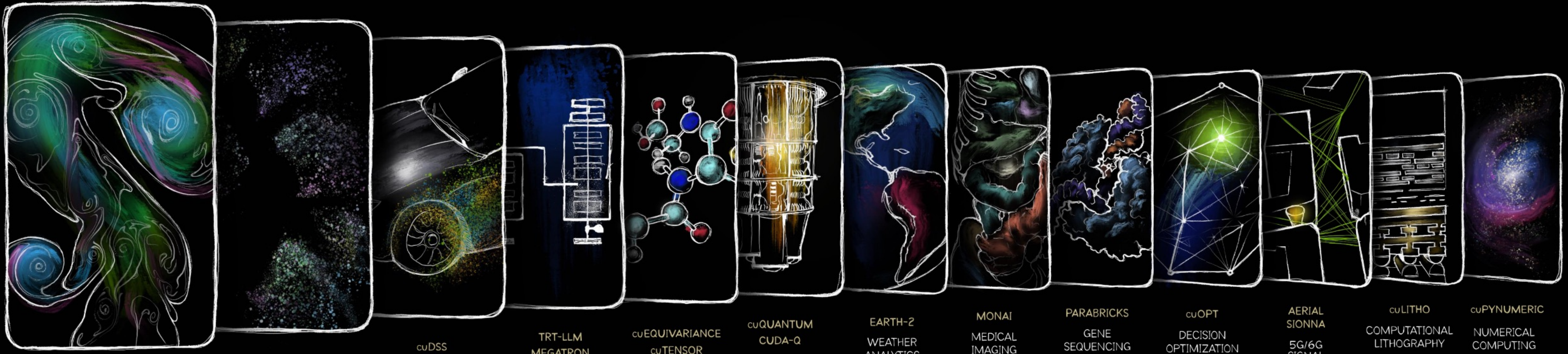
Beyond double precision: AI-driven innovations in HPC offer a quantum leap to scientific computing

Harun Bayraktar, Senior Director – Libraries Engineering

1st International Workshop on Distributed and Parallel Programming for Extreme-scale AI | June 16-17th, 2025 | Paris, France

CUDA-X FOR EVERY INDUSTRY



WARP
PHYSICS

cuDF
cuML
DATA SCIENCE AND PROCESSING

cuDSS
cuSPARSE
cuFFT
AMGX
COMPUTER AIDED ENGINEERING

TRT-LLM
MEGATRON
NCCL
cuDNN
CUTLASS
cuBLAS
DEEP LEARNING

cuEQUIVARIANCE
cuTENSOR
QUANTUM CHEMISTRY

cuQUANTUM
CUDA-Q
QUANTUM COMPUTING

EARTH-2
WEATHER ANALYTICS

MONAI
MEDICAL IMAGING

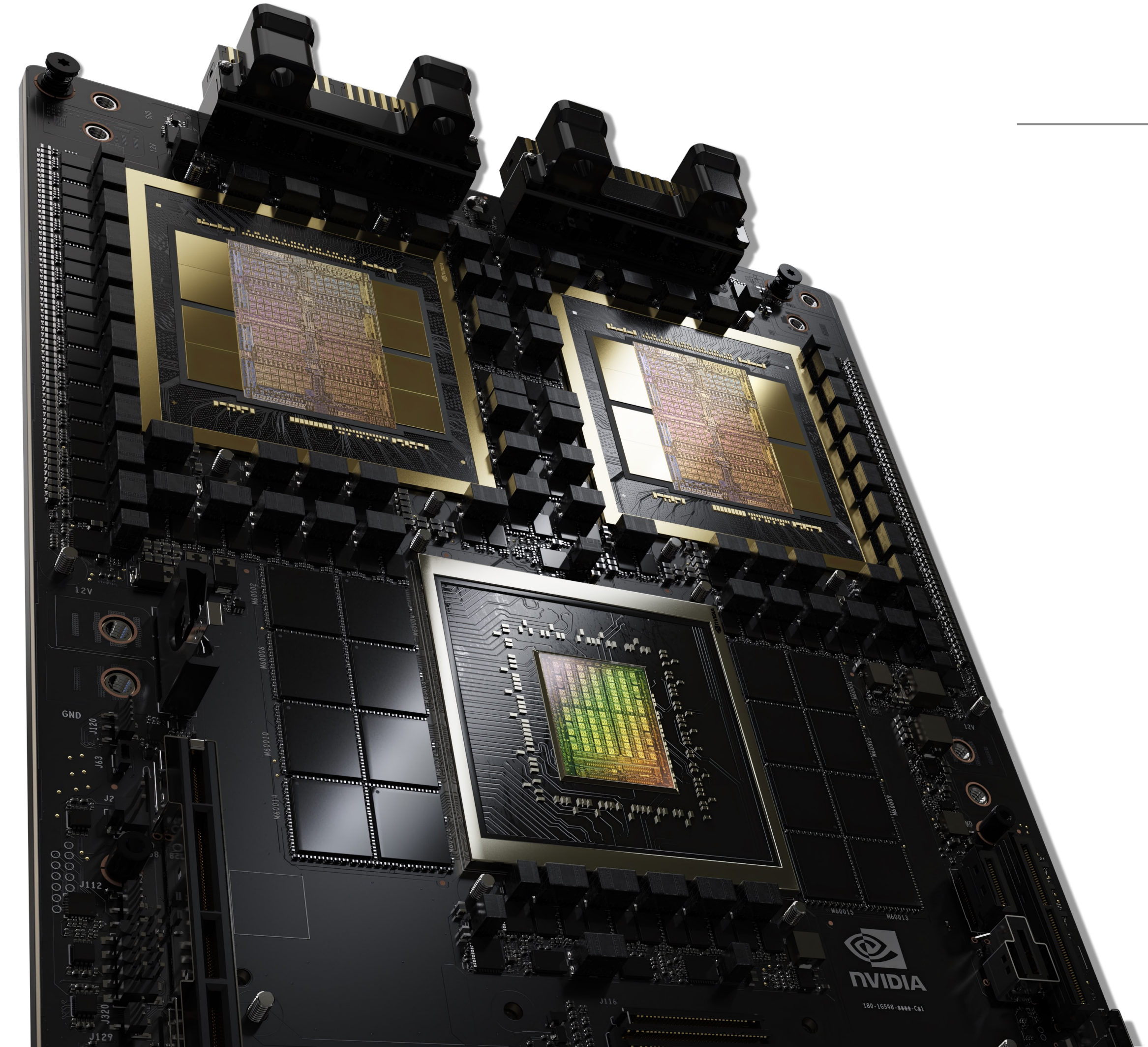
PARABRICKS
GENE SEQUENCING

cuOPT
DECISION OPTIMIZATION

AERIAL
SIONNA
5G/6G SIGNAL PROCESSING

cuLITHO
COMPUTATIONAL LITHOGRAPHY

cuPYNUMERIC
NUMERICAL COMPUTING



Overview

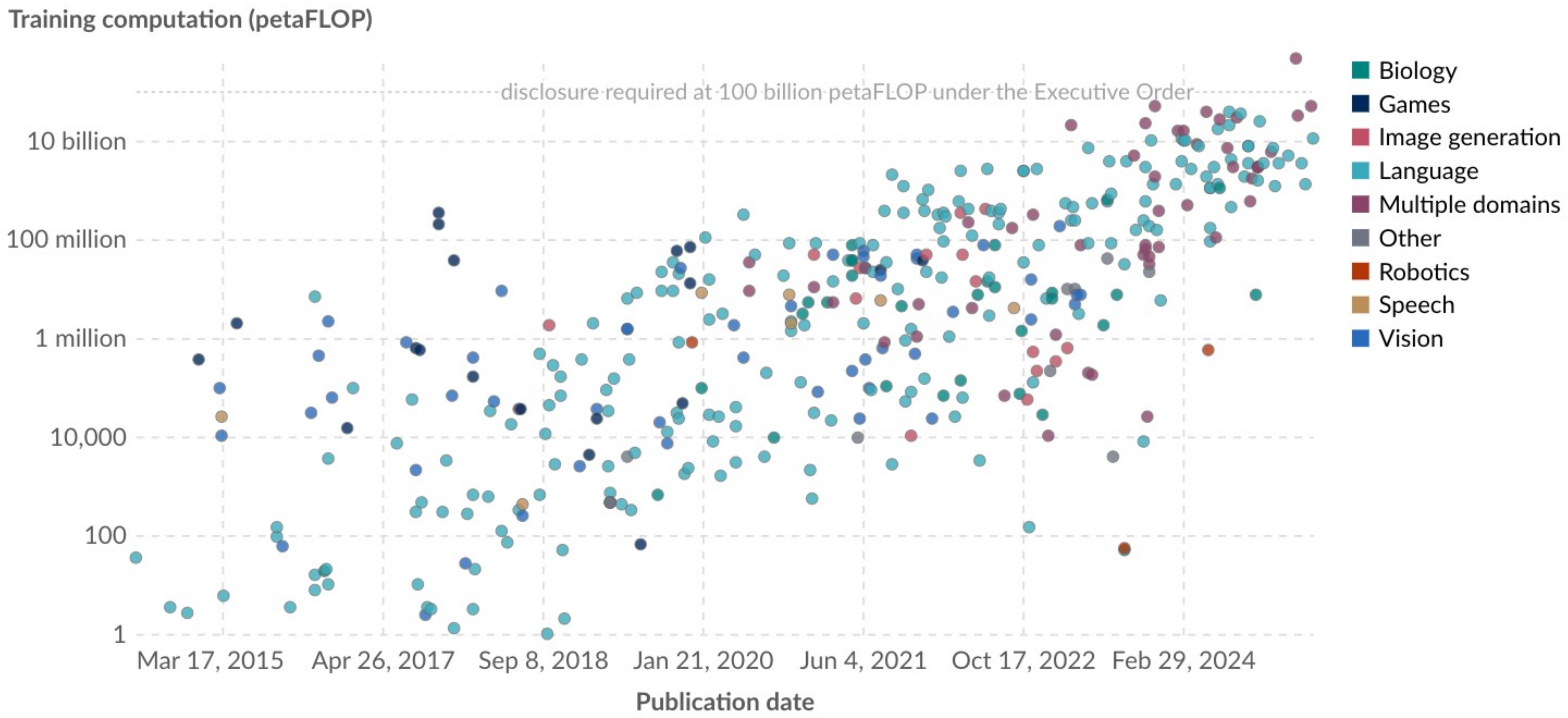
What I would like for you to remember from this talk

- AI has led to significant innovations in mixed-precision (MxP) computing, processor, and system architecture
  - Opportunities and challenges for scientific computing
- We can deliver much higher throughput and efficiency by employing
  - MxP algorithms
  - Floating Point Emulation
- We should not think about AI, HPC, and Scientific Computing as separate things

A Matrix Multiplication Focused Look at Developments that Enabled AI Model Size & Capability Growth

Computation used to train notable AI systems, by domain

Growth over the last decade

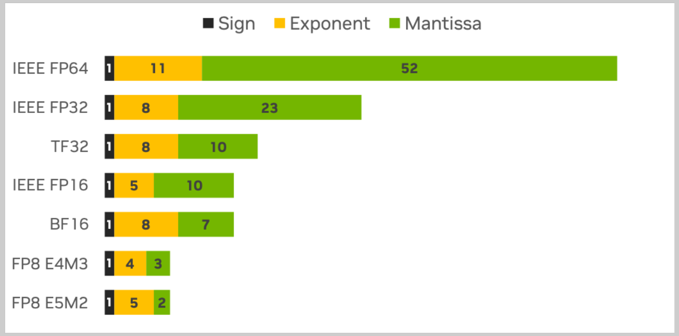


https://ourworldindata.org/artificial-intelligence

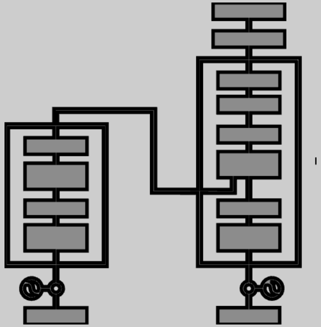
Developments That Made This Possible

A Virtuous Cycle

Reduced & Mixed Precision

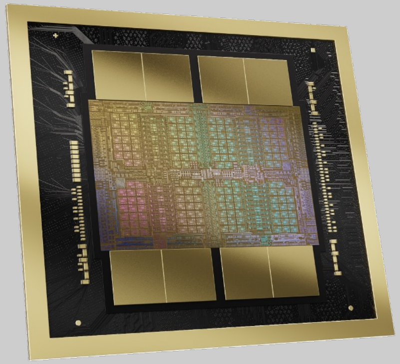


Algorithms

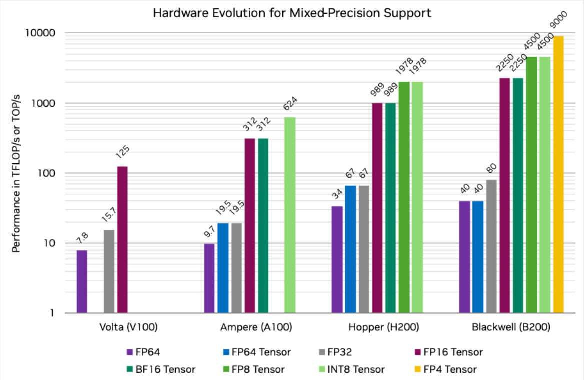


Datacenters

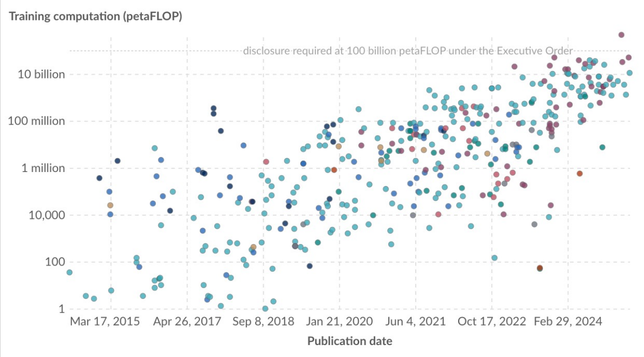
Chip Design & Manufacturing



Performance of a Single GPU



AI Model Growth



Specialized Instructions Amortize Overhead

The Case for Tensor Cores for Matrix Multiplications

9.7.14.4.5. Warp-level Matrix Multiply-and-Accumulate Instruction: `wmma.mma`

`wmma.mma`

Perform a single matrix multiply-and-accumulate operation across a warp

Syntax

```
// Floating point (.f16 multiplicands) wmma.mma
wmma.mma.sync.aligned.alayout.blayout.shape.dtype.ctype d, a, b, c;

// Integer (.u8/.s8 multiplicands) wmma.mma
wmma.mma.sync.aligned.alayout.blayout.shape.s32.atype.btype.s32{.satfinite} d, a, b, c;

.alayout = {.row, .col};
.blayout = {.row, .col};
.shape  =  {.m16n16k16, .m8n32k16, .m32n8k16};
.dtype   = {.f16, .f32};
.atype   = {.s8, .u8};
.btype   = {.s8, .u8};
.ctype   = {.f16, .f32};
```

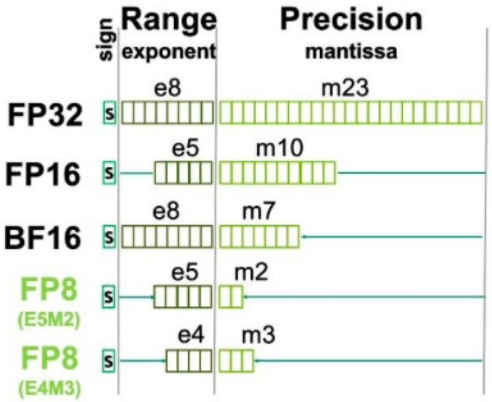
Operation	Energy**	Overhead*
HFMA	1.5pJ	2000%
HDP4A	6.0pJ	500%
HMMA	110pJ	22%
IMMA	160pJ	16%

*Overhead is instruction fetch, decode, and operand fetch – 30pJ
**Energy numbers from 45nm process
Source Bill Dally, Chief Scientist and SVP of Research, NVIDIA Corporation & Adjunct Professor of CS and EE, Stanford
https://www.youtube.com/watch?v=gofI47kfD28
https://cra.org/wp-content/uploads/2024/08/Deep-Learning-Hardware-Session-Slides.pdf

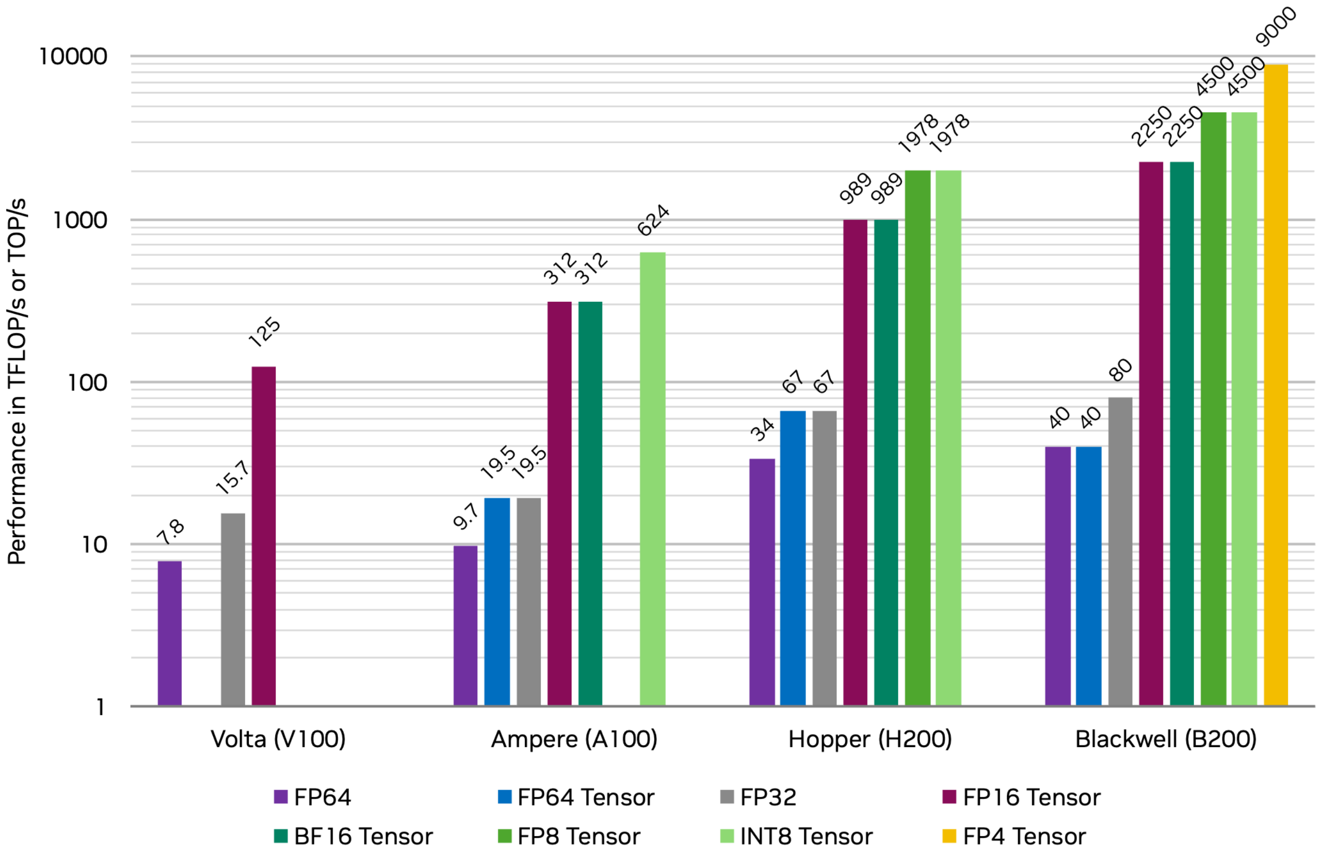
Reduced Precision Floating Point Types

Numerical Concerns in LLMs

- FP8: LLM training and inference for Llama4 most recently, MOEs, …
  - FP8 is either e4m3 or e5m2
    - Allows us be even more compact than before just lik half-floats did at some point
    - Introduced on Hopper with 2x more throughput



- Is that enough though? How can we avoid losing too much of the dynamic range?
  - Key is scaling
  - But scaling can be slow if not done in hardware
  - What kind of scaling do we need?

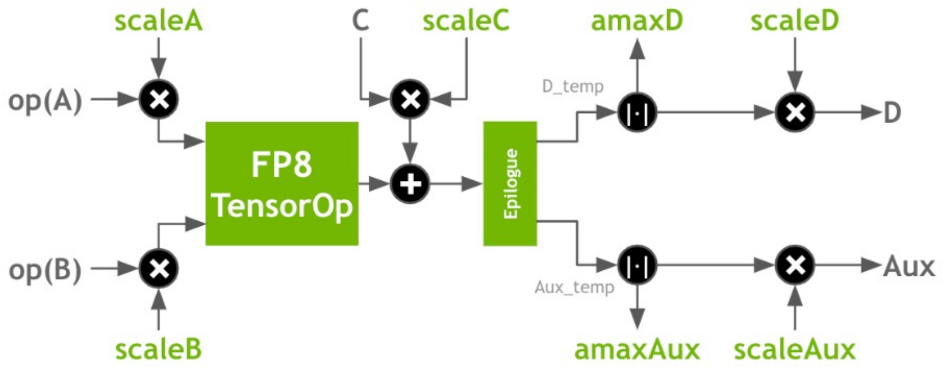


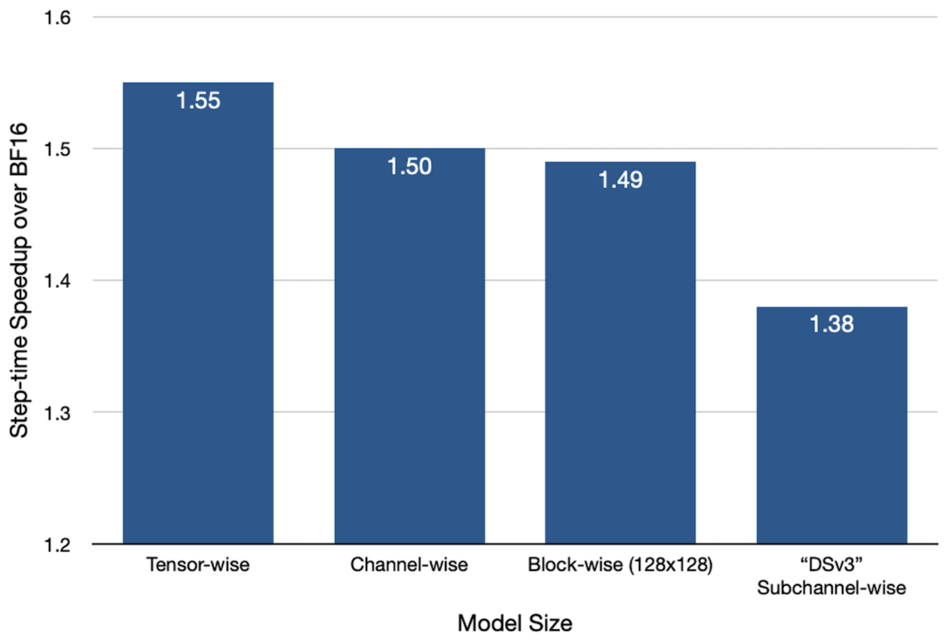
Hardware-accelerated Scaling

Hopper architecture GPUs

- Hopper GPUs introduced tensor-wide HW scaling
  - One problem is that entire matrices A & B are normalized with one scaling factor
  - You need to **either** calculate scaleD at every step of training
    - Remember it's a global value so not tile-based
  - **Or** adopt delayed scaling to pay a smaller price in accuracy but 10% faster e2e

- The main issue is lack of flexibility
  - That why scaling recipes like DeepSeek appeared
  - Or channel-wide/outer vector recipes
  - Obviously on Hopper there is a perf hit
  - Maintaining more control over accuracy is worth it

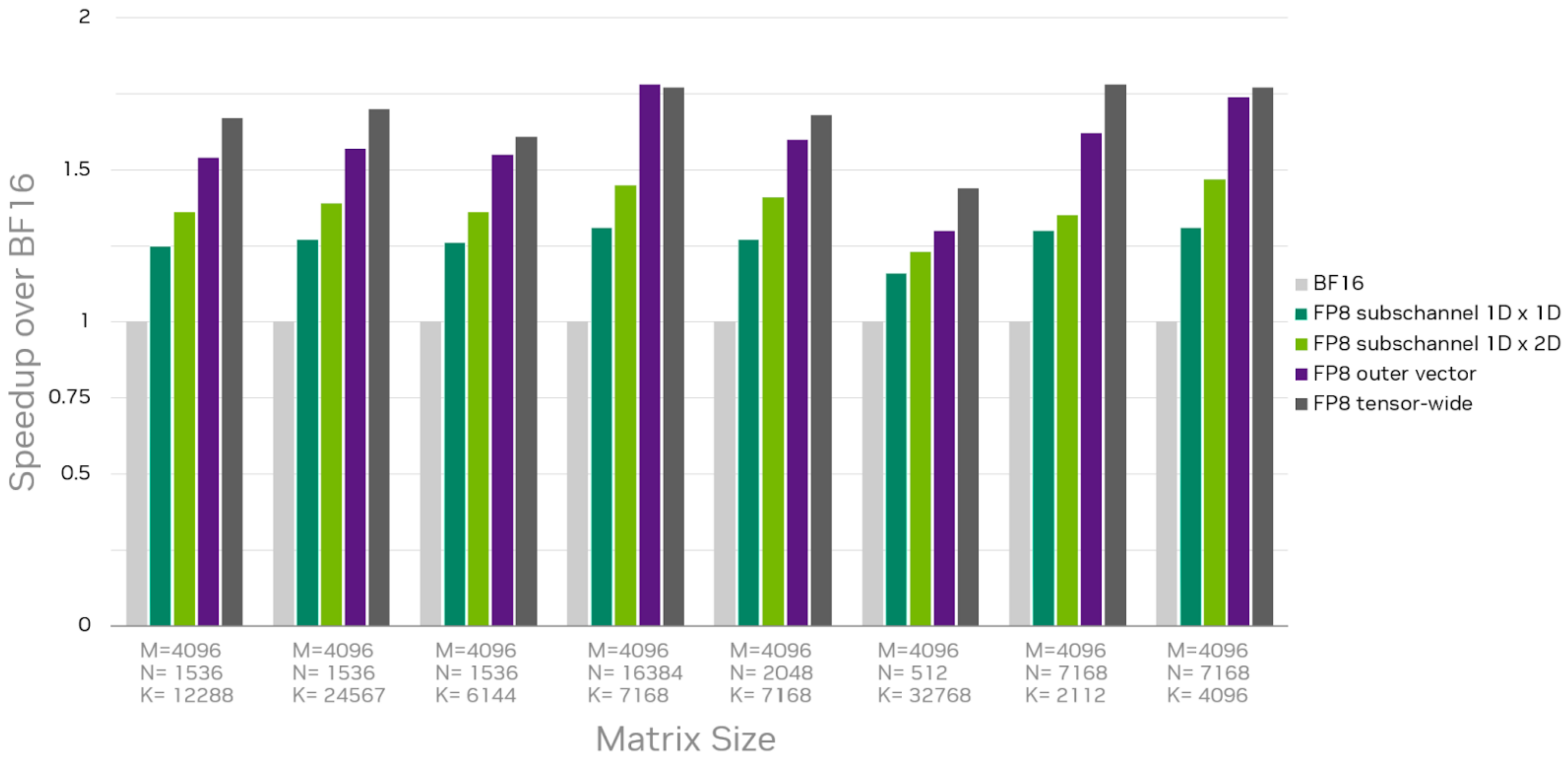




https://developer.nvidia.com/blog/new-cublas-12-0-features-and-matrix-multiplication-performance-on-nvidia-hopper-gpus/

Channel- and block-scaled FP8 matmuls on NVIDIA Hopper

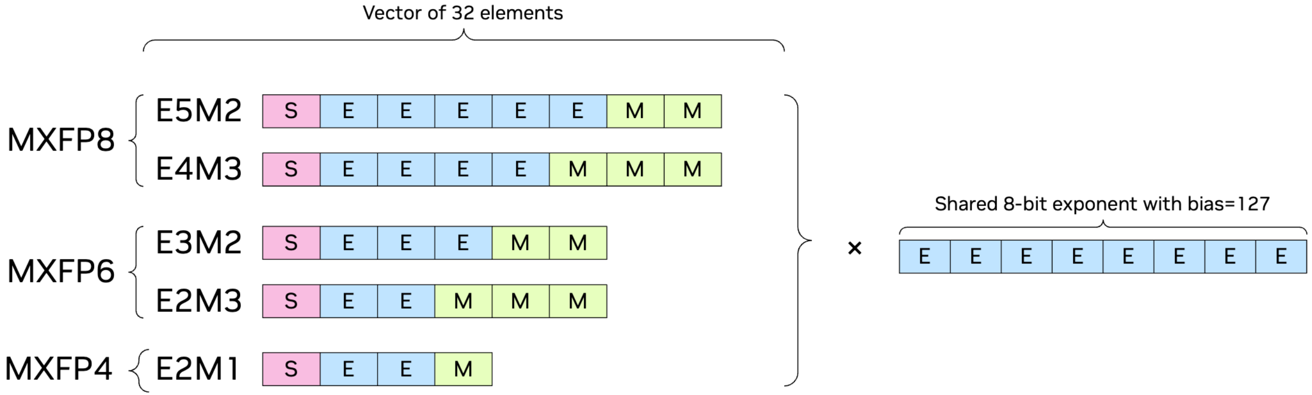
Introduced with cuBLAS in CUDA 12.9

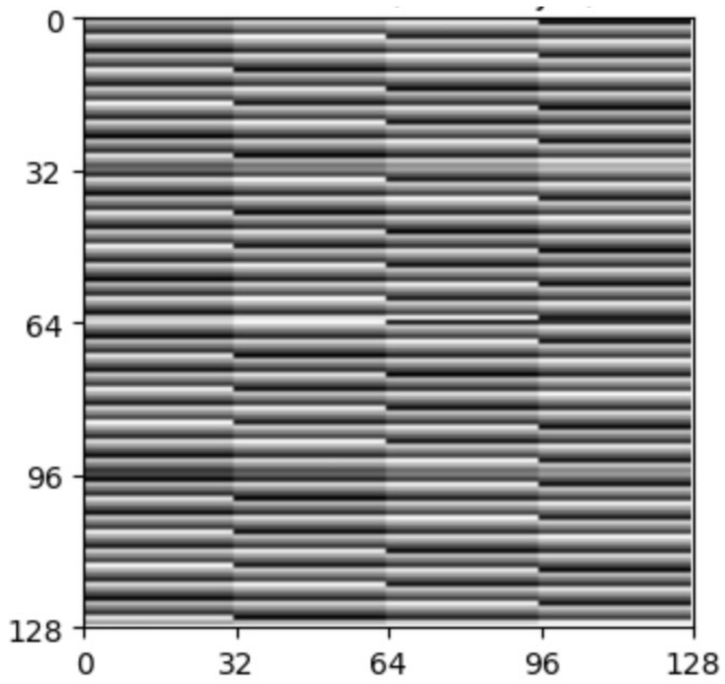


https://developer.nvidia.com/blog/boosting-matrix-multiplication-speed-and-flexibility-with-nvidia-cublas-12-9/

Block-Scaled Floating-Point Types

aka Microscaled Formats



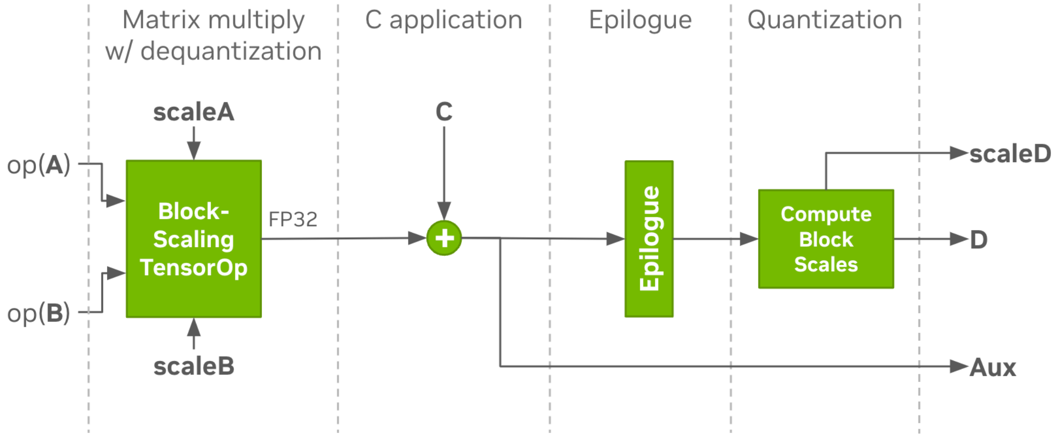


Each 32-element block has
a unique scaling factor

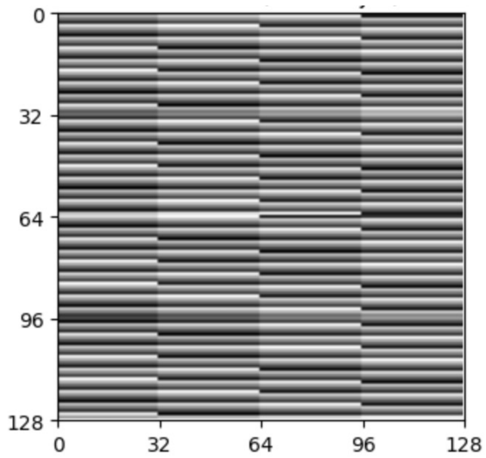
HW-Accelerated Block-Scaling

Blackwell

- **Blackwell** enables FP8 and FP4 with scaling applied to matrix **sub-blocks** [1]
- Block-scaled FP8 is more accurate than tensor scaling
- HW support makes block scaling fast and efficient
- No need to transfer the tensor to GPU global memory before computing the scaling factor
- **No need to use delayed scaling**



Data types:
- **A**, **B**: FP4/FP8
- **C**: FP16/BF16/FP32
- **D**: FP4/FP8 or wider
- **Aux:** BF16
- **Scales** : FP8 flavors



Each 32-element block has a unique scaling factor

API example[2]

```
const __nv_fp8_e8m0 *a_scale = ... /* device pointer */
// set block scaling mode
checkCublasStatus(cublasLtMatmulDescSetAttribute(operationDesc, CUBLASLT_MATMUL_DESC_A_SCALE_MODE, &AScaleMode, sizeof(AScaleMode)));
// set scaling factors
checkCublasStatus(cublasLtMatmulDescSetAttribute(operationDesc, CUBLASLT_MATMUL_DESC_A_SCALE_POINTER, &a_scale, sizeof(a_scale)));
```

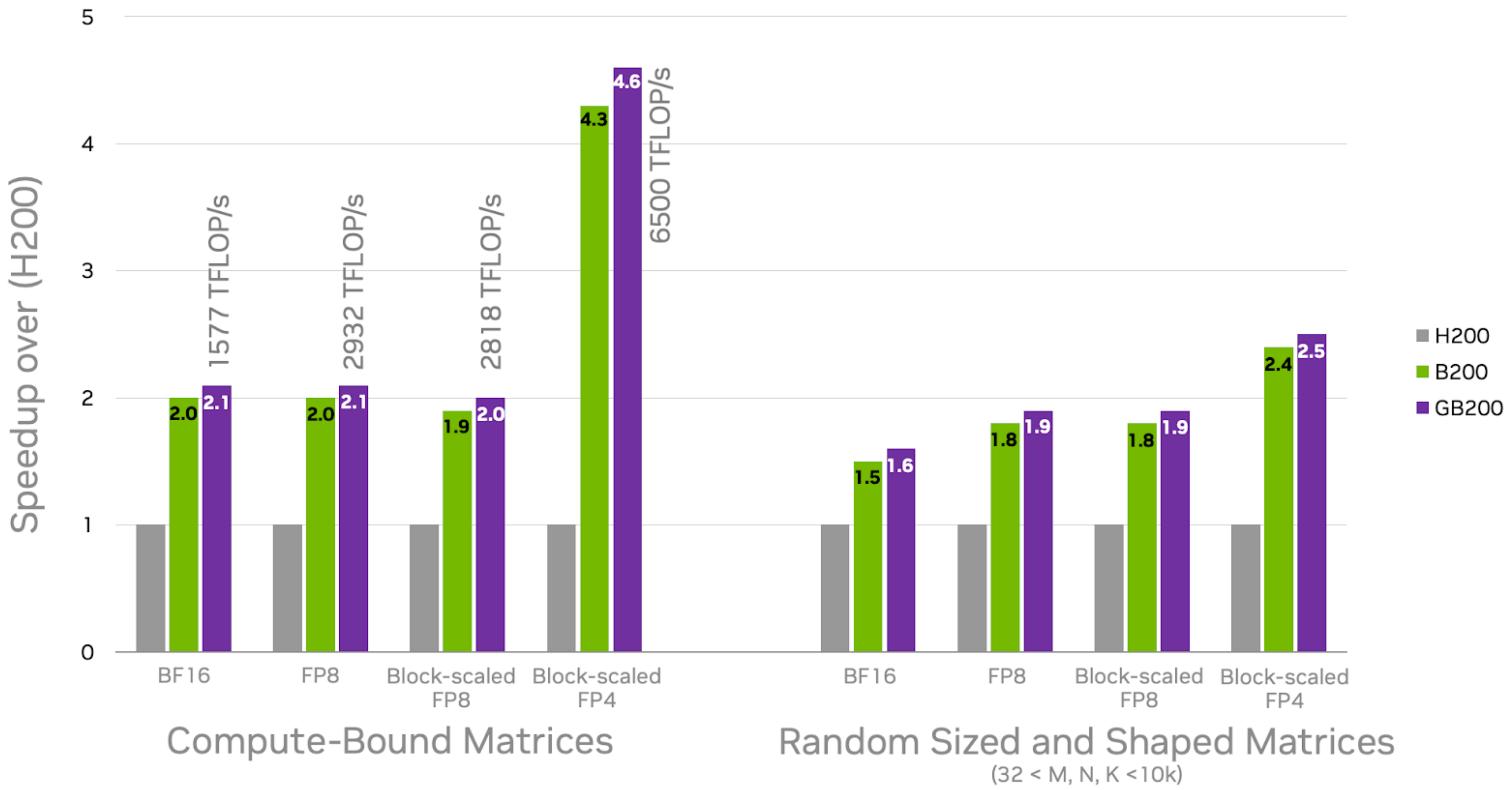
1 cuBLAS documentation https://docs.nvidia.com/cuda/cublas/#d-block-scaling-for-fp8-and-fp4-data-types

2 Samples: https://github.com/NVIDIA/CUDALibrarySamples/blob/master/cuBLASLt/LtMxfp8Matmul/sample_cublasLt_LtMxfp8Matmul.cu

cuBLAS Matmul Performance on Blackwell

Kernels and Runtime Heuristics in cuBLAS Optimized for B200 and GB200

…but what is this?



For block-scaled FP8 and FP4, the baseline on Hopper is chosen to be FP8 with tensor scaling

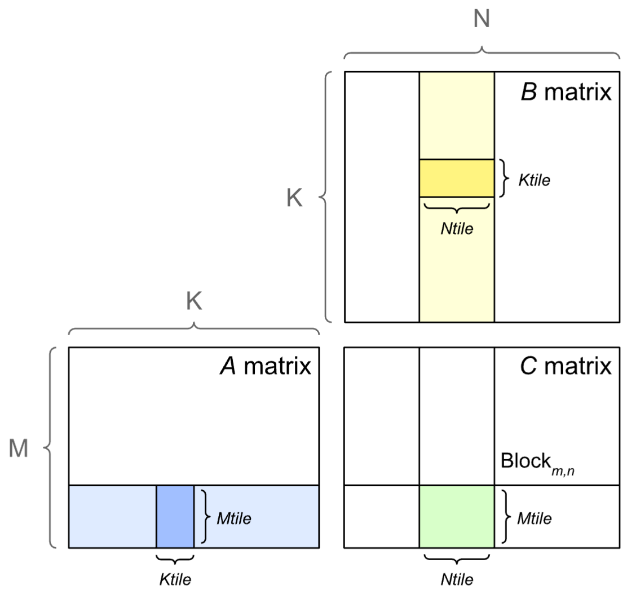
How do we implement kernels for fast matrix multiply on GPUs?

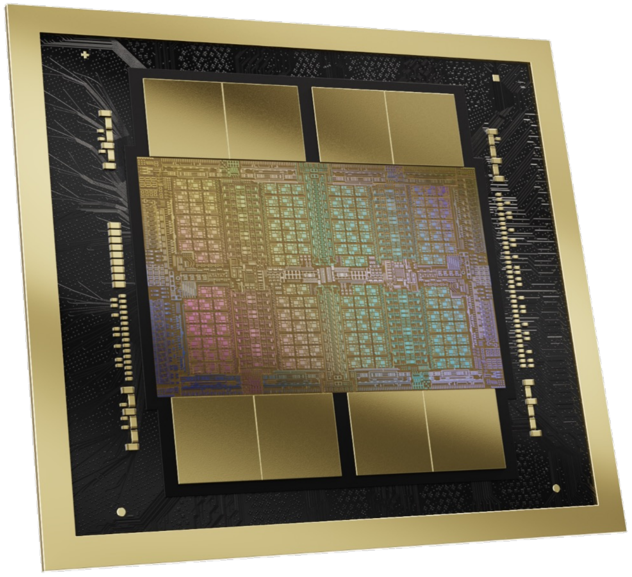
Challenges in Library Design & Development

- Problem space is very large: M, N, K, Batch Size, Transposes, Epilogues, Bias

$$D = Activation(\alpha A^{T?} B^{T?} + \beta C + bias)$$

- We need the maximum performance implementation for each problem





Blackwell B200 GPU

How many kernel options are there on a GPU?

Hopper Generation & Newer

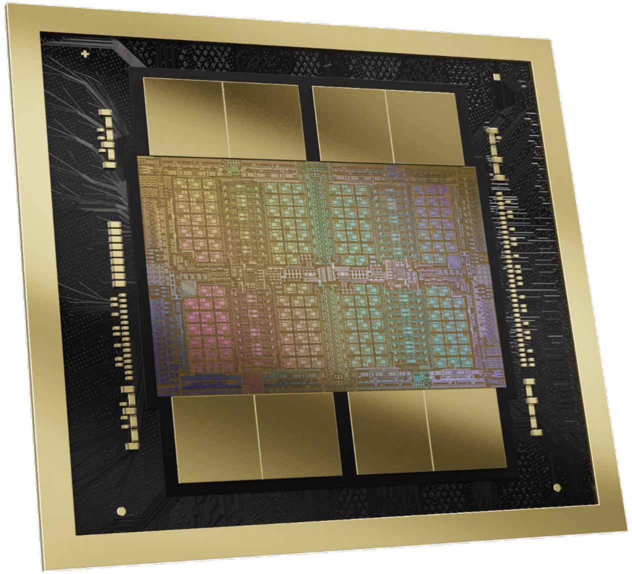
A. 100

B. 1,000

C. 10,000

D. 100,000

E. More than a million



Blackwell B200 GPU

How many kernel options are there on a GPU?

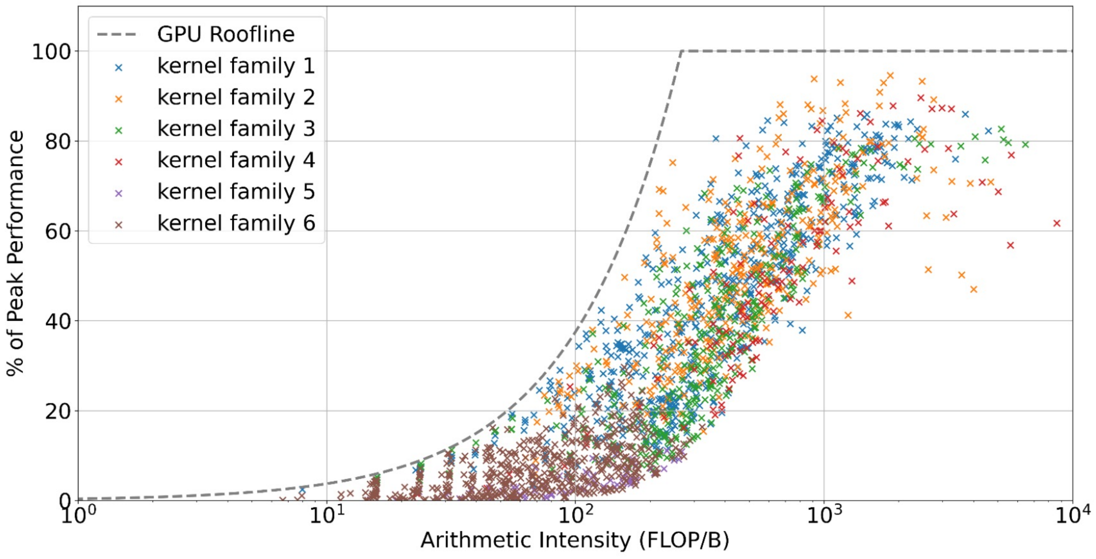
Hopper Generation & Newer

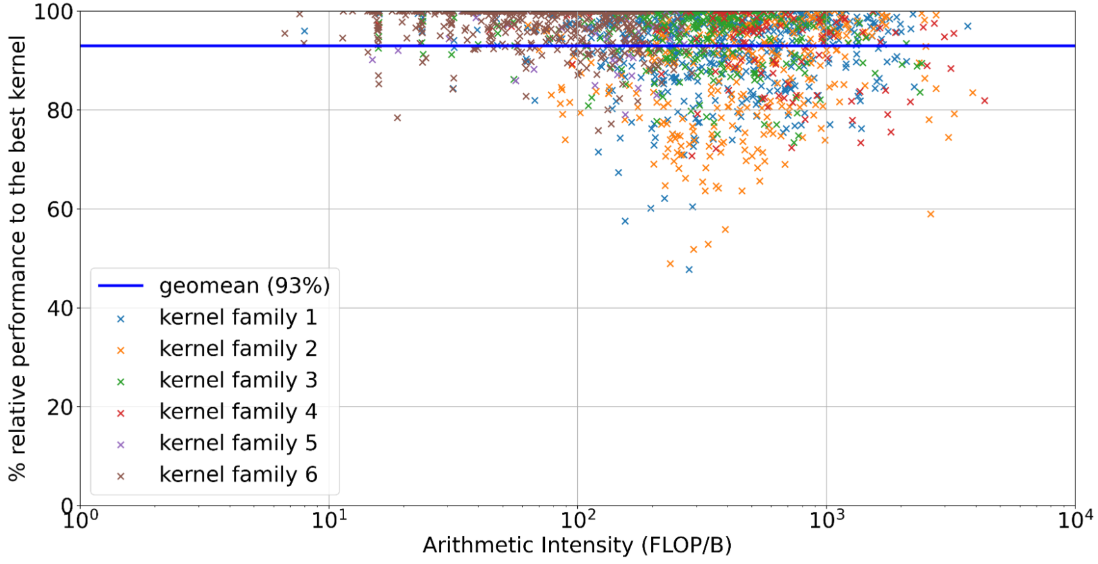
A. 100
B. 1,000
C. 10,000
D. 100,000
✅ E. More than a million

Parameter	# of options
Tile Size	32
Data Stages	32
Thread Block Cluster	50
Split-k	16
CTA Swizzle	3
Possible Combinations	***2,457,600***

Runtime Heuristics Using AI in an AI Library

...or how cuBLAS selects the best kernels from all possible kernels at runtime, and fast!





- cuBLAS ships with lots of kernel families (or implementations)
- Some are more suitable for a specific problem shapes than others
- The library must offer best out-of-the-box perf at launch everywhere

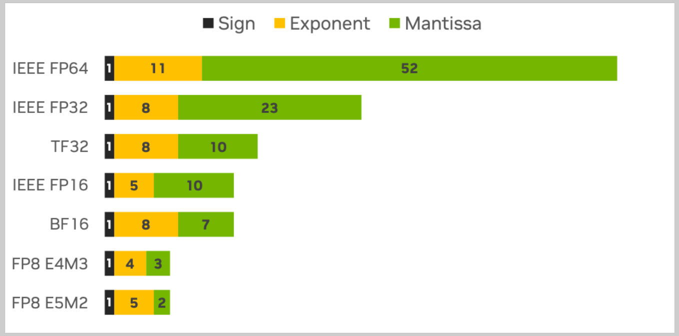
- We train a recommender model to make fast inference at runtime
- Accuracy is guaranteed on an ensemble of problems (in geomean)
- Runtime models can also be analytical
- Autotuning can add extra performance for outliers (example)

More in cuBLAS 12.5 blogpost

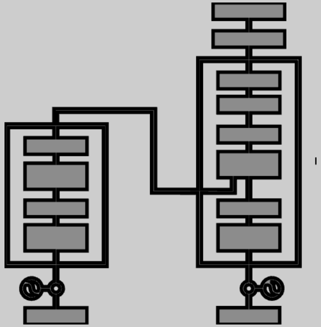
Developments That Made This Possible

A Virtuous Cycle

Reduced & Mixed Precision

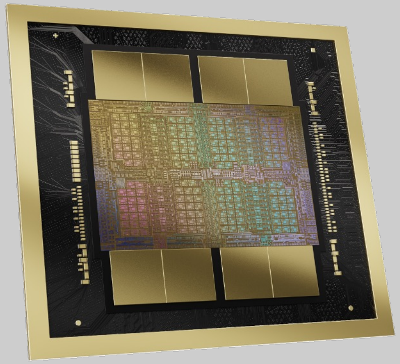


Algorithms

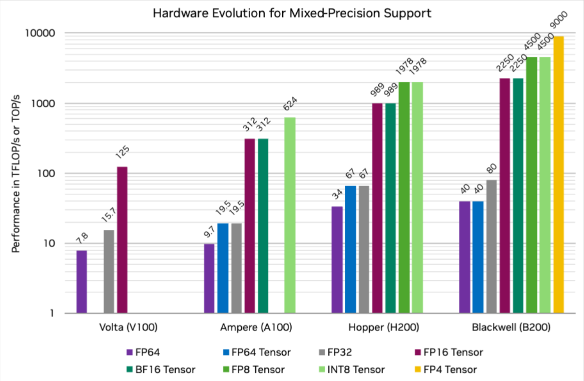


Datacenters

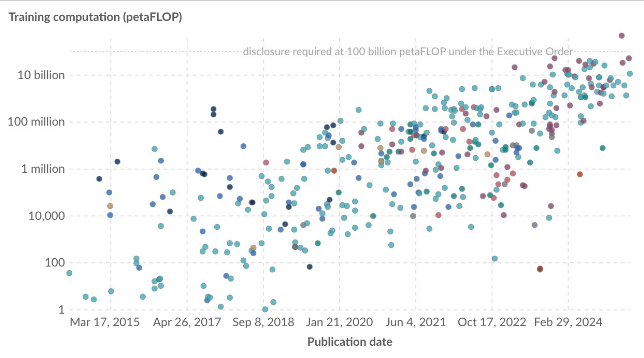
Chip Design & Manufacturing



Performance of a Single GPU



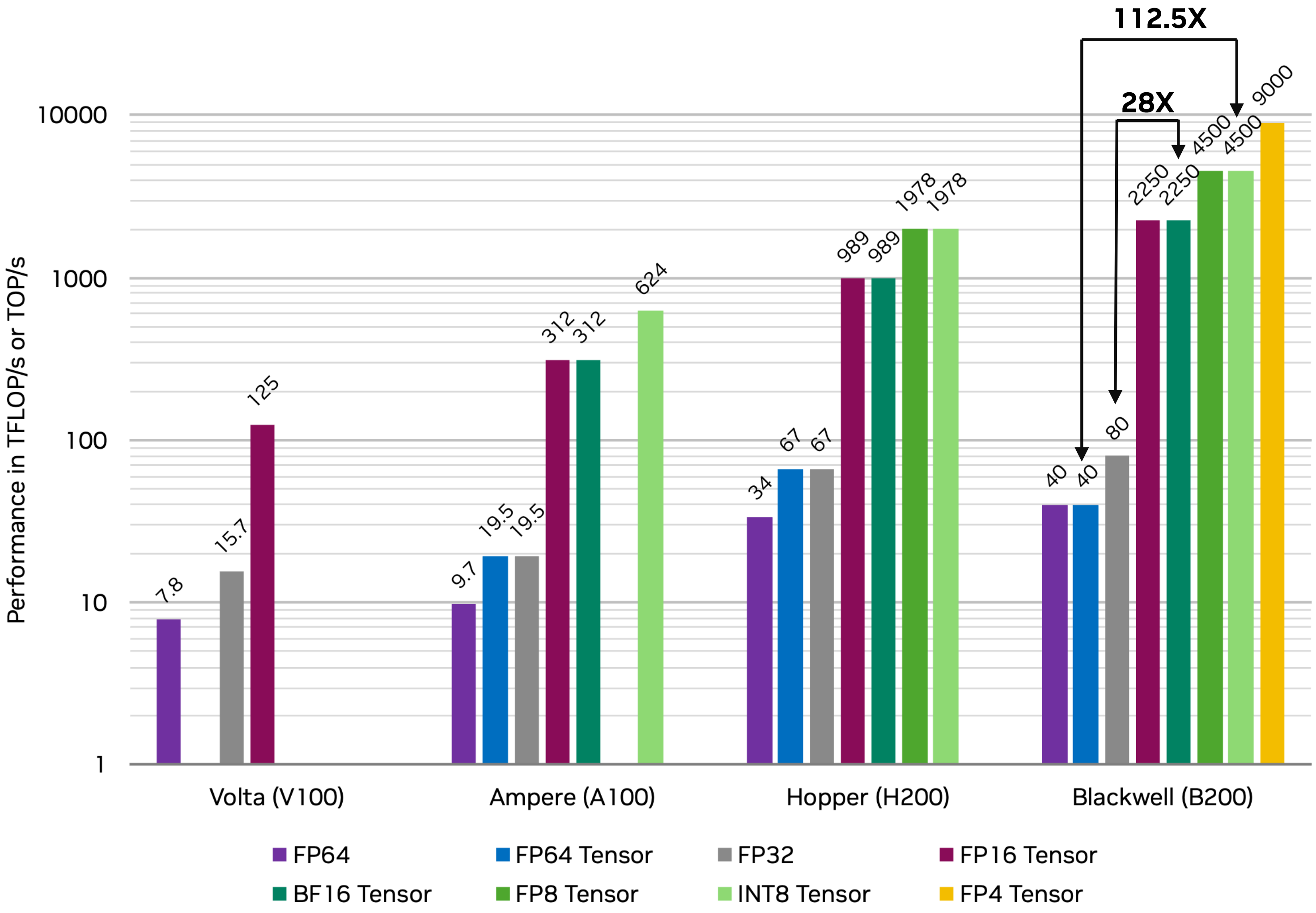
AI Model Growth



Mixed-Precision Computing & Floating-Point Emulation

Single GPU Matrix Multiplication Performance

Tensor Core Throughput Over GPU Generations Since Introduction



How can we leverage higher performance and energy-efficient hardware?

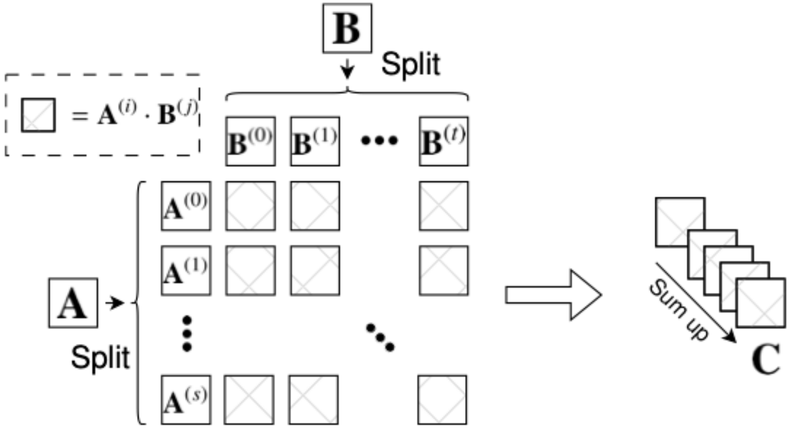
Two different approaches

- **Mixed-precision algorithms** (e.g., iterative refinement solvers, HPL-MxP) result in great performance and efficiency gains
  - Fundamentally different numerics and significant adoption challenges due to the need to invest in these algorithmic changes

- **Emulation of FP64 and FP32 matrix multiplies** without sacrificing accuracy for a performance gain and without requiring any code changes
  - Requires robust implementations and community acceptance even if code changes are not required

Data: An $n \times n$ matrix A, and size n vector b.
Result: A solution vector $x^{(i)}$ approximating x in $Ax = b$, and an LU factorization of $A = LU$.
(`FP16`) Solve $Ax^{(1)} = b$ using `FP16` LU factorization and triangular solve;
$i \leftarrow 1$;
repeat
  (`FP64`) Compute residual $r^{(i)} \leftarrow Ax^{(i)} - b$;
  (Low Precision) Solve $Ac = r^{(i)}$ using
    `IR`: `FP16` triangular solve using the LU factors, casting c to `FP64`, or
    `IRGM`: `FP64` GMRES preconditioned by $M = LU$;
  (`FP64`) Update $x^{(i+1)} = x^{(i)} - c$;
  $i \leftarrow i + 1$;
until $x^{(i)}$ *is accurate enough*;

https://dl.acm.org/doi/10.1109/SC.2018.00050



https://arxiv.org/abs/2306.11975

Full FP64 Precision on FP16 Tensor Cores

LU-factorization algorithm[1] using low-precision tensor cores to give an IEEE-754 FP64 accurate result

Input 64-bit matrix A^{64} for LU factorization

Convert precision $A^{64} \rightarrow A^{16}$

LU factorization of A^{16} gives solution x_0

Start by doing the compute-intensive factorization in FP16

Convert x_0 to 64-bit precision

Compute residual $r_i = b - Ax_i$ in FP64

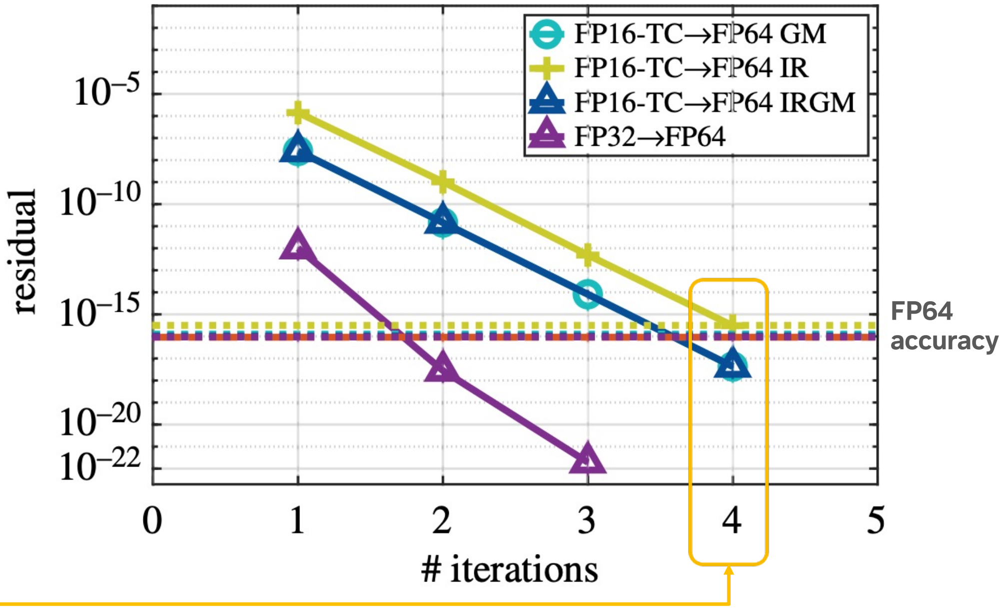
Solve $U^{-1}L^{-1}Ac_i = U^{-1}L^{-1}r_i$ by GMRES

Iterative step refines accuracy of result to desired precision

Correct current solution $x_{i+1} = x_i + c_i$ in FP64

repeat until x_i is accurate enough

Output is IEEE-754 accurate FP64 result

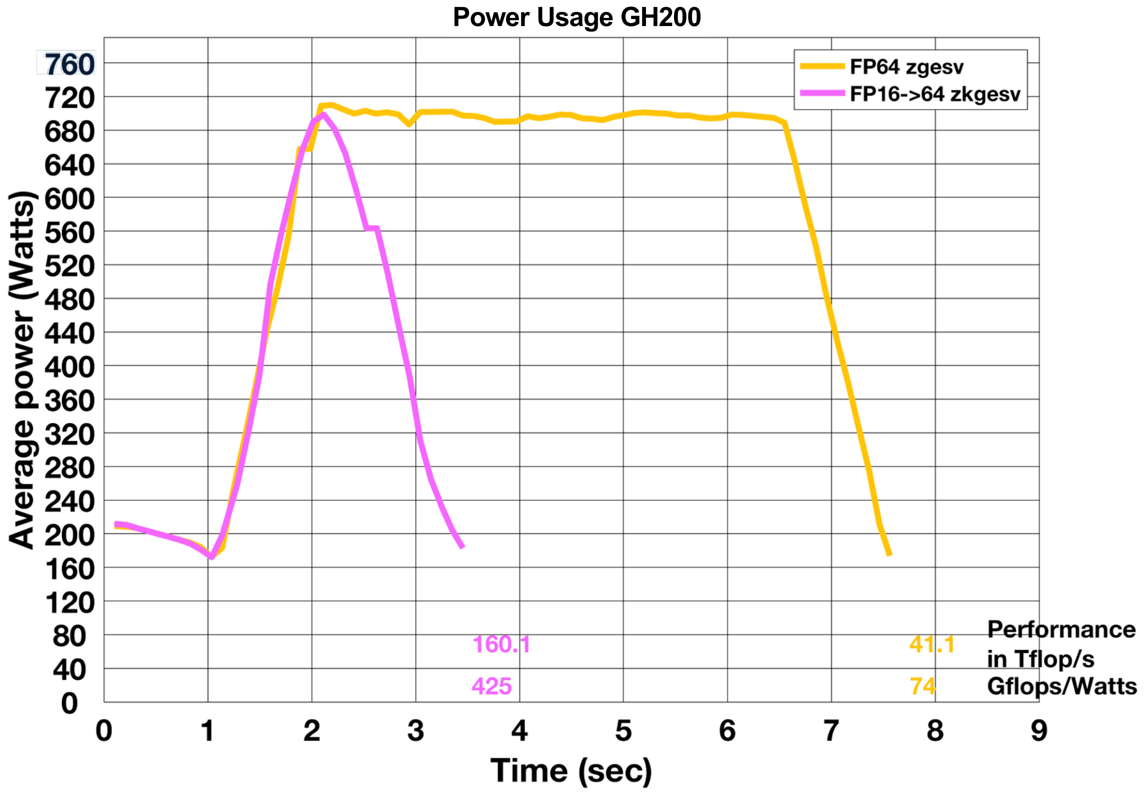


[1] Haidar Azzam, Bayraktar Harun, Tomov Stanimire, Dongarra Jack and Higham Nicholas J., 2020, Mixed-precision iterative refinement using tensor cores on GPUs to accelerate solution of linear systems, Proc. R. Soc. A.4762020011020200110 https://royalsocietypublishing.org/doi/10.1098/rspa.2020.0110

Eating your cake and having it too: Both Power and Performance

	FP64	FP16+FP64	Mixed Precision Benefit
Performance (TFLOP/s)	41.1	160.1	3.9x
Perf/Watt (GFLOPs/Watts)	74	425	5.7x

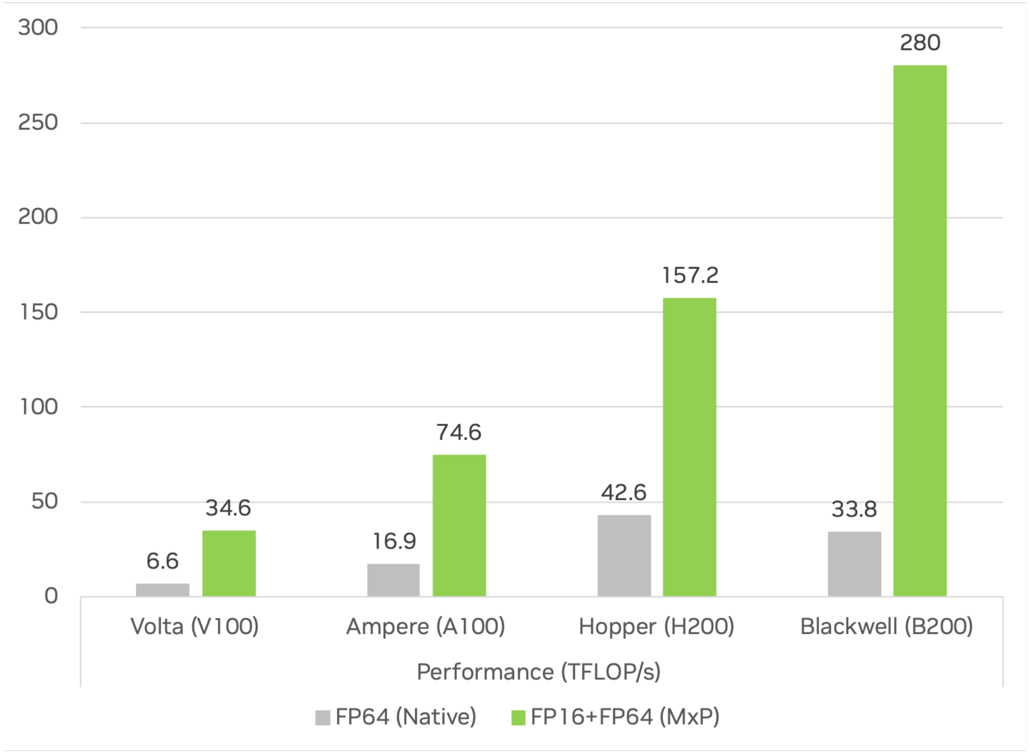
Performance and Perf/Watt improvement when using FP16 tensor cores with iterative refinement method

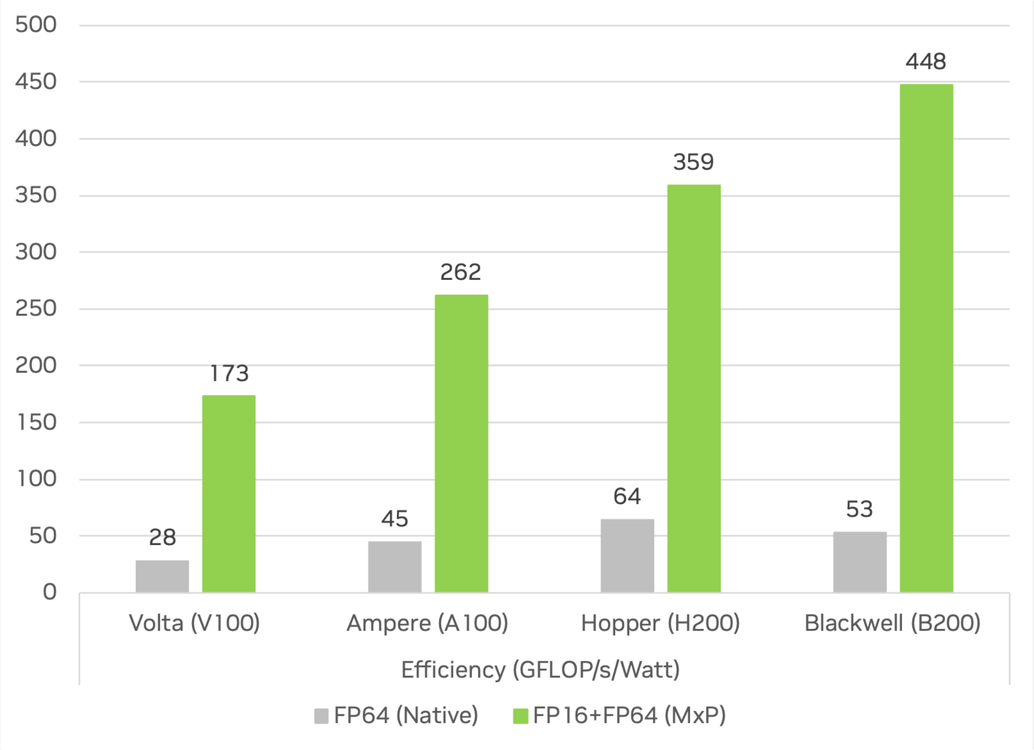


Complex matrix solve via LU decomposition
ZGESV on GH200, n=44000

Mixed-Precision Iterative Refinement Solver

Performance and Efficiency Improvements Across Four GPU Generations



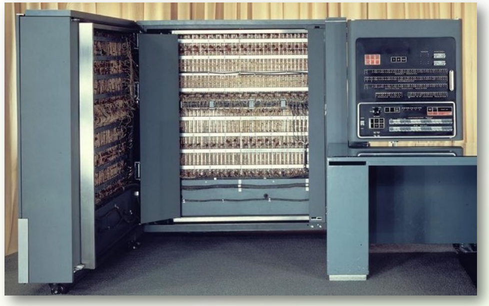


Historical Perspective on Emulation

The evolution of floating-point (FP) computation

1950s
Simulated floating-point arithmetic utilizing fixed-point representations

IBM 701 Speedcoding System

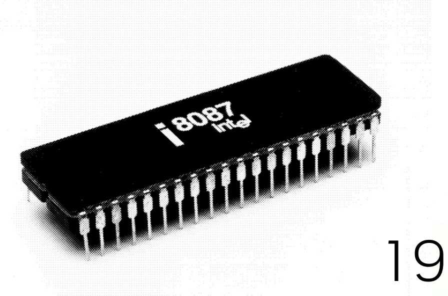


1960s-70s
IBM Hexadecimal FP
Cray FP

Diversity in representations

1980
FPU Coprocessor

Intel 8087

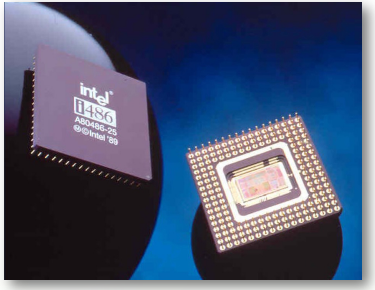


1985
IEEE 754

Standardization of FP

1989
Integrated FPU

Intel i486

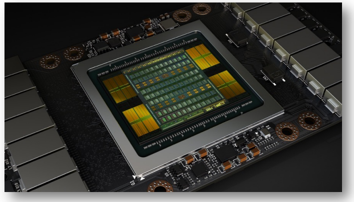


2001
GPUs with programmable shaders

NVIDIA GeForce3

2017+
GPU Tensor Cores introduced for for reduced & mixed-precision

FP16, BF16, TF32, FP8, NVFP4, MXFP8

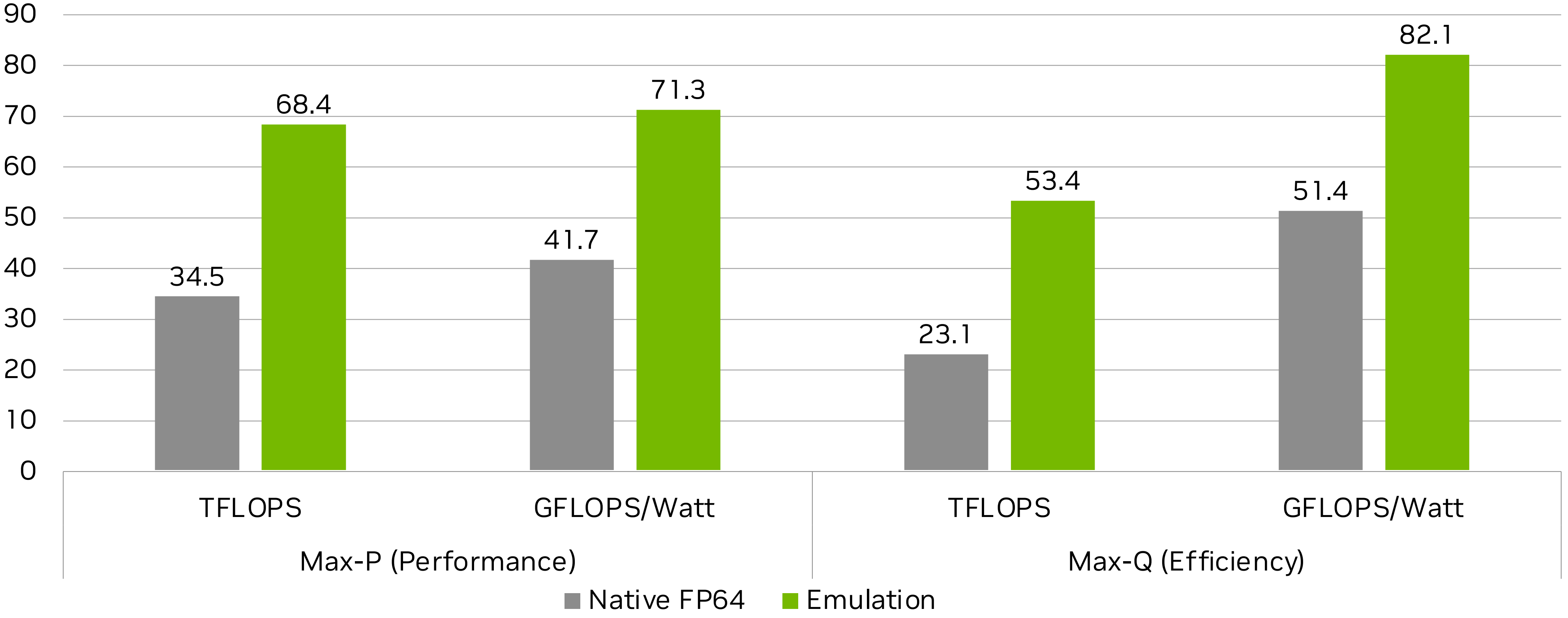


Today
FP emulation returns (e.g., Ozaki-I & II)

GPU Tensor Cores Accelerated Matrix Multiplication using AI FP types

Performance and Perf/Watt: Emulated vs. Native HPL

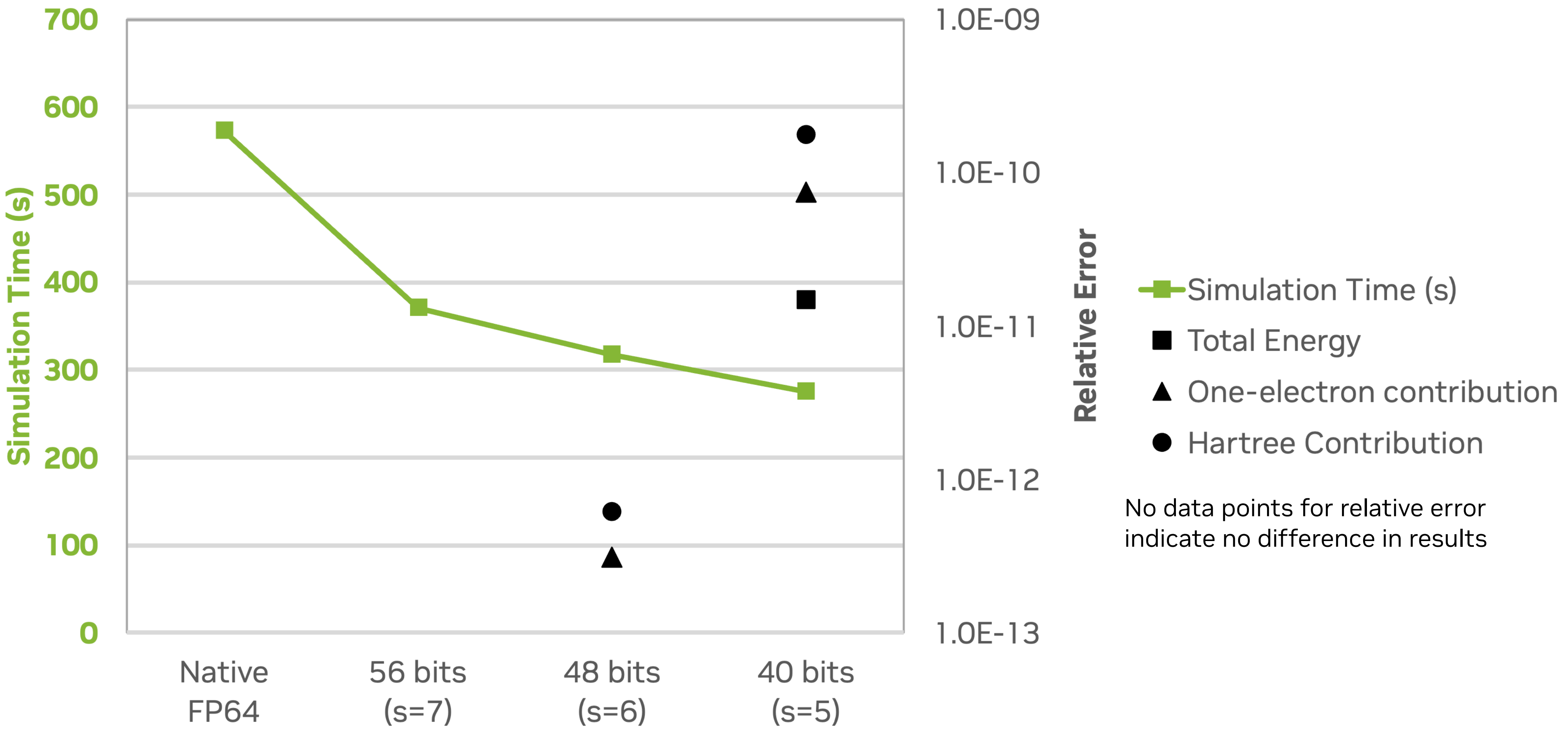
At Max-P (Performance) Blackwell HPL runs 2.0x faster and 1.7x more efficiently using emulation (55 bits)
At Max-Q (Efficiency) Blackwell HPL runs 2.3x faster and 1.6x more efficiently using emulation (55 bits)



Double Precision Emulation using Ozaki-I Scheme
https://arxiv.org/abs/2409.13313
https://arxiv.org/abs/2306.11975

Quantum Espresso Performance With FP Emulation (Ozaki-I)

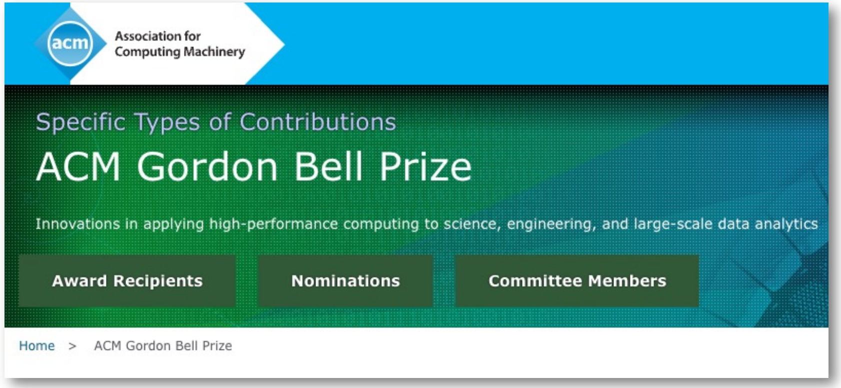
AuSurf Benchmark on Blackwell RTX (Low Native FP64)



Reduced, Mixed-Precision, and AI for Scientific Computing

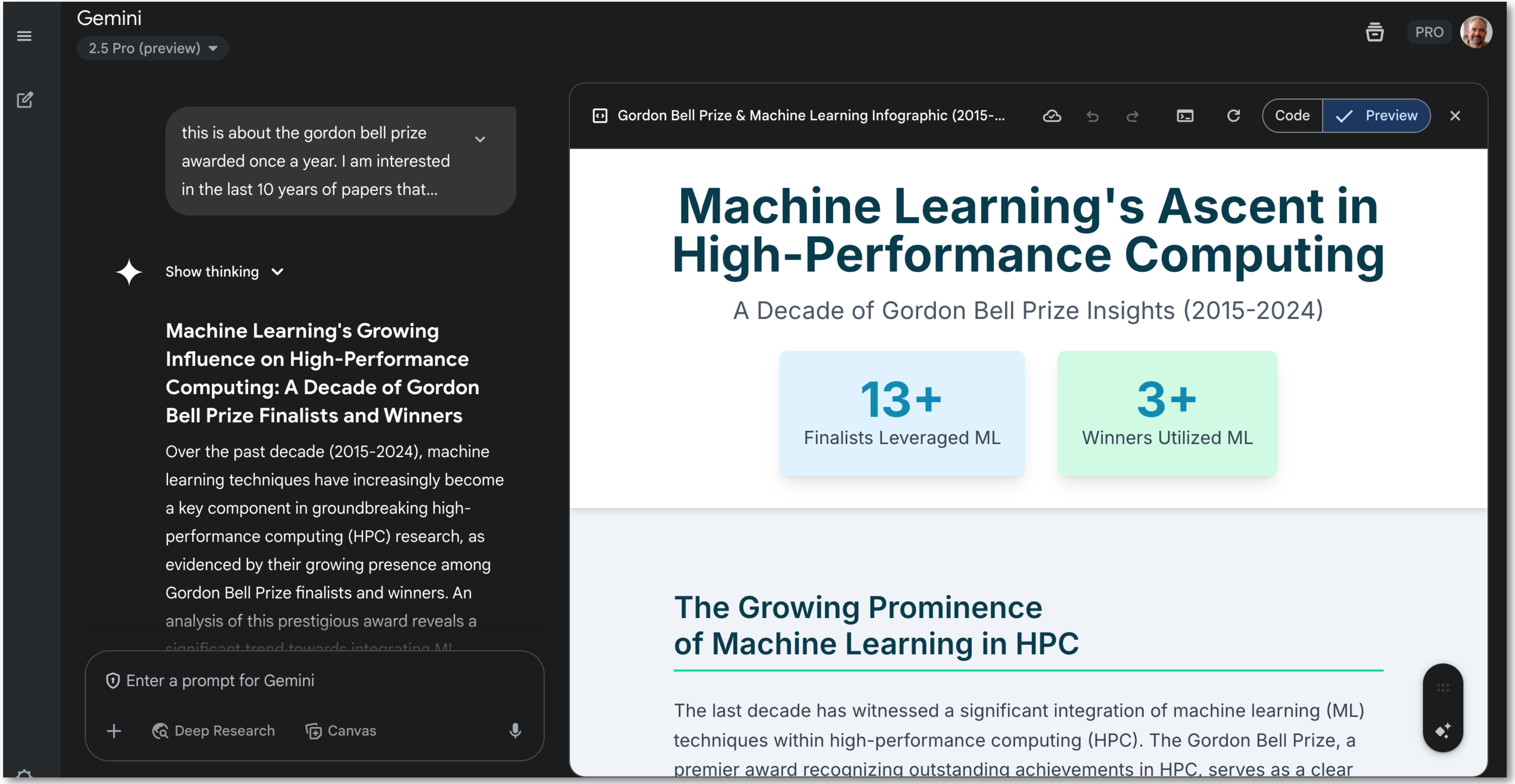
Some Statistics From the Gordon Bell Finalists

Looking at Reduced/Mixed-Precision Usage in the Last 10 Years



Year	Total # of Finalists	# Finalists w/ Reduced/MxP	Maximum Performance	# of Winners	# Winners w/ Reduced/MxP
2015	5	2	2.0PF	1	0
2016	6	1	?	1	0
2017	3	2	18.9PF	1	1
2018	6	6	1.3EF	2	2
2019	2	2	90.9PF	1	1
2020	6	5	136PF	1	1
2021	6	3	4.4EF	1	1
2022	6	2	1.0EF	1	0
2023	6	4	0.66EF	1	1
2024	6	4	1.8EF	1	0
Total	***52***	***31***		***11***	***7***

60%!


Gemini
2.5 Pro (preview)
PRO
this is about the gordon bell prize awarded once a year. I am interested in the last 10 years of papers that...
Show thinking
Machine Learning's Growing Influence on High-Performance Computing: A Decade of Gordon Bell Prize Finalists and Winners
Over the past decade (2015-2024), machine learning techniques have increasingly become a key component in groundbreaking high-performance computing (HPC) research, as evidenced by their growing presence among Gordon Bell Prize finalists and winners. An analysis of this prestigious award reveals a
Enter a prompt for Gemini
Deep Research
Canvas
Gordon Bell Prize & Machine Learning Infographic (2015-...
Code
Preview
Machine Learning's Ascent in High-Performance Computing
A Decade of Gordon Bell Prize Insights (2015-2024)
13+
Finalists Leveraged ML
3+
Winners Utilized ML
The Growing Prominence of Machine Learning in HPC
The last decade has witnessed a significant integration of machine learning (ML) techniques within high-performance computing (HPC). The Gordon Bell Prize, a

AlphaFold

The Nobel Prize in Chemistry 2024

They cracked the code for proteins' amazing structures

The Nobel Prize in Chemistry 2024 is about proteins, life's ingenious chemical tools. David Baker has succeeded with the almost impossible feat of building entirely new kinds of proteins. Demis Hassabis and John Jumper have developed an AI model to solve a 50-year-old problem: predicting proteins' complex structures. These discoveries hold enormous potential.

Related articles

- Press release
- Popular information: They have revealed proteins' secrets through computing and artificial intelligence
- Scientific background: Computational protein design and protein structure prediction

David Baker

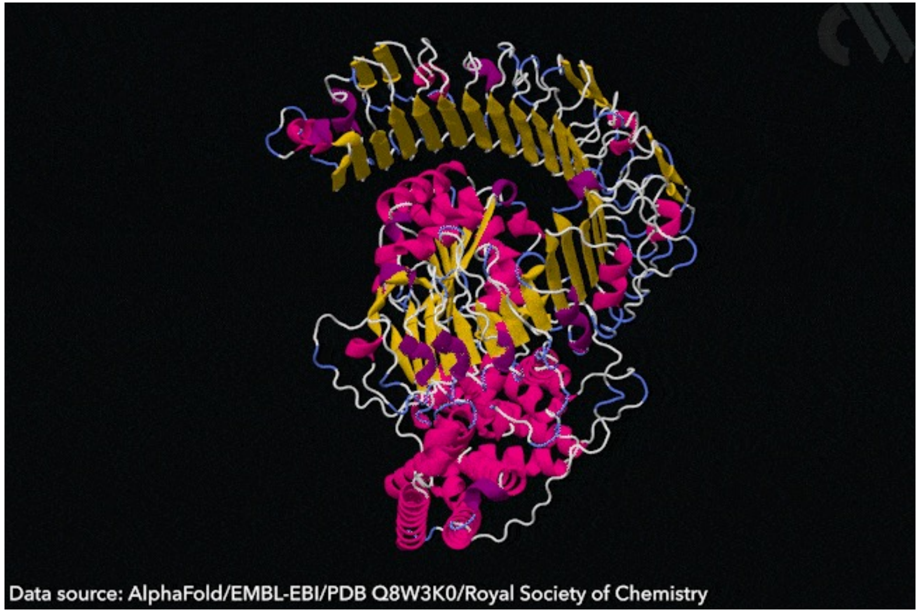
"for computational protein design"

Demis Hassabis

"for protein structure prediction"

John Jumper

"for protein structure prediction"



On the topic of HPC & AI

Convergence or divergence?

Post

Daniel Bowers
@Daniel_Bowers

HPC and AI are totally different. HPC is large, expensive clusters of GPUs running arcane algorithms from PhDs who belittle AI. But AI is large, expensive clusters of GPUs running arcane algorithms from PhDs who belittle HPC.

2:37 AM · Apr 6, 2023 · **8,609** Views

https://x.com/Daniel_Bowers/status/1643774967115292672

GB200 NVL72

Closing Remarks

What I would like for you to remember from this talk

- There will be an increased demand for library APIs and algorithms that can deliver energy efficient, high-throughput linear algebra capabilities for reduced-precision types
  - Mixed-Precision Algorithms
  - Floating-Point Emulation
  - Batched APIs
- Application and library developers have an opportunity to do more with the resources present on the evolving processor and system architectures
  - Tools that aid precision tuning and mixed-precision algorithm development are needed
- Energy efficient kernel design will become increasingly important
  - Max-Q vs. Max-P
- Scientific computing leverages both AI & HPC
  - The role of AI will continue to grow
  - Greater focus will be on algorithms that accelerate both

NVIDIA