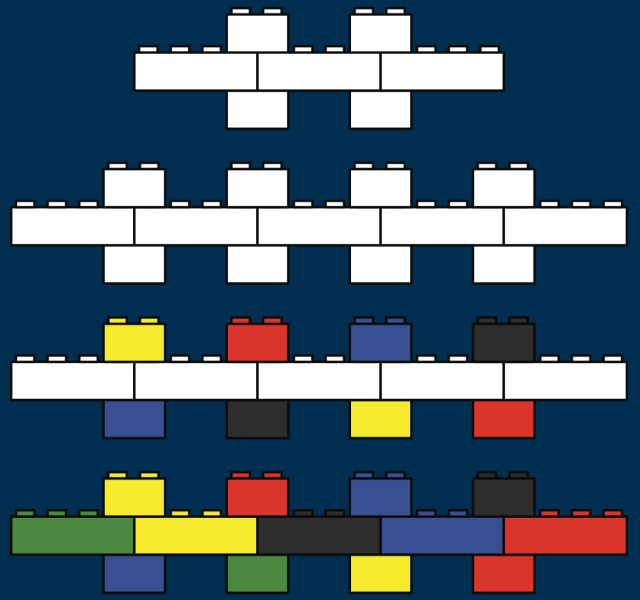
database tables/spreadsheets/LLMs: I,J,V tokens

Associative Arrays: I,J,V strict totally ordered sets

Mathematics of Big Data

Spreadsheets, Databases, Matrices, and Graphs

Jeremy Kepner and Hayden Jananthan



Foreword by Charles Leiserson

MIT LINCOLN LABORATORY SERIES

Outline

- Hardware, Precision, Bandwidth
- Data & Math
- Benchmarks, Applications, Vectorization
- Results
- Summary

Benchmark: STREAM Triad (memory bandwidth)

$$C = A + qB$$

Python

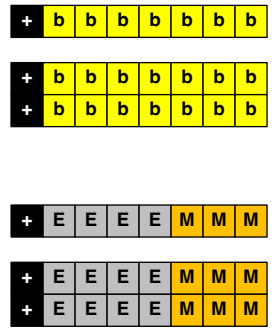
```
C = A + q*B
```

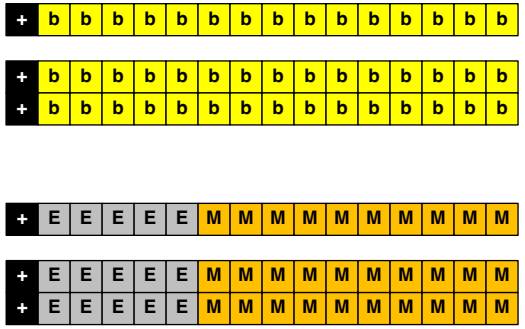
Matlab

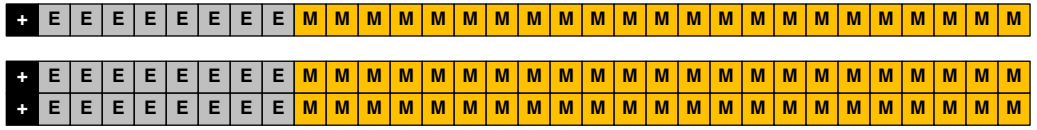
```
C = A + q*B;
```

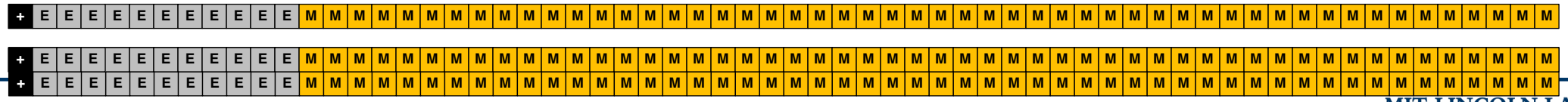
Benchmark: Dense Matrix Multiplication

$$C = A \times B$$









Python

```
C = A @ B
```

Matlab

```
C = A * B;
```

Associative (Token) Array: Construction, Addition, Multiplication

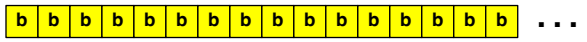
$$\mathbf{A} = \mathbb{A}(I,J,1) \qquad \mathbf{C} = \mathbf{A} + \mathbf{B} \qquad \mathbf{C} = \mathbf{A} \times \mathbf{B}$$

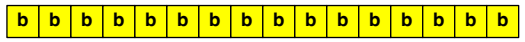
Python

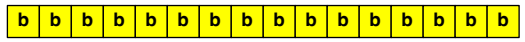
```
A = Assoc(I,J,1)        C = A + B        C = A @ B
```

Matlab/Octave

```
A = Assoc(I,J,1,@sum);        C = A + B;        C = A * B;
```



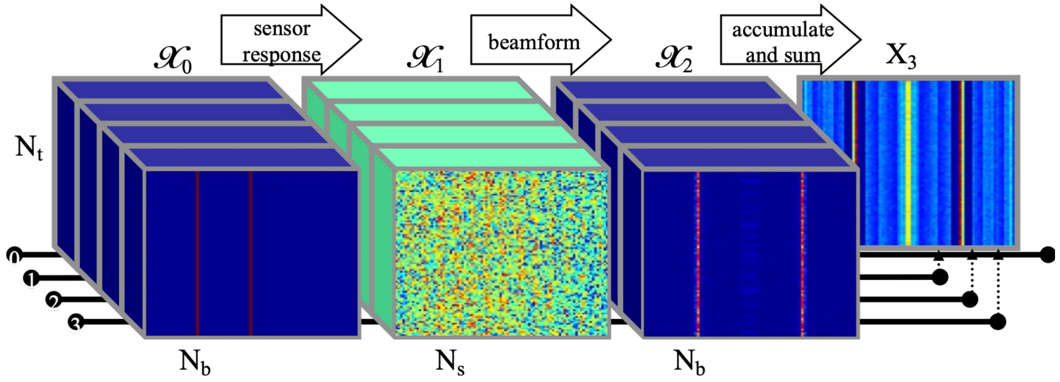
...


...


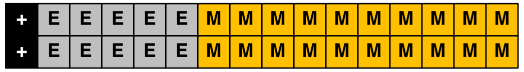
...

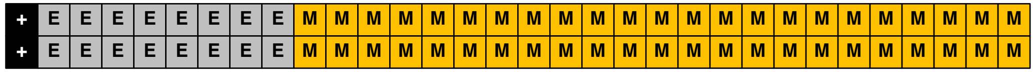
Application: Beamforming (complex matrix multiply)

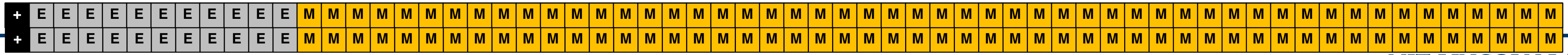
- Mineral exploration
- Medical imaging (MRI, ultrasound, …)
- Wireless communications
- Land/sea/undersea/air/space manned/unmanned navigation and traffic management
- Satellite imaging
- Weather monitoring
- Oceanography
- Astronomy



Phased Array Beamformer

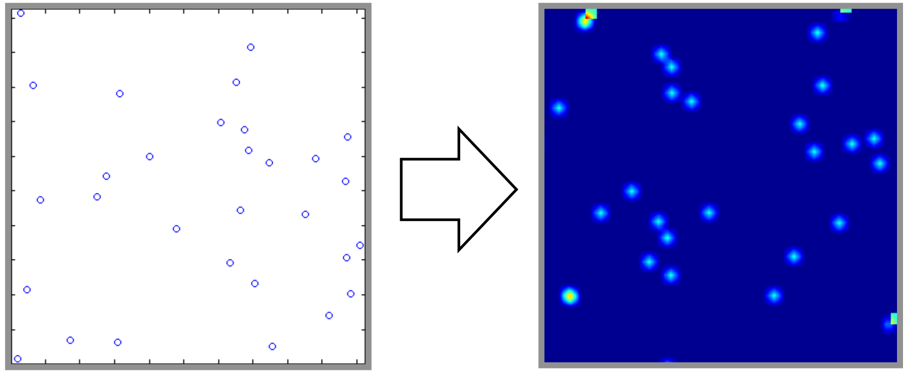


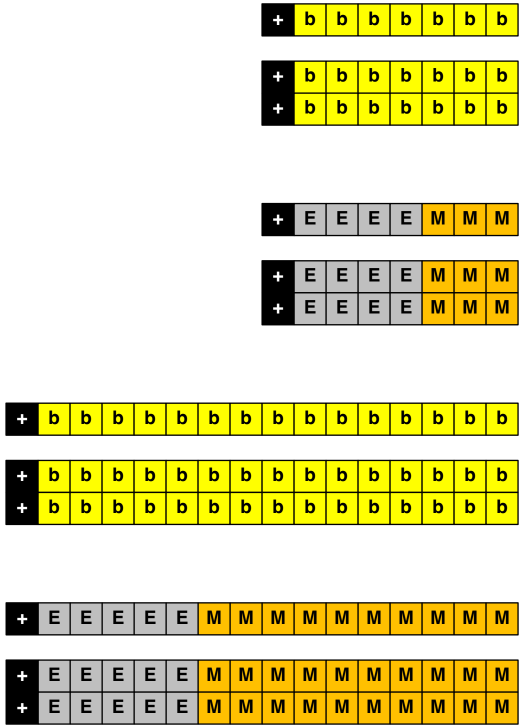


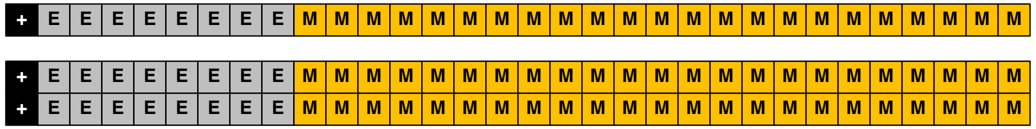


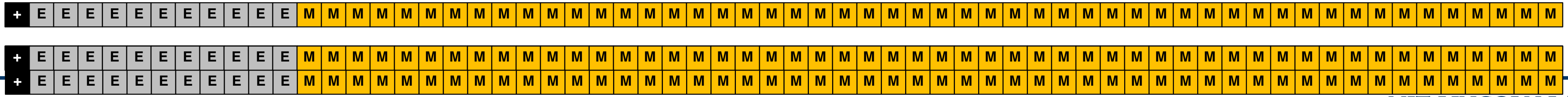
Application: Blurimage (2D convolution)

- Convolutional neural networks
- Medical imaging
- Video
- Land/sea/undersea/air/space manned/unmanned navigation and traffic management
- Satellite imaging
- Weather monitoring
- Oceanography
- Astronomy

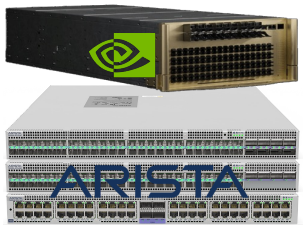


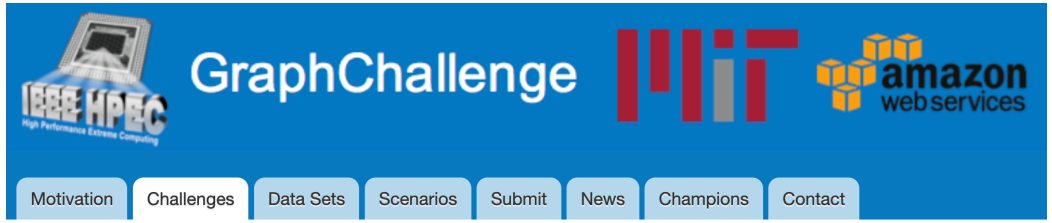


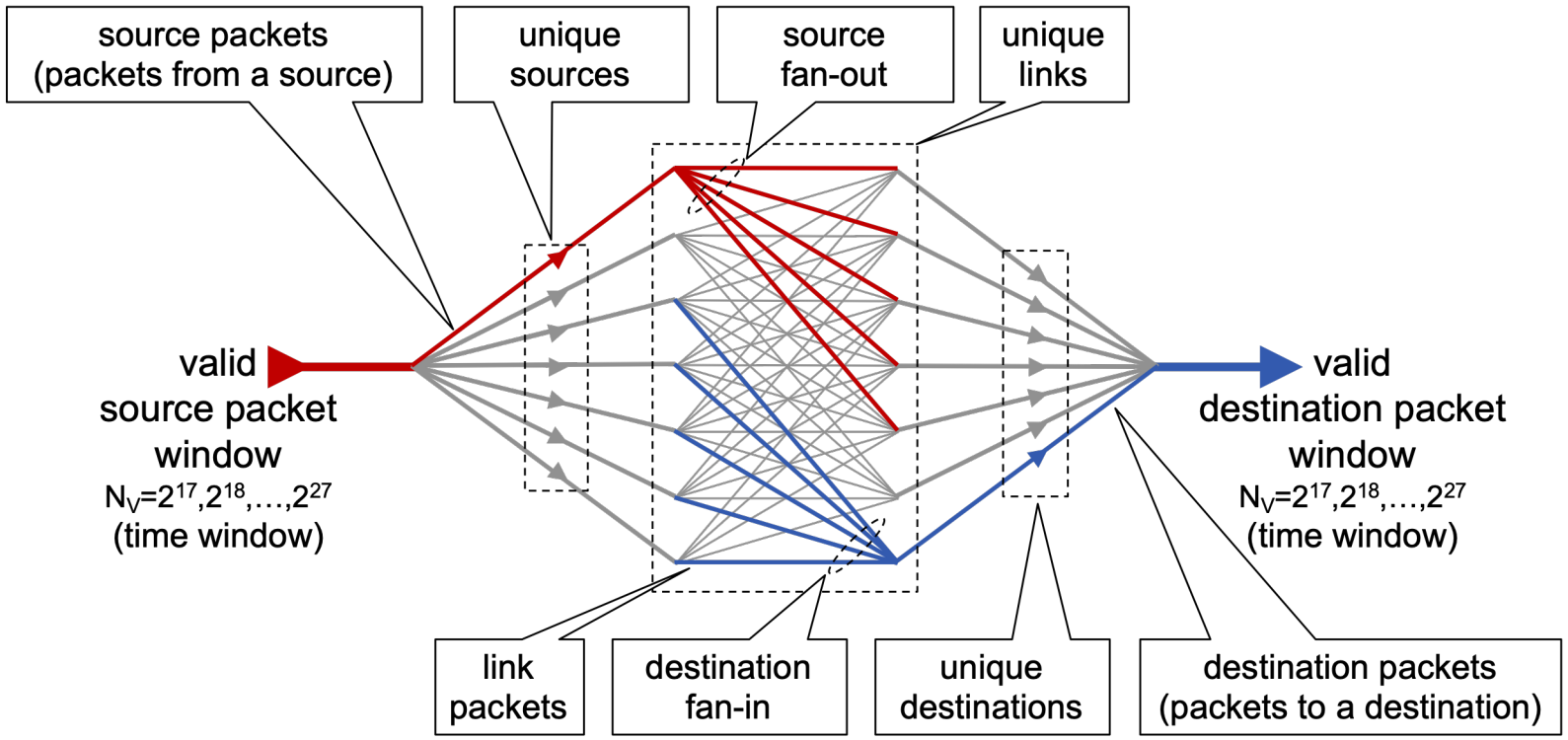


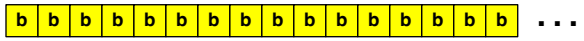


Application: Network Sensing (associative arrays)









Key Code Concept: Vectorized Computation

$$C = A \times B$$

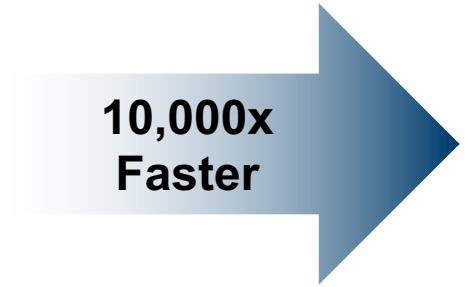
Standard Code

Python

```
for i in range(N):
 for j in range(M):
  for k in range(L):
   C[i,j] += A[i,k] * B[k,j]
```

Matlab

```
for i=1:N
 for j=1:M
  for k=1:L
   C(i,j)=C(i,j)+A(i,k)*B(k,j);
```



Vectorized Code

Python

```
C = A @ B
```

Matlab

```
C = A * B;
```

Key Code Concept: Vectorized Row/Column Selection

$$\mathbf{A}_{sub} = \mathbf{A}(I,J) = \mathbf{I}_{diag} \times \mathbf{A} \times \mathbf{J}_{diag}$$

Standard Code

Python

```
for i in I:
 for j in J:
  Asub[i][j] = A[i][k]
```

Matlab

```
for i=I
 for j=J
   Asub(i,j)= A(i,k);
```

10,000x Faster

Vectorized Code

Python

```
Idiag = csr_array(I,I,1)
Jdiag = csr_array(J,J,1)
Asub = Idiag @ A @ Jdiag
```

Matlab

```
Idiag = sparse(I,I,1);
Jdiag = sparse(J,J,1);
Asub = Idiag * A * Jdiag;
```

Outline

- Hardware, Precision, Bandwidth
- Data & Math
- Benchmarks, Applications, Vectorization
- Results
- Summary

Hardware

Node Label	Processor				Memory	
	Era	Part	Clock	Cores	Part	Size
amd-e9	2024	Dual AMD EPYC 9254	2.9 GHz	48	DDR5	750 GB
- h100nvl	2024	Dual Nvidia H100 NVL	1.7 GHz		HBM3	188 GB
xeon-p8	2020	Dual Xeon Platinum 8260	2.4 GHz	48	DDR4	192 GB
xeon-g6	2018	Dual Xeon Gold 6248	2.5 GHz	40	DDR4	384 GB
- v100	2018	Dual Nvidia V100	1.2 GHz		HBM2	64 GB

Operation Counts

Operation	Standard	Alternative	Standard/Alternative
Matrix Addition	N^2	$k\ N$ (sparse)	N/k
Matrix Multiplication	N^3	$k^2\ N$ (sparse)	$(N/k)^2$
2D Convolution	$N^2\ N_{kern}^2$	$N^2 \log(N^2)$ (FFT)	$N_{kern}^2 / \log(N^2)$

N = matrix size; k = average non-zeros per row; N_{kern} = kernel size

Large fast memory enables significant benefits for alternative sparse algorithms and spectral methods

Benchmark: STREAM Triad (memory bandwidth)

Accessible Performance GB/s

	Year	2018	2020	2024	2018	2024	2018	2024
	Language	Matlab	Matlab	Matlab	Matlab	Matlab	Python	Python
	Type	CPU	CPU	CPU	GPU	GPU	GPU	GPU
Benchmark	Precision / HW	xeon-g6	xeon-p8	amd-e9254	v100	h100nvl	v100	h100nvl
Stream	FP64	80	89	249	967	4,257	402	3,962

20x

$$C = A + qB$$

Memory bandwidth performance is readily accessible in high level programming environments via vectorized operations

E=Emulation
u = unsupported

Bold = much better than double MatMul
Blue = best precision

Benchmark: Dense Matrix Multiplication

Accessible
Performance
GigaFlops

Benchmark	Precision / HW	Year	2018	2020	2024	2018	2024	2018	2024
		Language	Matlab	Matlab	Matlab	Matlab	Matlab	Python	Python
		Type	CPU	CPU	CPU	GPU	GPU	GPU	GPU
			xeon-g6	xeon-p8	amd-e9254	v100	h100nvl	v100	h100nvl
MatMul	FP64		2,204	2,713	2,218	14,041	103,479	13,852	85,984
MatMul	FP32		**4,684**	**5,808**	**4,002**	**27,728**	97,200	**27,556**	86,900
MatMul	FP16		E	E	E	u	**686,129**	**184,460**	**1,006,242**
MatMul	INT16		u	u	u	u	u	E	E
MatMul	FP8		u	u	u	u	u	u	**1,752,992**
MatMul	INT8		u	u	u	u	u	u	INT8
MatMul	FP64 (complex)		2,628	2,990	2,262	13,936	104,017	C	3,230
MatMul	FP32 (complex)		**5,216**	**6,182**	**4,642**	**27,239**	97,699	C	6,462
MatMul	FP16 (complex)		E	E	E	u	**295,656**	S	S
MatMul	INT16 (complex)		u	u	u	u	u	u	u
MatMul	FP8 (complex)		u	u	u	u	u	u	u
MatMul	INT8 (complex)		u	u	u	u	u	u	u

175x

400x

60x

$$C = A \times B$$

Mixed precision performance is readily accessible in high level programming environments via vectorized operations

E=Emulation
u = unsupported

Bold = much better than double MatMul
Blue = best precision

Associative (Token) Array: Construction, Plus, MatMul

Accessible Performance GigaFlops

Benchmark	Precision / HW	Year	2018	2020	2024	2018	2024	2018	2024
		Language	Matlab	Matlab	Matlab	Matlab	Matlab	Python	Python
		Type	CPU	CPU	CPU	GPU	GPU	GPU	GPU
			xeon-g6	xeon-p8	amd-e9254	v100	h100nvl	v100	h100nvl
Assoc Constuct			532	498	1,240	3,876	12,916	u	u
Assoc Plus			3,129	3,300	**25,803**	8,980	104,318	u	u
Assoc Minus			2,415	2,506	**22,609**	8,672	100,775	u	u
Assoc Times			**6,095**	**5,622**	**27,142**	**19,333**	112,329	u	u
Assoc MatMul			**1,589,137,900**	**1,510,271,563**	**68,849,757,411**	**10,391,243,876**	**366,068,652,573**	u	u
			>100,000x	>100,000x	>10,00,000x	>100,000x	>1,000,000x		

Faster Than Dense MatMul

$$\mathbf{A} = \mathbb{A}(I,J,1) \qquad \mathbf{C} = \mathbf{A} + \mathbf{B} \qquad \mathbf{C} = \mathbf{A} \times \mathbf{B}$$

Sparse algorithm boost is readily accessible in high level programming environments via vectorized operations

E=Emulation
u = unsupported

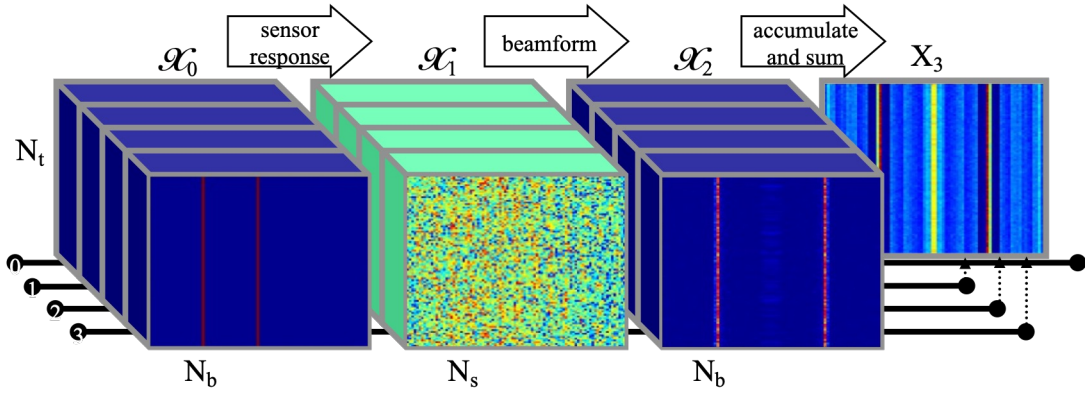
Bold = much better than double MatMul
Blue = best precision

Application: Beamforming (complex matrix multiply)

Accessible
Performance
GigaFlops

	Year	2018	2020	2024	2018	2024	2018	2024
	Language	Matlab	Matlab	Matlab	Matlab	Matlab	Python	Python
	Type	CPU	CPU	CPU	GPU	GPU	GPU	GPU
Benchmark	Precision / HW	xeon-g6	xeon-p8	amd-e9254	v100	h100nvl	v100	h100nvl
Beamformer	FP64 (complex)	2,091	2,516	2,200	13,817	77,149	12,149	69,892
Beamformer	FP32 (complex)	**4,546**	**5,432**	**4,643**	**27,125**	86,048	**26,447**	**218,530**
Beamformer	FP16 (complex)	E	E	E	u	**250,729**	**26,135**	**218,230**
Beamformer	INT16 (complex)	u	u	u	u	u	u	u
Beamformer	FP8 (complex)	u	u	u	u	u	u	u
Beamformer	INT8 (complex)	u	u	u	u	u	u	u

50x



Complex mixed precision performance can be achieved in applications via vectorized operations

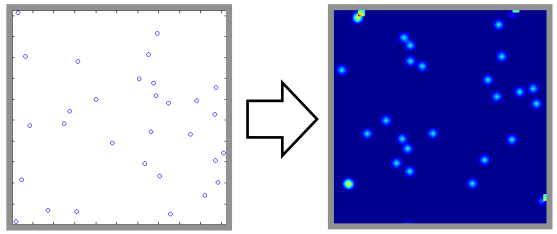
E=Emulation
u = unsupported

Bold = much better than double MatMul
Blue = best precision

Application: Blurimage (2D FFT Convolution)

Accessible
Performance
GigaFlops

Benchmark	Precision / HW	Year: 2018 Language: Matlab Type: CPU xeon-g6	2020 Matlab CPU xeon-p8	2024 Matlab CPU amd-e9254	2018 Matlab GPU v100	2024 Matlab GPU h100nvl	2018 Python GPU v100	2024 Python GPU h100nvl
Blurimage (FFT)	FP64	**17,992**	**21,996**	**79,093**	**216,172**	**538,203**	**356,450**	**1,869,907**
Blurimage (FFT)	FP32	**30,173**	**36,541**	**129,510**	**345,396**	**705,916**	**889,540**	**3,220,154**
Blurimage (FFT)	FP16	**27,502**	**32,894**	**113,965**	u	u	**851,716**	**3,018,172**
Blurimage (FFT)	INT16	**18,261**	**23,194**	**78,058**	**216,017**	**504,129**	**352,583**	**1,803,563**
Blurimage (FFT)	FP8	u	u	u	u	u	u	u
Blurimage (FFT)	INT8	**18,625**	**23,020**	**79,222**	**229,887**	**509,873**	**353,763**	**1,801,938**
Blurimage (FFT)	FP64 (complex)	**60,424**	**74,441**	**265,580**	**881,918**	**2,130,639**	**812,984**	**3,459,935**
Blurimage (FFT)	FP32 (complex)	**118,680**	**138,015**	**481,290**	**1,350,745**	**2,812,656**	**1,623,058**	**6,400,063**
Blurimage (FFT)	FP16 (complex)	**109,527**	**131,764**	**461,739**	u	u	**1,623,904**	**6,359,006**
Blurimage (FFT)	INT16 (complex)	**60,891**	**73,908**	**263,376**	**864,854**	**2,020,847**	**813,222**	**3,489,245**
Blurimage (FFT)	FP8 (complex)	u	u	u	u	u	u	u
Blurimage (FFT)	INT8 (complex)	**62,341**	**76,883**	**271,335**	**878,376**	**2,032,205**	**813,089**	**3,488,037**
		>10x	>10x	>30x	>10x	>10x	>30x	>30x



Faster Than Dense MatMul

Spectral method boost can be achieved in applications via vectorized operations

E=Emulation
u = unsupported

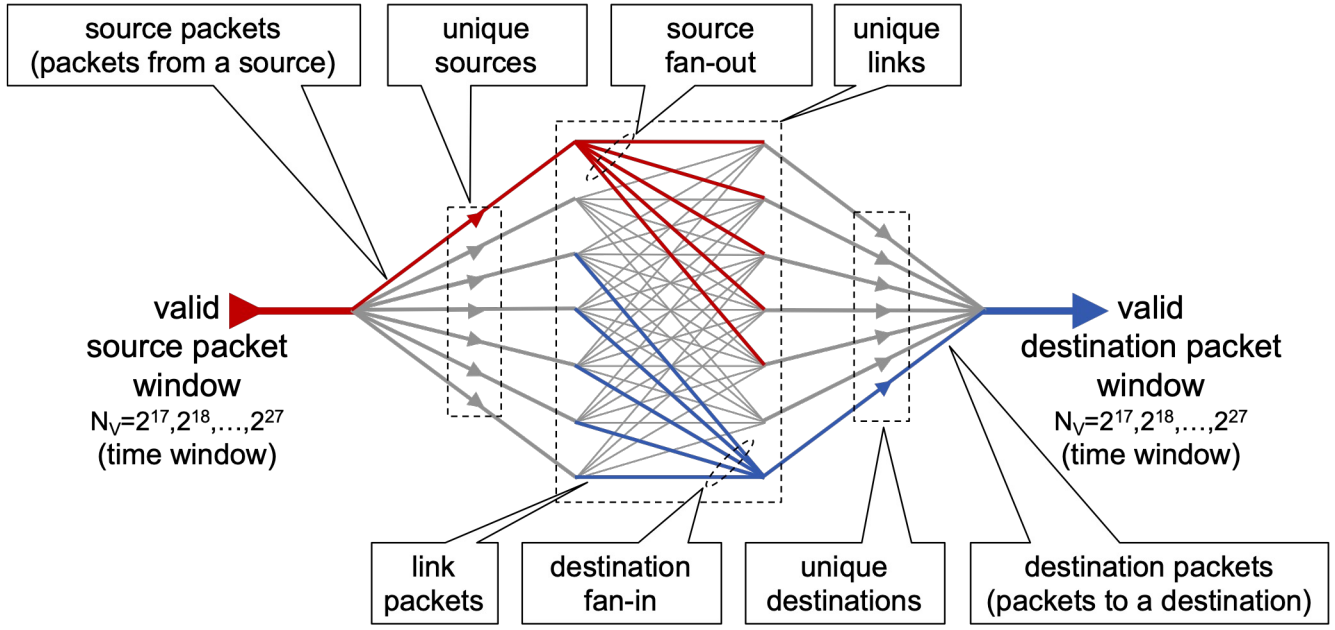
Bold = much better than double MatMul
Blue = best precision

Application: Network Sensing (associative arrays)

Accessible
Performance
GigaFlops

Benchmark	Year / Language / Type / Precision / HW	2018 Matlab CPU xeon-g6	2020 Matlab CPU xeon-p8	2024 Matlab CPU amd-e9254	2018 Matlab GPU v100	2024 Matlab GPU h100nvl	2018 Python GPU v100	2024 Python GPU h100nvl
AnonNetSense	Assoc	**24,181,370,389**	**12,145,394,560**	**17,909,144,186**	**281,593,137,894**	**862,723,041,517**	u	u
		>1,000,000x	>1,000,000x	>1,000,000x	>10,00,000x	>10,000,000x		

Faster Than Dense MatMul



Sparse algorithm boost can be achieved in applications via vectorized operations

E=Emulation
u = unsupported

Bold = much better than double MatMul
Blue = best precision

Summary

Words

"How do I … "

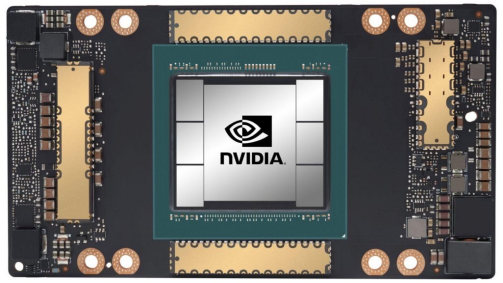
Code

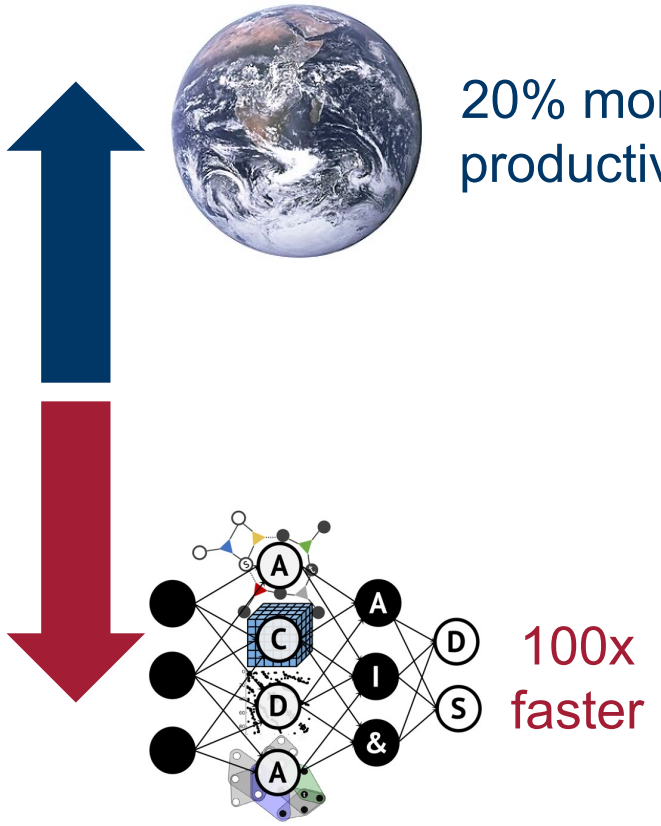
```
for i=1:N
 for j=1:M
  for k=1:L
   C(i,j)=C(i,j)+A(i,k)*B(k,j);
```

Math

$$C = A \times B$$

01101101

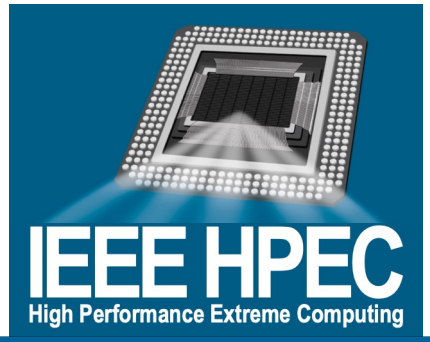




30th IEEE HPEC Virtual Conference
September 14-18, 2026 (ieee-hpec.org)

Papers Due:
July 14

- Premiere conference on High Performance Extreme Computing (850+ participants in 2025)
- 150+ presentations; 2026 Distinguished Speakers:
  - Prof. Saman Amarasinghe (Perkins Prof EECS MIT; ACM Fellow)
  - Prof. Yaneer Bar-Yam (President NECSI)
  - Craig Connors (VP & CTO Cisco Infrastructure & Security)
  - Prof. Danna Freedman (Quantum@MIT Director; MacArthur Fellow)
  - Prof. Sigal Gottlieb (UMass Dartmouth; SIAM & AWM Fellow)
  - Gregory Kurtzer (CEO CIQ & Founder of CentOS & Singularity)
  - Dave Martinez (MIT Lincoln Lab Fellow; IEEE Fellow)
  - Dr. CJ Newburn (Nvidia Distinguished Engineer)
  - Prof. Katsuhisa Ozaki (Shibaura Institute of Technology)
  - Prof. Daniela Rus (CSAIL Director; AAAS, ACM, AAAI, IEEE Fellow)
- 2026 Special Sessions
  - Age of Mixed-Precision
  - Bridging Quantum and HPC
  - Large Language Models: Opportunities and Challenges
  - MIT/Amazon/IEEE Graph Challenge (also due **July 14**)
  - GraphBLAS Forum
  - BRAINS: Building Resilience through Artificial Intelligence for Networked Systems
  - Scaling Research Computing Education



IEEE Boston Section

Sponsors	Thank You	Technical Organizer
NVIDIA MITRE	New England Section of SIAM Boston Chapter of the IEEE Computer Society MIT Student Chapter of SIAM	

- **HPEC focuses on hardware, software, systems, and applications where performance matters**
- **HPEC welcomes experts and people who are new to the field**