Growing Up at Argonne National Laboratory

Jack Dongarra

University of Tennessee

University of Manchester

Oak Ridge National Laboratory

I wanted to be a science high school teacher

- Enrolled as an undergraduate at a college for teachers for the Chicago public school system
- My last semester in college my physics professor encouraged me to apply to a program to spend a semester at Argonne working with a scientist.

Brian Smith

Worked on a software project called EISPACK.

Many visitors from various universities.

Cleve Moler, U of New Mexico

1970s HPC Systems

CDC 7600 36.4 MHz (27.5 ns clock cycle)

- Primary memory 65 Kwords (60-bit words)
- Seymour Cray design
- Peak 36 Mflop/s
- Broke down at least once/day (often four or five times)

IBM 370/195 18.5 MHz (54 ns clock cycle)

- High degree of parallelism
- Up to 7 operations at a time
- Up to 4 MB of memory

Both systems had a high degree of instruction-level pipelining and parallelism.

Evolution of HPC Technology in the Last 50 Years

1970s-1980s: Vector Supercomputers (Cray)
1990s-2000s: Parallel computing and distributed systems
2010s: Rise of GPUs, cloud HPC
2020s and beyond: AI acceleration, quantum computing explorations

Vector Supercomputers
Cray-1

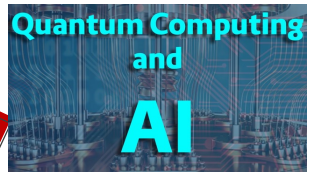
Shared Memory
Multiprocessors
SGI Power Challenge

Distributed
Memory

Massively Parallel
Processors
TMC CM-5

x86 Linux Clusters

Accelerated Clusters



EISPACK (1970's) NATS Project
(Translation of Algol to F66)

LINPACK (1980's)
(Vector operations)

Level 1 Basic Linear Algebra
Subprograms (BLAS)

Level 2 & 3 BLAS - ATLAS

LAPACK (1990's)
(Blocking, cache friendly)

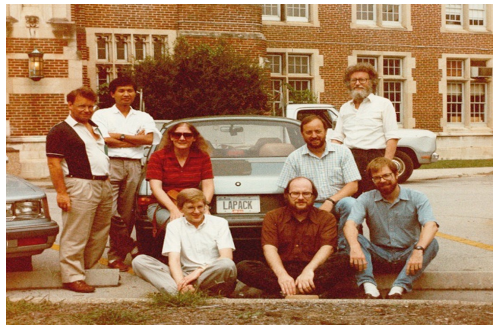
PVM and MPI

ScaLAPACK (2000's)
(Distributed Memory)

PLASMA / MAGMA (2010's)
(Many-core friendly & GPUs)

SLATE (2020's)
(DM and Heterogeneous arch)

Evolving Software and Algorithms
Following Hardware Developments



LINPACK Benchmark ⟶ Top500

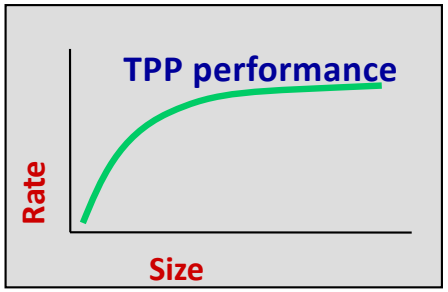
- Since 1977 I maintained a LINPACK Benchmark list.
- Hans Meuer and Erich Strohmaier had a list of fastest computers ranked by peak performance.
- Since 1993 listing of the 500 most powerful computers using 64-bit floating point arithmetic.
- Yardstick: Performance for

  Ax=b, dense problem

Maintained and updated twice a year:

SC‘xy in the States in November

Meeting in Germany in June



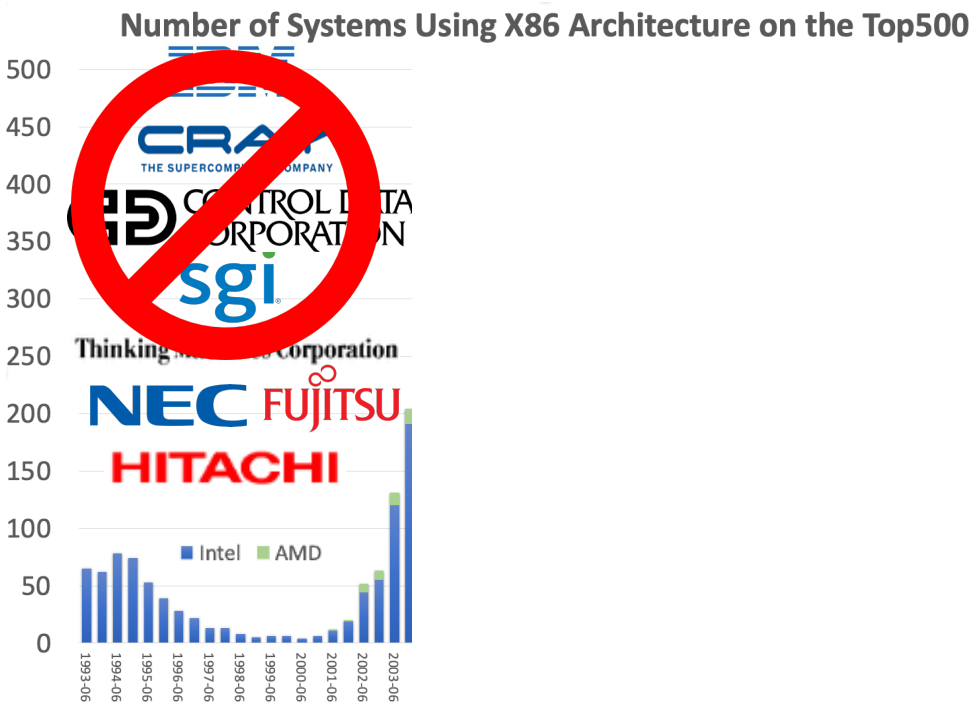
TOP500 The List.

Major paradigm shift

Attach of the Killer Micros

- TOP500 list began in 1993
  - 65 systems used Intel's i860 architecture
  - Remainder had specialized architectures, mainly vector based

Most of the HPC systems were specially built for computational science applications

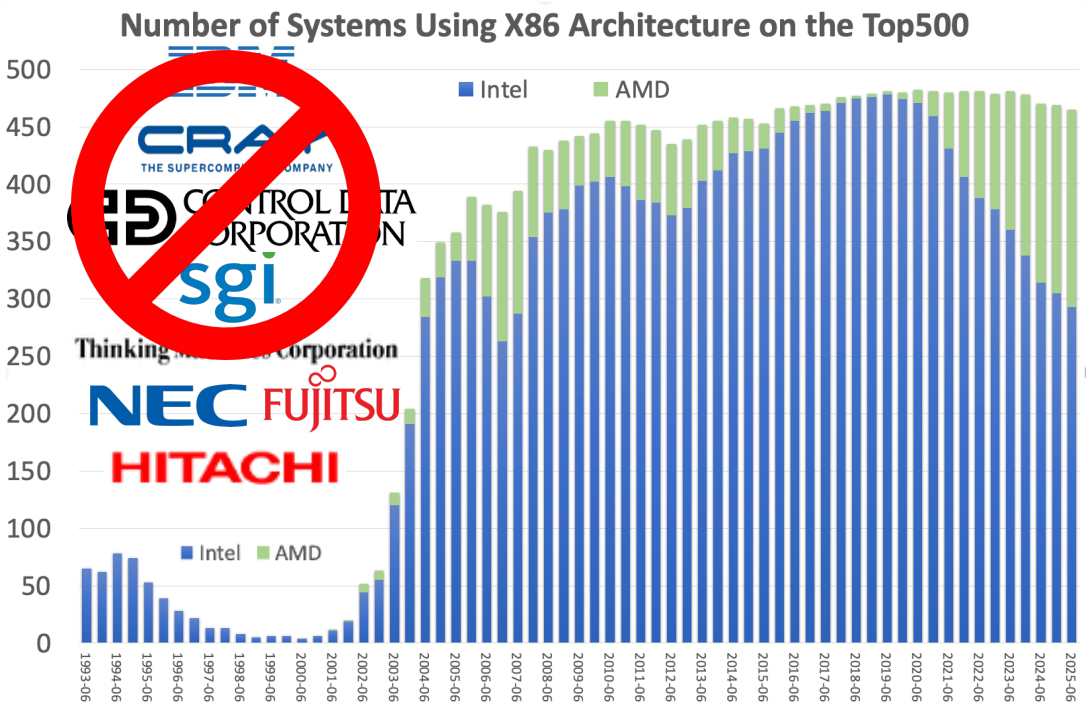


Scientific High Performance Computing based on Commodity Processors

Major paradigm shift

Attach of the Killer Micros

- TOP500 list began in 1993
  - 65 systems used Intel's i860 architecture
  - Remainder had specialized architectures, mainly vector based
- Today's TOP500 list
  - 59% of systems used Intel processors
  - Another 34% used AMD processors
- **93% of the systems use x86-64 architecture**
  - Many use GPU accelerators



Today, Our HPC Systems are Based on Commodity Parts

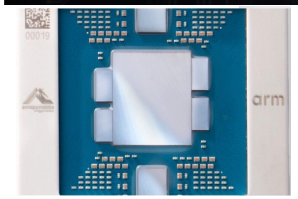
- **Commodity Processors**
  - 93% of the Top500 system use X86 (Intel & AMD) instruction set
- **Commodity Accelerators**
  - 92% of accelerated systems use NVIDIA
- **Commodity Interconnect**
  - 85% of the Top500 systems use Ethernet or Infiniband
- **Commodity OS**
  - 100% of the Top500 systems run on Linux

- **Unlike the HPC Community, the Hyperscalers (Cloud Providers)**
  - They are building their processors, accelerators, and interconnects

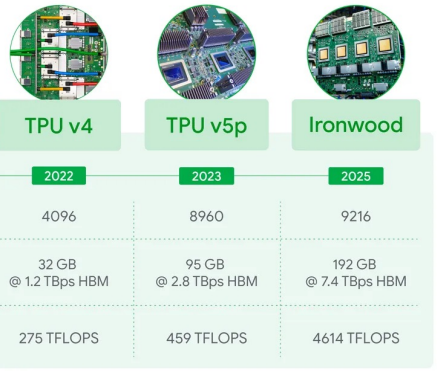
Cloud Providers are Designing and Using Their Own Processors

- Alibaba
  - CIPU, 128 core ARM based
  - Alibaba's Elastic Compute Service

- AWS Graviton4
  - 96 ARM cores, 7 chiplet design
  - ~100 billion transistors, DDR5 memory



- Google TPU7
  - 2X TPU3 performance
  - 4096 units per "pod"
  - Reconfigurable optical interconnect

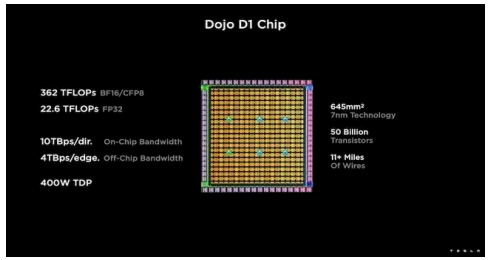


	TPU v4	TPU v5p	Ironwood
	2022	2023	2025
Pod Size (chips)	4096	8960	9216
HBM Bandwidth/ Capacity	32 GB @ 1.2 TBps HBM	95 GB @ 2.8 TBps HBM	192 GB @ 7.4 TBps HBM
Peak Flops per chip	275 TFLOPS	459 TFLOPS	4614 TFLOPS

- Microsoft Azure
  - Project Catapult/Brainwave FPGA accelerator
  - Cobalt 100 (128 Neoverse N2 ARMv9 cores)
  - Maia100 (Athena) AI accelerator
  - $10B+ OpenAI investment/$80B in data centers

Even car makers

- Tesla



OpenAI

SambaNova SYSTEMS

cerebras

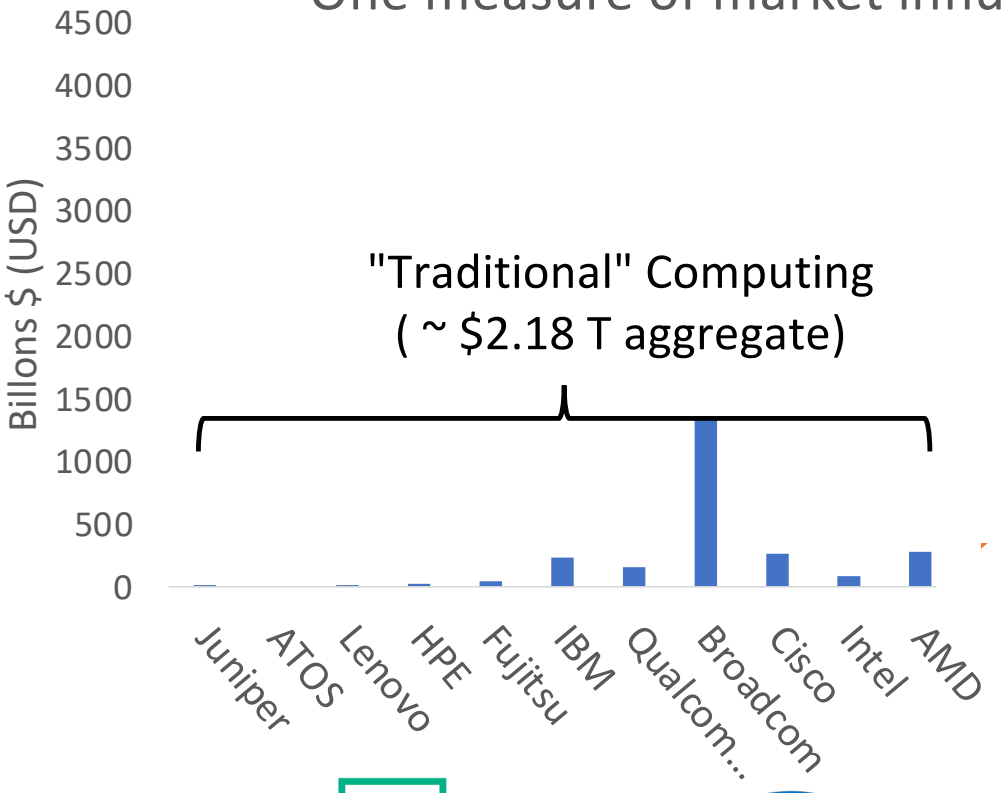
GRAPHCORE

groq

Market Capitalizations

August 2, 2025

One measure of market influence



Hewlett Packard Enterprise

intel

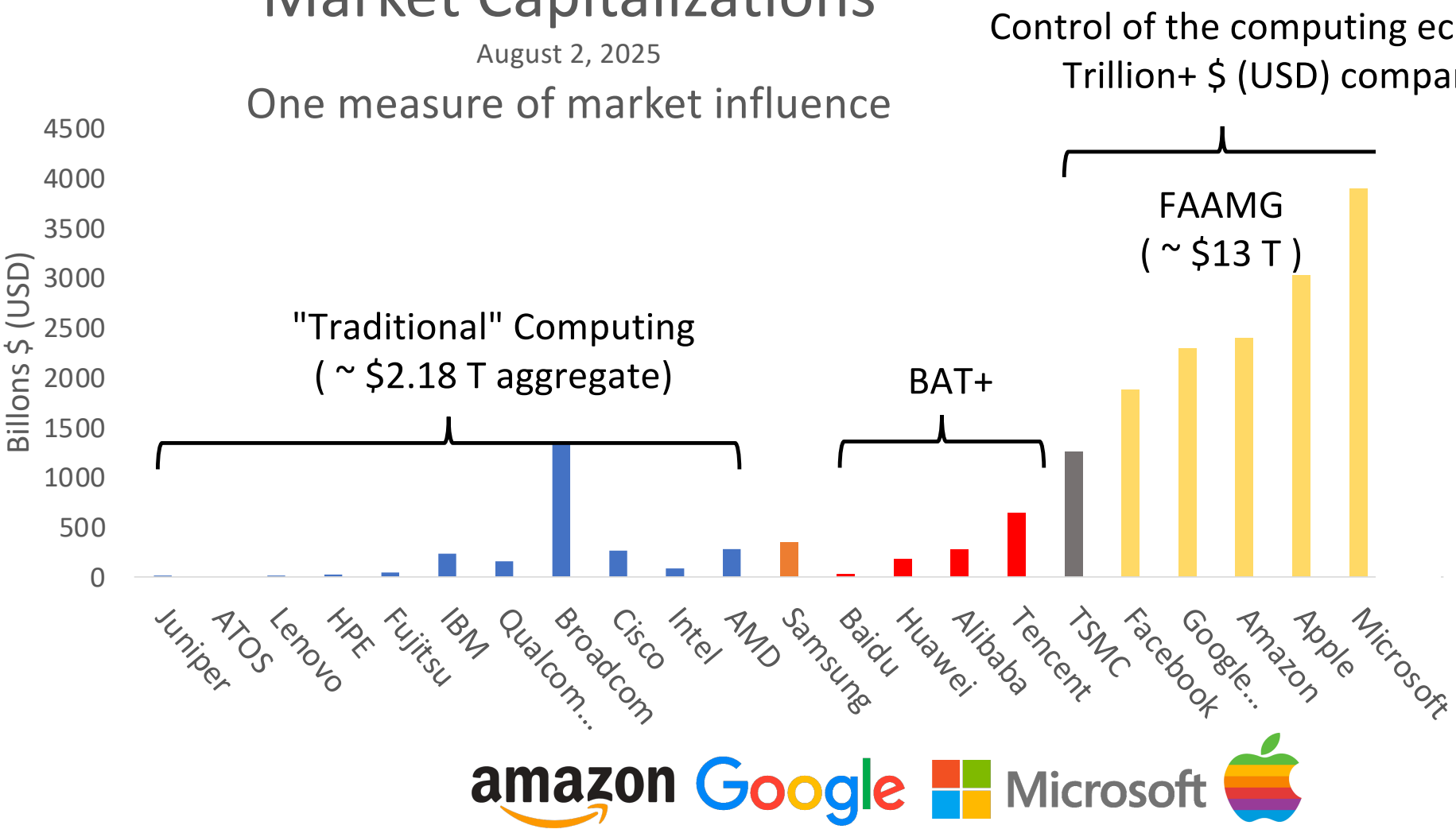
IBM

AMD

Market Capitalizations

August 2, 2025

One measure of market influence

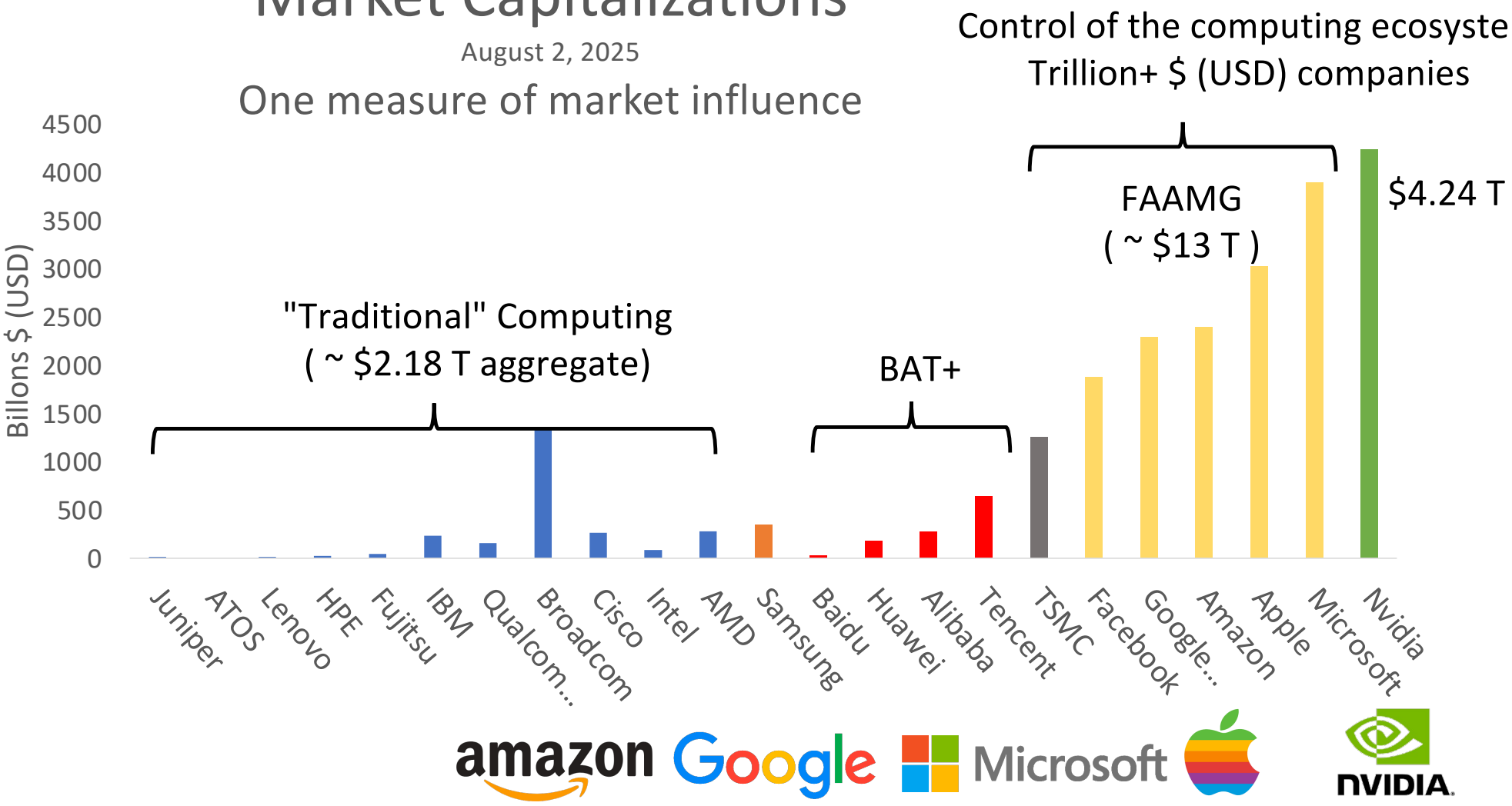


amazon Google Microsoft

Market Capitalizations

August 2, 2025

One measure of market influence



The Fastest Supercomputers are at an Exaflop.

What's an Exaflop?

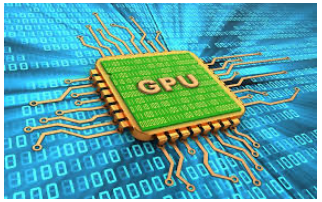
- 1 flop = Addition or Multiplication of 64-bit floating point numbers
- Exaflop is a billion-billion (10^{18}) floating point operations per second
- If each person on Earth completed 1 calculation per second, it would take more than 4 years to do what an Exascale computer can do in 1 second

The Environment for High Performance Computing in Scientific Computation

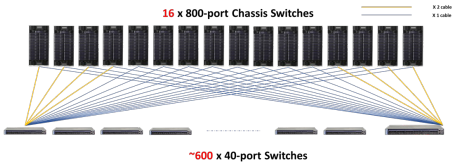
- Highly parallel
  - Distributed memory
  - MPI + Open-MP programming model

LLNL El Capitan, 2.7 Eflop/s,
11 x 10^6 Cores, 11,136 nodes, 35 MW
(node = 3-AMD CPU + 3-AMD GPUs)
> 99% of performance from GPUs

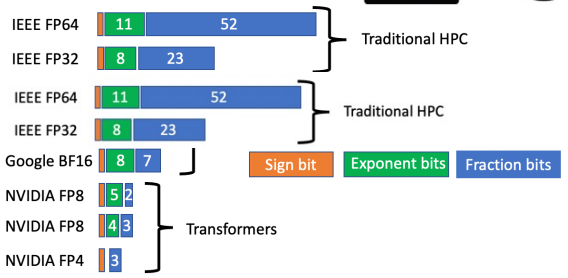
- Heterogeneous
  - Commodity processors + GPU accelerators



- Communication between parts very expensive compared to floating point ops



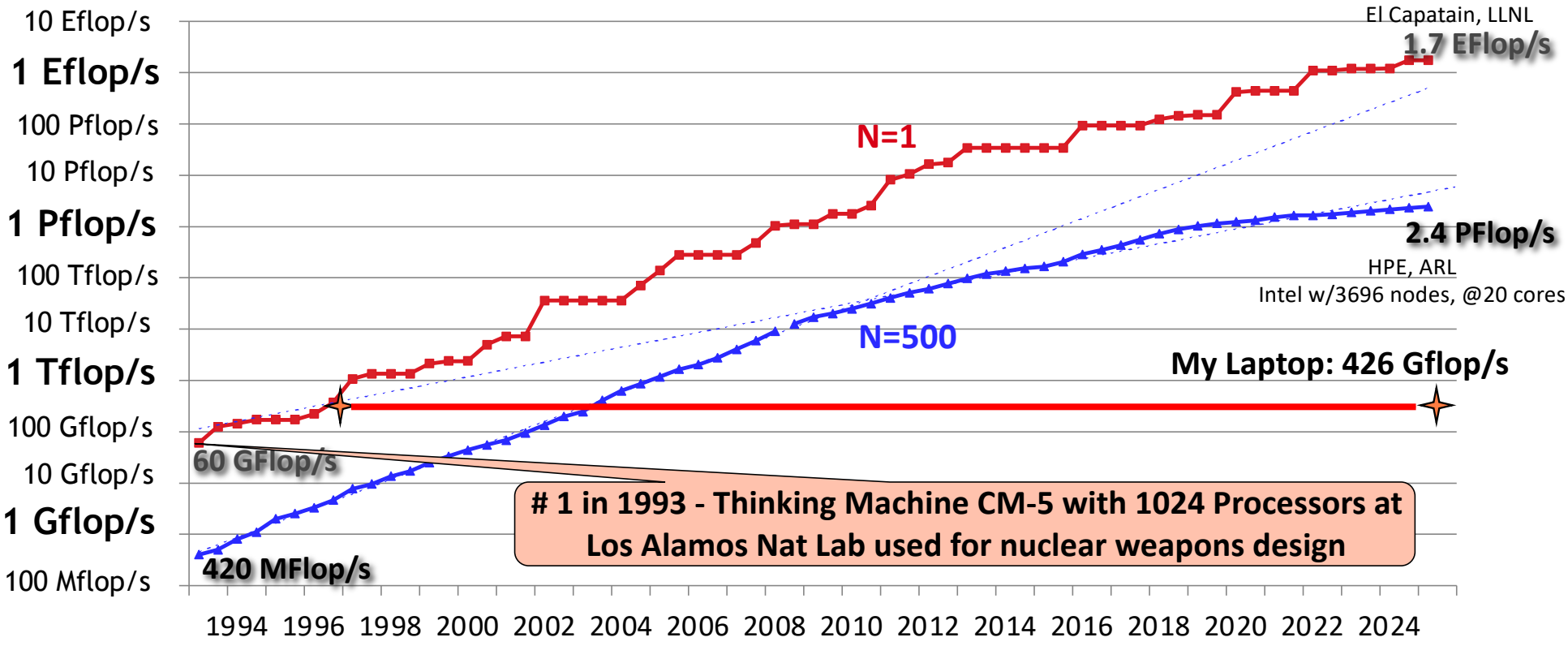
- Comparison of operation counts may not reflect time to solution



- Floating point hardware at 64 & 32 bits

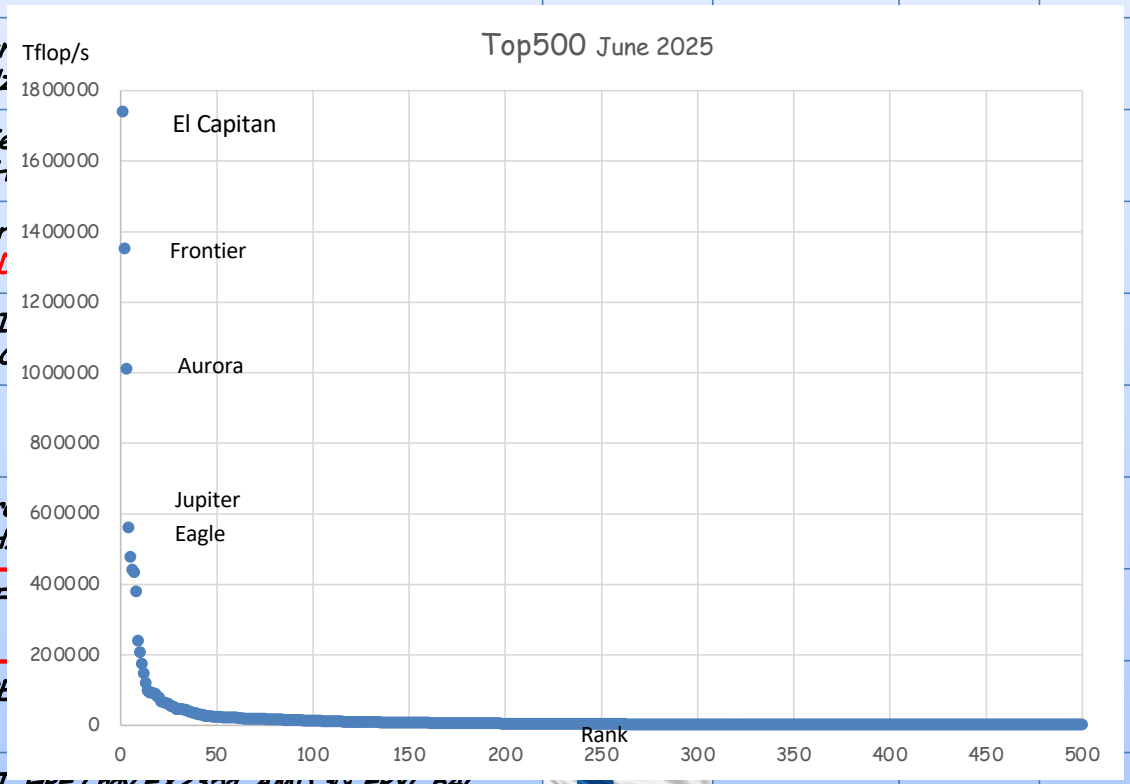
- Floating point hardware at 64, 32, 16, 8, & 4 bits

Performance Development of HPC over the Last 33 Years from the Top500



June 2025: The TOP 10 Systems (54% of the Total Performance of Top500)

Rank	Site	Computer	Country	Cores	Rmax [Pflops]	% of Peak	Power [MW]	GFlops/ Watt
1	DOE / NNSA LLNL	El Capitar GH					29.5	58.9
2	DOE / OS Oak Ridge Nat Lab	Frontie 2 GH					24.6	55.0
3	DOE / OS Argonne Nat Lab	Auror Intel					38.7	26.1
4	EuroHPC/FZL	JUPI 72					13.1	60.5
5	Microsoft, Azure Cloud						-	
6	Eni S.p.A.	HPE Cr 2GH					8.46	56.5
7	RIKEN Center for Computational Science	F					29.9	14.8
8	Swiss National Supercomputing Center CSCS	Alps, HPE					7.12	61.0
9	EuroHPC /CSC	LUMI, HPE Cray EX235a, AMD 3rd EPYC 64C, 2 GHz, AMD Instinct MI250X, Slingshot 11	Finland	2,752,704	380.	71	7.10	52.3
10	EuroHPC/CINECA	Leonardo, BullSequana XH2000, Xeon Platinum 8358 32C, 2.6GHz, NVIDIA A100 (108C), Quad-rail NVIDIA HDR100	Italy	1,824,768	241.	78	7.49	32.2

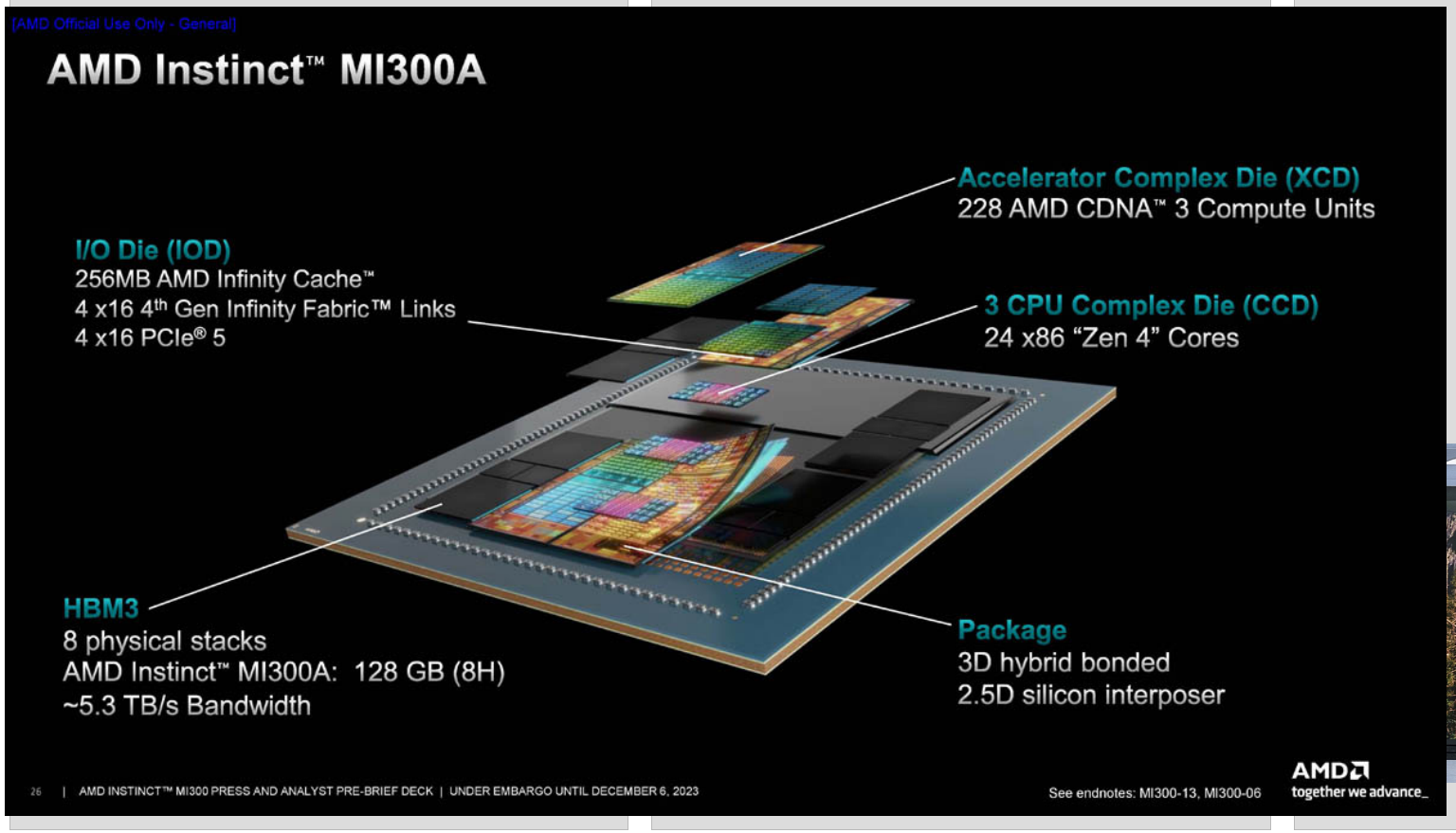


El Capitan Current #1 System Overview

System Performance

Each node has

The system includes

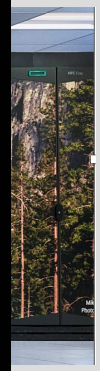


11,136 nodes

- 3 - 8-core "Zen 4" CPU dies
- 6 - AMD 38-core CDNA 3 GPU die

Cray Slingshot interconnect

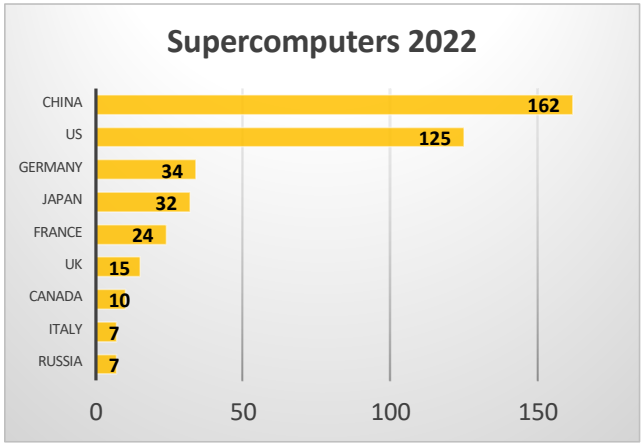
- 4 end points per node

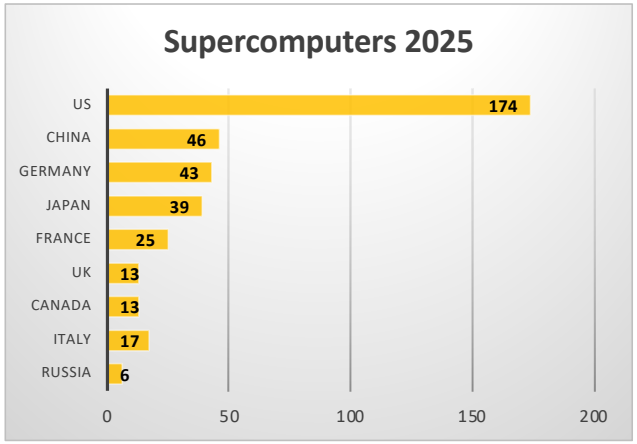


Rumored to be 3-4 Exascale Systems in China

- In the US, El Capitan, Frontier, and Aurora systems remain the only exascale systems on the Top500

- China stopped its submissions to the Top500



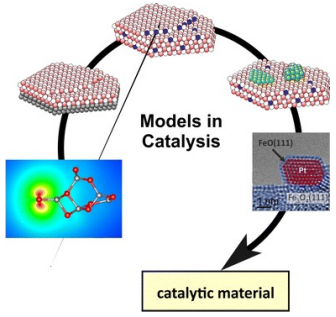


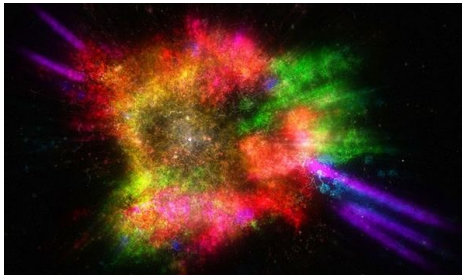
Performance and Benchmarking Evaluation Tools

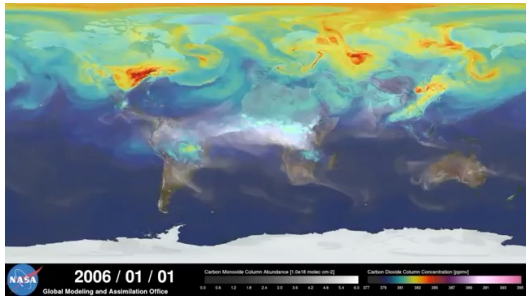
- **Linpack Benchmark - Longstanding benchmark started in 1979**
  - **Lots of positive features; easy to understand and run; shows trends**
- **However, much has changed since 1979**
  - **Arithmetic was expensive then and today it is over-provisioned and inexpensive**
- **Linpack performance of computer systems is no longer strongly correlated to real application performance**
  - **Linpack benchmark based on dense matrix multiplication**
- **Designing a system for good Linpack performance can lead to design choices that are wrong for today's applications**

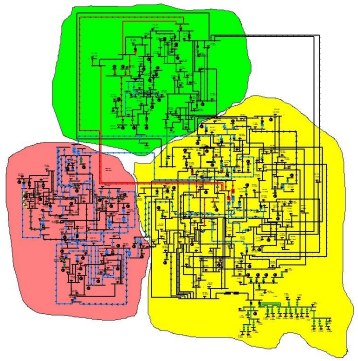
Today's Top HPC Systems Used to do Simulations

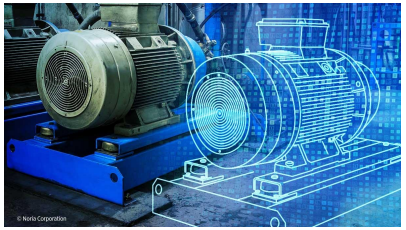
- *Climate*
- *Combustion*
- *Nuclear Reactors*
- *Catalysis*
- *Electric Grid*
- *Fusion*
- *Stockpile*
- *Supernovae*
- *Materials*
- *Digital Twins*
- *Accelerators*
- …

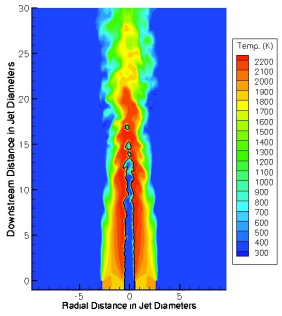


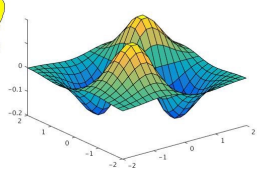


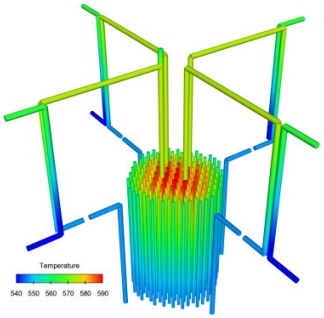


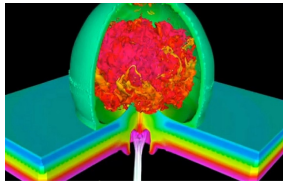


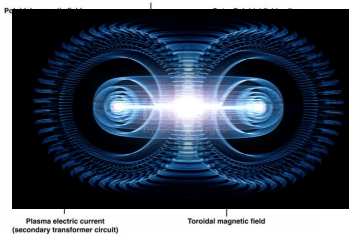










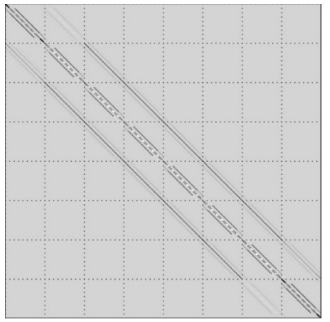


- Usually 3-D PDE's
  - Sparse matrix computations, not dense

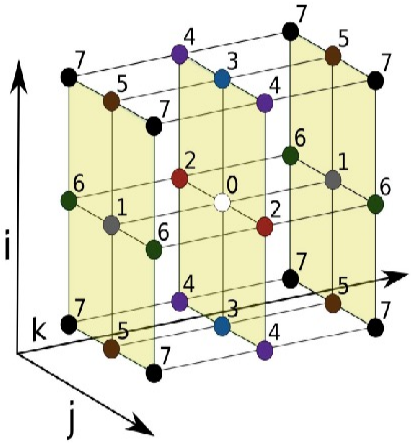
hpcg-benchmark.org With Piotr Luszczek and Mike Heroux

HPCG Results; The Other Benchmark

- High Performance Conjugate Gradients (HPCG).
- Solves *Ax=b, A* large, sparse, *b* known, *x* computed.
- An optimized implementation of PCG contains essential computational and communication patterns that are prevalent in a variety of methods for discretization and numerical solution of PDEs



- Patterns:
  - Dense and sparse computations.
  - Dense and sparse collectives.
  - Multi-scale execution of kernels via MG (truncated) V cycle.
  - Data-driven parallelism (unstructured sparse triangular solves).
- Strong verification (via spectral properties of PCG).



27-point stencil operator

HPCG Top 10, June 2025

Ax=b Dense A

Ax=b Sparse A

Rank	Site	Computer	Cores	HPL Rmax (Pflop/s)	TOP500 Rank	HPCG (Pflop/s)	Fraction of Peak HPCG
1	DOE/SC/LLNL **USA**	**El Capitan**, HPE Cray 255a, AMD 4th Gen EPYC 24C 1.8 GHz, AMD Instinct MI300A, Slingshot-11	11,039,616	1742	1	17.4	0.6%
2	RIKEN Center for Computational Science **Japan**	**Fugaku**, Fujitsu A64FX 48C 2.2GHz, Tofu D, Fujitsu	7,630,848	442	7	16.0	**3.0%**
3	DOE/SC/ORNL **USA**	**Frontier,** HPE Cray Ex235a, AMD 3rd EPYC 64C, 2 GHz, AMD Instinct MI250X, Slingshot-11	9,066,176	1353	2	14.1	0.7%
4	DOE/SC/ANL **USA**	**Aurora,** HPE Cray EX, Intel Max 9470 52C, 2.4 GHz, Intel GPU MAX, Slingshot-11	9,264,128	1012	3	5.6	0.3%
5	EuroHPC/CSC **Finland**	**LUMI**, HPE Cray EX235a, AMD Zen-3 (Milan) 64C 2GHz, AMD MI250X, Slingshot-11	2,752,704	380	9	4.6	0.9%
6	CSCS **Switzerland**	**Alps**, HPE Cray EX254n, NVIDIA Grace 72C 3.1GHz, NVIDIA GH200 Superchip, Slingshot-11	2,121,600	435	8	3.7	0.6%
7	EuroHPC/CINECA **Italy**	**Leonardo**, BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 40 GB, Quad-rail NVIDIA HDR100 Infiniband	1,824,768	241	10	3.1	1.0%
[illegible]	AIST	**ABCI 3.0**, HPE Cray XD670, Xeon Platinum 8558 48C [illegible]	[illegible]	[illegible]	[illegible]	[illegible]	**1.3%**
9	[illegible] **USA**	[illegible], 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-10	888,832	79	25	1.9	[illegible]%
10	DOE/NNSA/LLNL **USA**	**Sierra**, S922LC, IBM POWER9 20C 3.1 GHz, Mellanox EDR, NVIDIA Volta V100, IBM	1,572,480	95	20	1.8	**1.4%**

Think of a race car that has the potential of 200 KPH but only goes 2 KPH!

WHY MIXED PRECISION? (Less is Faster)

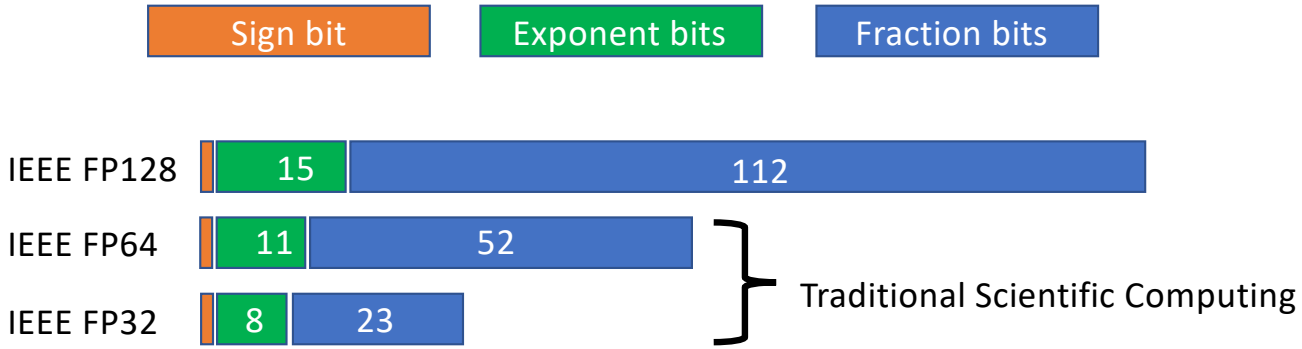
- **There are many reasons to consider using mixing precisions within an application:**
  - Less Communication
    - Reduce memory traffic (from memory to processor)
    - Reduce network traffic (from node to node)
  - **Reduce memory footprint (less data to store)**
  - **Arithmetic faster (usually factor of 2 or more)**
    - **Lower precision is usually faster than high precision operations**
    - **Architecture may have an accelerator**
  - **Suitable numerical properties for the algorithm & problems.**

The hope is to improve the algorithm performance without compromising the quality of science

“Responsibly Reckless” Algorithms

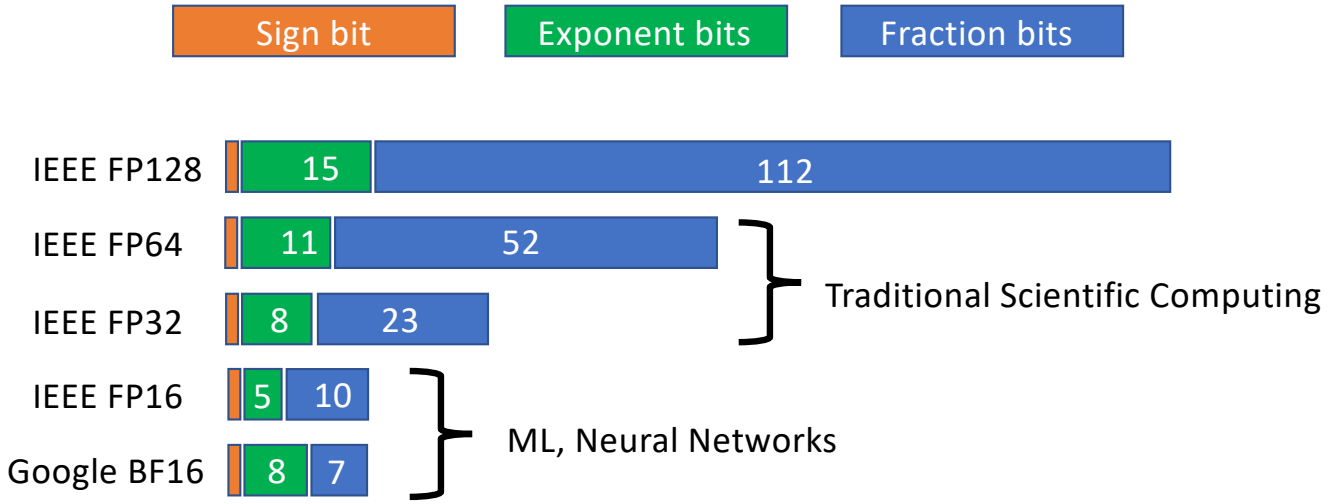
- Try a fast algorithm (unstable algorithm) that might fail (but rarely)
  - Avoiding Data Movement
  - Avoiding Synchronization
  - Use Mixed Precision
- Check for instability
- If needed, recompute with a stable algorithm

Floating Point Representation



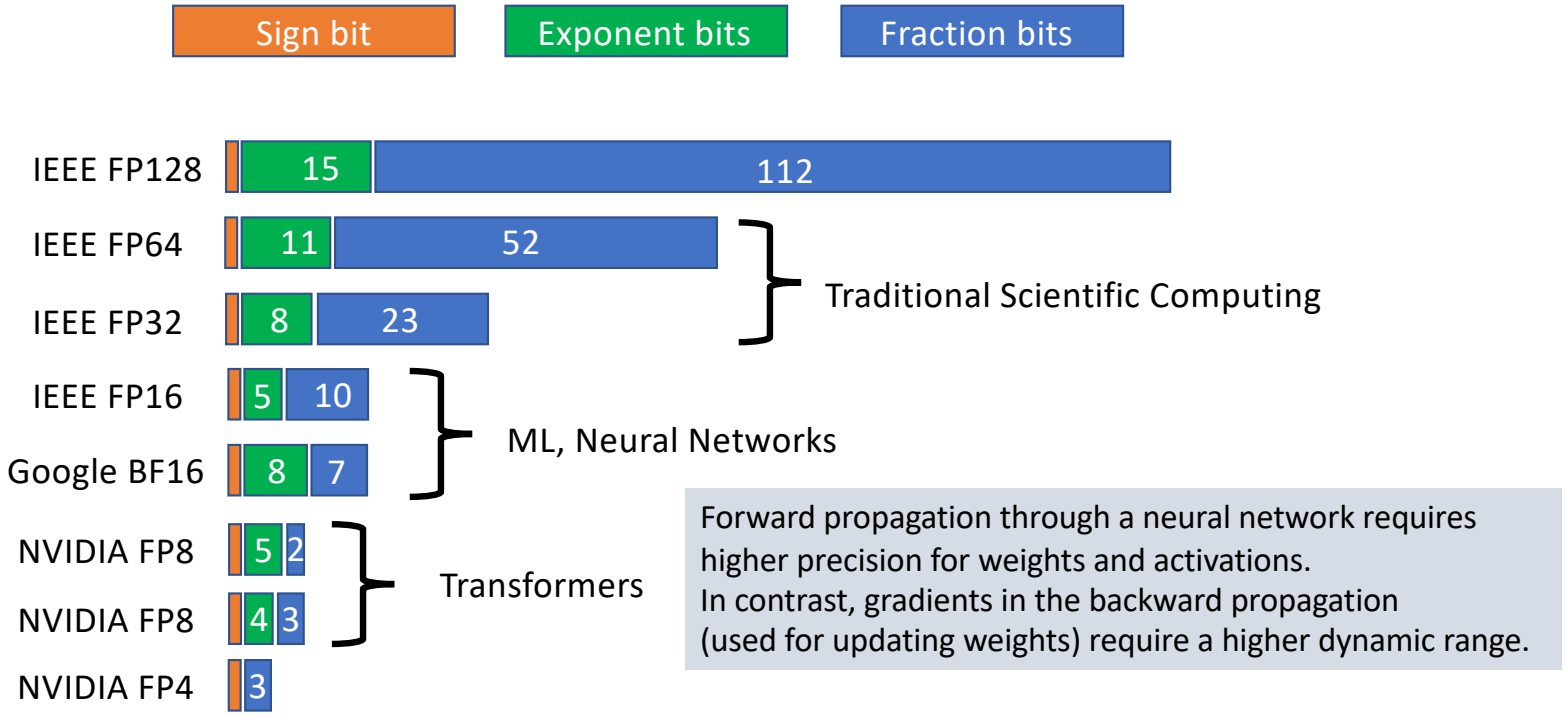
Can we leverage the short precision in our "traditional" scientific numerical computations?

Floating Point Representation



Can we leverage the short precision in our "traditional" scientific numerical computations?

Floating Point Representation



Forward propagation through a neural network requires higher precision for weights and activations.
In contrast, gradients in the backward propagation (used for updating weights) require a higher dynamic range.

Can we leverage the short precision in our "traditional" scientific numerical computations?

Mixed Precision++

Use a mathematical technique

- **Get an approximation in lower precision (fast) then use something like Newton's method to enhance accuracy.**
- **Newton's Method**
  - $x_+ = x - f(x)/f'(x)$
  - For $Ax = b$; $f(x) = b - Ax$ and $f'(x) = -A$
  - $x_+ = x + A \setminus (b - Ax)$; $r = b - Ax$
  - $(x_+ - x) = A^{-1} * r$
  - $\Delta = (L*U)^{-1} * r$

Leveraging Half Precision

Idea: use low precision to compute the expensive flops (LU $O(n^3)$) and then iteratively refine ($O(n^2)$) the solution in order to achieve the FP64 arithmetic

Iterative refinement for dense systems, $Ax = b$, can work this way.

L U = lu(A)	lower precision	$O(n^3)$
x = U\(L\b)	lower precision	$O(n^2)$
r = b – Ax (with original A)	FP64 precision	$O(n^2)$

WHILE || r || not small enough

1. find a correction "z" to adjust x that satisfy Az=r
   solving Az=r could be done by either:
   - ~~z = U\(L\r) Classical Iterative Refinement~~ ~~lower precision~~ ~~$O(n^2)$~~
   - GMRes preconditioned by the LU to solve Az=r Iterative Refinement GMRes — lower precision $O(n^2)$
2. x = x + z — FP64 precision $O(n^1)$
3. r = b – Ax (with original A) — FP64 precision $O(n^2)$

END

Higham and Carson showed can solve the inner problem with iterative method and not infect the solution with the conditioning of the original matrix.

J. Langou, et al., Exploiting the Performance of 32 bit fl-pt Arithmetic in Obtaining 64 bit Accuracy, in: Proc. of SC06

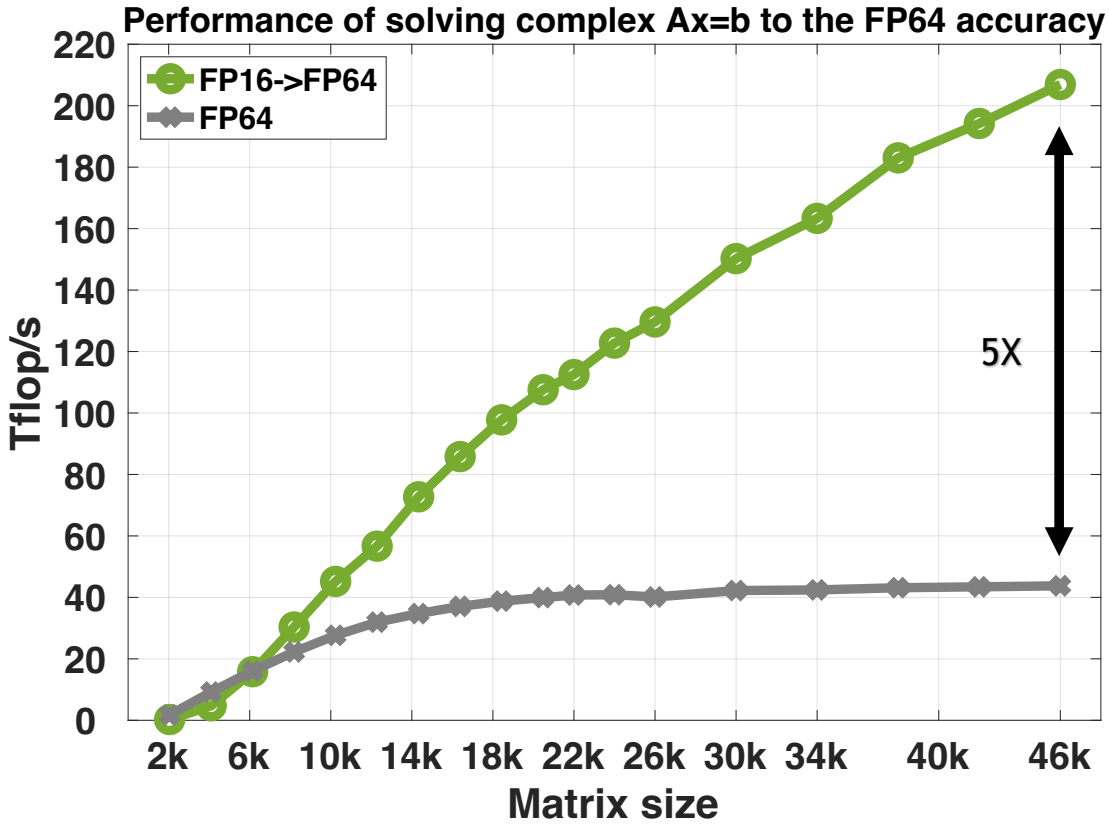
E. Carson & N. Higham, "Accelerating the Solution of Linear Systems by Iterative Refinement in Three Precisions *SIAM J. Sci. Comput.*, 40(2), A817–A847.

Originally motivated by the Sony PlayStation
SP peak 205 Gflop/s, DP peak 15 Gflop/s

- Wilkinson, Moler, Stewart, & Higham provide error bound for SP fl pt results when using DP fl pt

Tensor Core Accelerated Iterative Refinement

Nvidia H200



Flop/s = $2n^3/(3\ \text{time})$

(Meaning 5X is 5 times faster)

HPL-MxP Benchmark Utilizing 16-bit Arithmetic

1. Generate random linear system Ax=b
2. Represent the matrix A in low precision (16-bit floating point)
3. Factor A in lower precision into LU by Gaussian elimination

4. Compute approximate solution with LU factors in low precision
5. Perform a few iterations of refinement, e.g., GMRES to get accuracy up to 64-bit floating point
   a. Use LU factors for preconditioning

Iterative refinement for dense systems, *Ax = b*, can work this way.		
L U = lu(A)	Lower precision	$O(n^3)$
x = U\(L\b)	Lower precision	$O(n^2)$
GMRes preconditioned by the LU to solve Ax=b	FP64 precision	$O(n^2)$

6. Validate the answer is correct: scaled residual small $\frac{||Ax-b||}{||A||||x||+||b||} \times \frac{1}{n\epsilon} \leq O(10)$
7. Compute performance rate as $\frac{2}{3} \times \frac{n^3}{\text{time}}$

HPL-MxP Top 10 for June 2025

Rank	Site	Computer	Cores	HPL Rmax (Eflop/s)	TOP500 Rank	HPL-MxP (Eflop/s)	Speedup
1	DOE/SC/LLNL **USA**	**El Capitan**, HPE Cray 255a, AMD 4th Gen EPYC 24C 1.8 GHz, AMD Instinct MI300A, Slingshot-11	11,039,616	1.742	1	16.7	9.6
2	DOE/SC/ANL **USA**	**Aurora**, HPE Cray EX, Intel Max 9470 52C, 2.4 GHz, Intel GPU MAX, Slingshot-11	8,159,232	1.012	3	11.6	11.5
3	DOE/SC/ORNL **USA**	**Frontier**, HPE Cray EX235a, AMD Zen-3 (Milan) 64C 2GHz, AMD MI250X, Slingshot-11	8,560,640	1.353	2	11.4	8.4
4	AIST **Japan**	**ABCI 3.0**, HPE Cray XD670, Xeon Platinum 8558 48C 2.1GHz, NVIDIA H200 SXM5 141 GB, Infiniband NDR200, HPE	479,232	0.145	15	2.36	16.3
5	EuroHPC/CSC **Finland**	**LUMI**, HPE Cray EX235a, AMD Zen-3 (Milan) 64C 2GHz, AMD MI250X, Slingshot-11	2,752,704	0.380	9	2.35	6.2
6	RIKEN Center for Computational Science, **Japan**	**Fugaku**, Fujitsu A64FX 48C 2.2GHz, Tofu D	7,630,848	0.442	7	2.0	4.5
7	EuroHPC/CINECA **Italy**	**Leonardo**, BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 40 GB, Quad-rail NVIDIA HDR100 Infiniband	1,824,768	0.241	10	1.8	7.6
8	CII, Institute of Science **Japan**	**TSUBAME 4**, HPE Cray XD665, AMD EPYC 9654 96C 2.4GHz, NVIDIA H100 SXM5 94 GB, Mellanox NDR200	172,800	0.035	46	0.64	16.2
9	NVIDIA **USA**	**Selene**, DGX SuperPOD, AMD EPYC 7742 64C 2.25 GHz, Mellanox HDR, NVIDIA A100	555,520	0.063	30	0.63	9.9
10	DOE/SC/LBNL/NERSC **USA**	**Perlmutter**, HPE Cray EX235n, AMD EPYC 7763 64C 2.45 GHz, Slingshot-10, NVIDIA A100	761,856	0.079	25	0.59	7.5

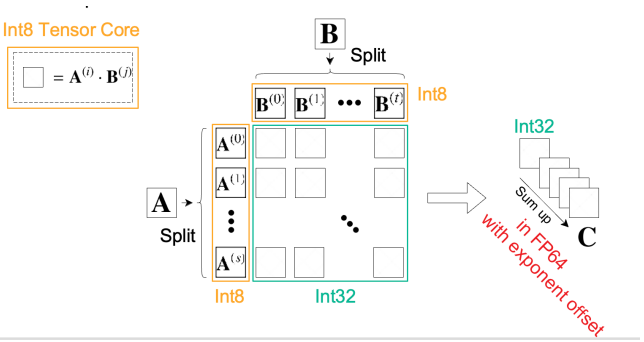
Recent Nvidia GPUs

	Figure of Merit Peak Performance	
Operations	2022 Hopper (H200)	2024 Blackwell (B200)
FP64 FMA	33.5 Tflop/s	40 Tflop/s
FP64 Tensor Core	67 Tflop/s	40 Tflop/s
FP32 FMA	67 Tflop/s	80 Tflop/s
FP16 Tensor Core	989 Tflop/s	2250 Tflop/s
BF16 Tensor Core	989 Tflop/s	2250 Tflop/s
INT8 Tensor Core	1979 TOP/s	4500 TOP/s
Memory BW	4.8 TB/s	8 TB/s

Opportunity Breeds Innovation, Emulating Fl.Pt. with Integer arithmetic, “Ozaki Scheme”

$$
\begin{aligned}
d &= a \cdot b + c \\
&= (a_0 + 2^{-8}a_1 + 2^{-16}a_2) \cdot (b_0 + 2^{-8}b_1 + 2^{-16}b_2) + c \\
&= a_0b_0 + 2^{-8}a_0b_1 + 2^{-16}a_0b_2 \\
&\quad 2^{-8}a_1b_0 + 2^{-16}a_1b_1 + 2^{-24}a_1b_2 \\
&\quad 2^{-16}a_2b_0 + 2^{-24}a_2b_1 + 2^{-32}a_2b_2 + c
\end{aligned}
$$

Divide the numbers into “slices” of 2^{-8}



Use Int8 Tensor Cores for each matrix multiplication
Int8-input Int32-accumulation

Analysis of Floating-Point Matrix Multiplication Computed via Integer Arithmetic

Ahmad Abdelfattah, Jack Dongarra, Massimiliano Fasi, Mantas Mikaitis, Françoise Tisseur

http://arxiv.org/abs/2506.11277

- The FP32 inputs are decomposed into 3 scaled BF16 components[1]

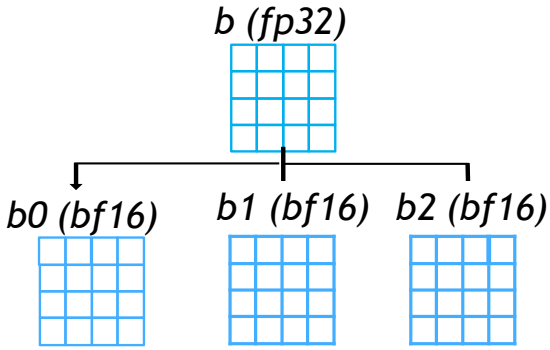
  $a = a0 + 2^{-8}.a1 + 2^{-16}.a2$

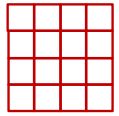
  $b = b0 + 2^{-8}.b1 + 2^{-16}.b2$

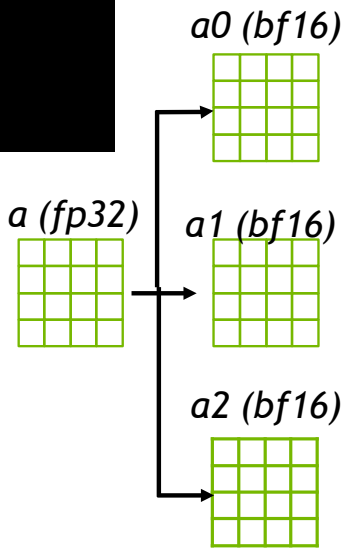
- The multiply-add operation is computed as a sum of 9 scaled partial products

  $a * b + c = a0.b0 + 2^{-8}.a0.b1 + 2^{-16}.a0.b2 + 2^{-8}.a1.b0 + 2^{-16}.a1.b1 + 2^{-24}.a1.b2 + 2^{-16}.a2.b0 + 2^{-24}.a2.b1 + 2^{-32}.a2.b2 + c$

- The partial products are computed in the BF16 Tensor cores
- The partial products are scaled appropriately in the CUDA cores
- The tensor cores and CUDA cores work in parallel
- The effective FP32 FLOPs is 1/9th that of the BF16 tensor core FLOPs
  - On B200 (per GPU) 250 vs 80 TFLOP/s ➔ **>3X maximum speed-up**







- The FP32 inputs are decomposed into 3 scaled BF16 components[1]

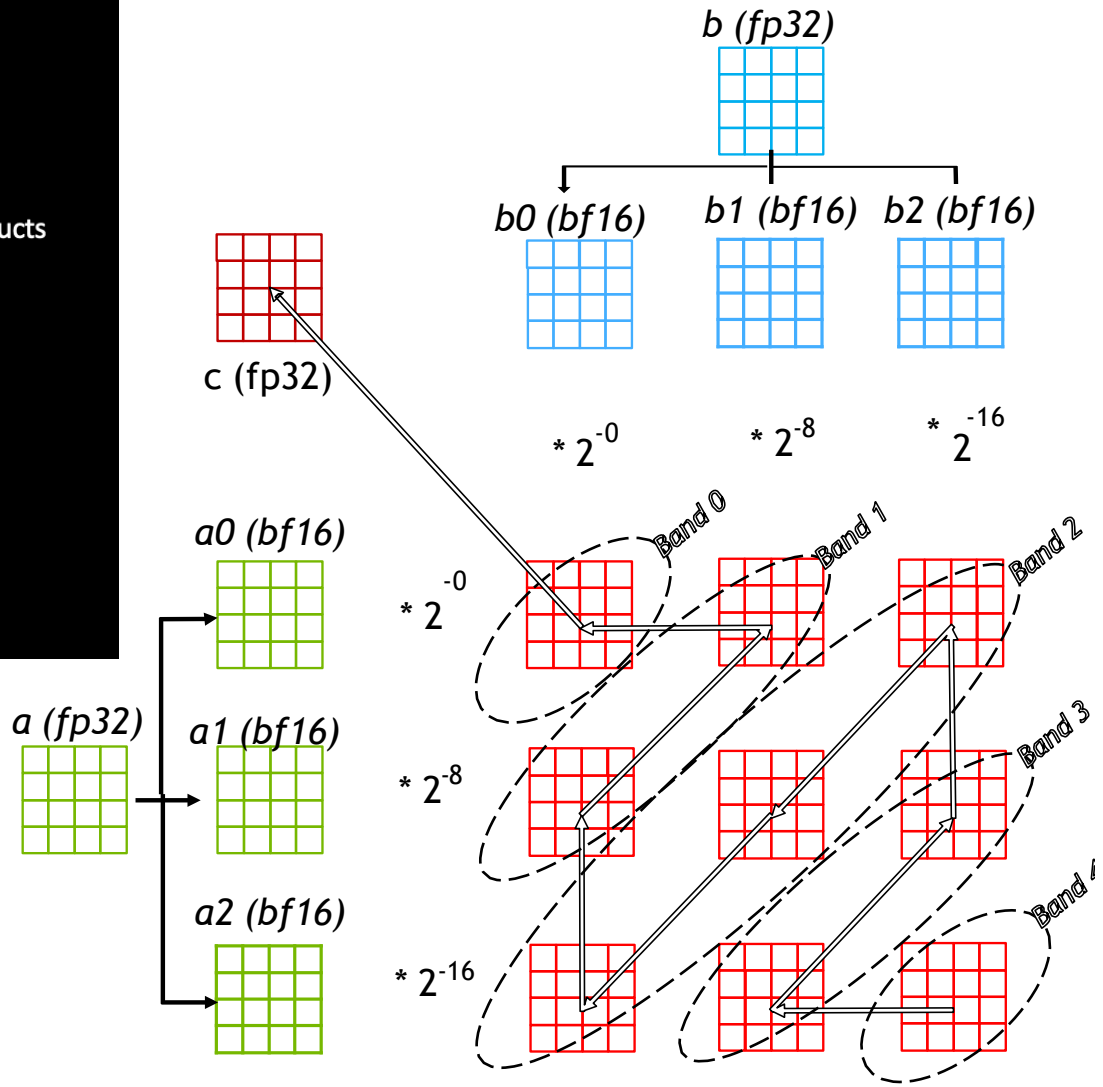
  $a = a0 + 2^{-8}.a1 + 2^{-16}.a2$

  $b = b0 + 2^{-8}.b1 + 2^{-16}.b2$

- The multiply-add operation is computed as a sum of 9 scaled partial products

  $a * b + c = a0.b0 + 2^{-8}.a0.b1 + 2^{-16}.a0.b2 + 2^{-8}.a1.b0 + 2^{-16}.a1.b1 + 2^{-24}.a1.b2 + 2^{-16}.a2.b0 + 2^{-24}.a2.b1 + 2^{-32}.a2.b2 + c$

- The partial products are computed in the BF16 Tensor cores
- The partial products are scaled appropriately in the CUDA cores
- The tensor cores and CUDA cores work in parallel
- The effective FP32 FLOPs is 1/9th that of the BF16 tensor core FLOPs
  - On B200 (per GPU) 250 vs 80 TFLOP/s ➔ **>3X maximum speed-up**



- The FP32 inputs are decomposed into 3 scaled BF16 components[1]

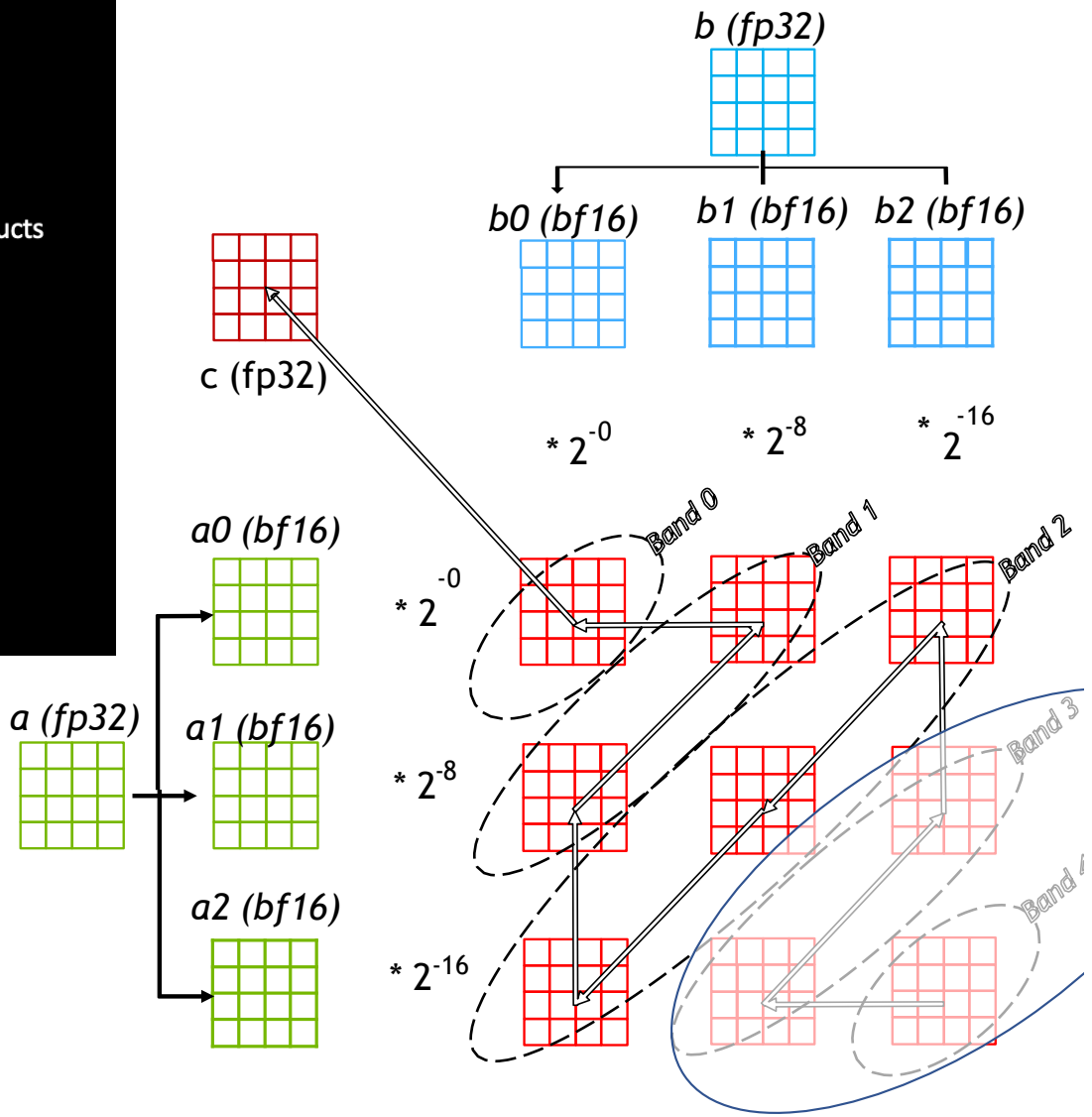
  $a = a0 + 2^{-8}.a1 + 2^{-16}.a2$

  $b = b0 + 2^{-8}.b1 + 2^{-16}.b2$

- The multiply-add operation is computed as a sum of 9 scaled partial products

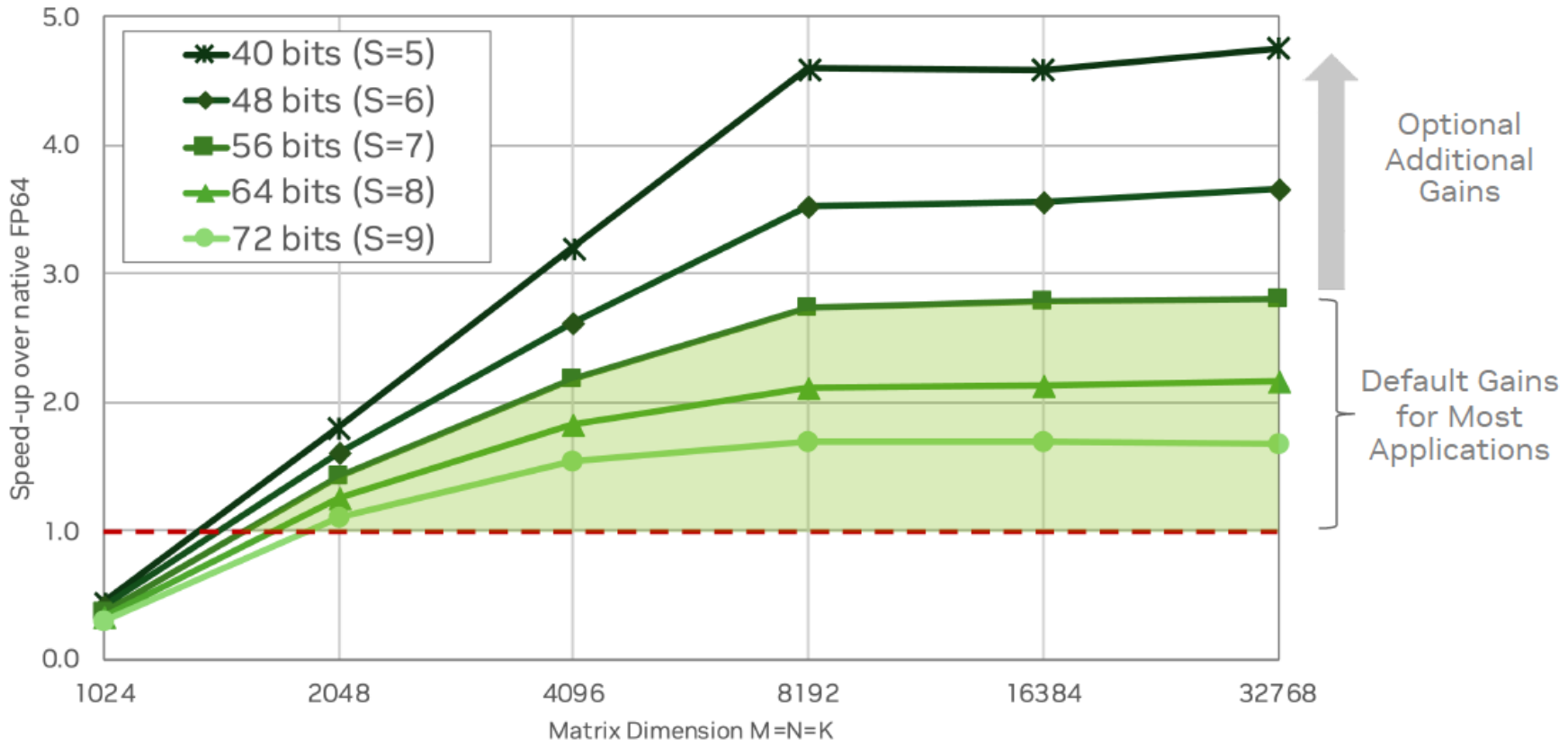
  $a * b + c = a0.b0 + 2^{-8}.a0.b1 + 2^{-16}.a0.b2 + 2^{-8}.a1.b0 + 2^{-16}.a1.b1 + 2^{-24}.a1.b2 + 2^{-16}.a2.b0 + 2^{-24}.a2.b1 + 2^{-32}.a2.b2 + c$

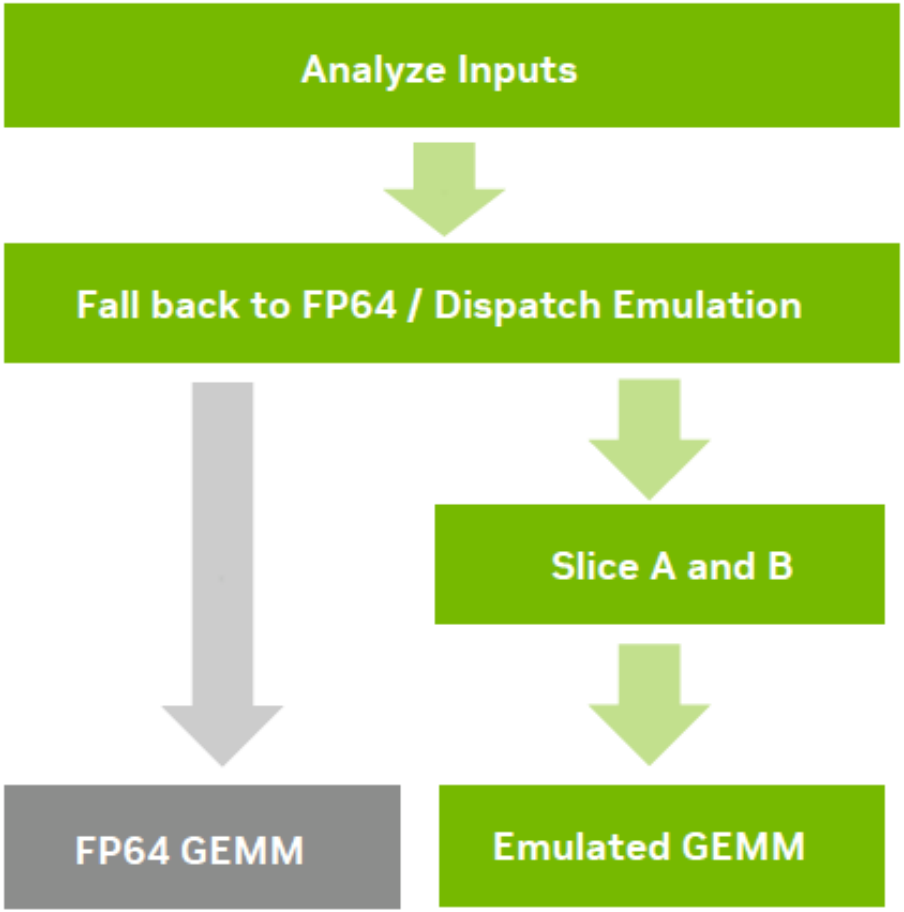
- The partial products are computed in the BF16 Tensor cores
- The partial products are scaled appropriately in the CUDA cores
- The tensor cores and CUDA cores work in parallel
- The effective FP32 FLOPs is 1/9th that of the BF16 tensor core FLOPs
  - On B200 (per GPU) 250 vs 80 TFLOP/s ➔ **>3X maximum speed-up**



Potential additional efficiency gains

Performance of Emulated GEMM on B200 GPUs for various number of bits used




Analyze Inputs
Fall back to FP64 / Dispatch Emulation
Slice A and B
FP64 GEMM
Emulated GEMM

GMRES

- General purpose, **sparse** linear solver
  - Iterative, Krylov solver
- Memory bound performance
- Selectively reduced precision
  - Improving performance
  - Without losing accuracy

Classic GMRES Algorithm with Restart

GMRES$_{res}(\boldsymbol{A}, \boldsymbol{x_0}, \boldsymbol{b}, \boldsymbol{M}^{-1})$ — Computing $\boldsymbol{Ax}=\boldsymbol{b}$. $\boldsymbol{A}^{-1}\approx\boldsymbol{M}^{-1}$

for k=0, 1, 2,… — Restarts

$\boldsymbol{r_k} \leftarrow \boldsymbol{b}-\boldsymbol{Ax_k}$

$\boldsymbol{z_k} \leftarrow \boldsymbol{M}^{-1}\,\boldsymbol{r_k}$

$\beta \leftarrow \|\boldsymbol{z_k}\|_2$

$\boldsymbol{V}_{:,0} \leftarrow \boldsymbol{z_k}/\beta$

$\mathbf{s} \leftarrow [\beta, 0, 0, \ldots, 0]^T$

for j=0, 1, 2, … — Iteration count

$\boldsymbol{w} \leftarrow \boldsymbol{M}^{-1}\,\boldsymbol{AV}_{:,j}$

$\boldsymbol{w}, \boldsymbol{H}_{:,j} \leftarrow orthogonalize(\boldsymbol{w}, \boldsymbol{V}_{:,j})$

$\boldsymbol{H}_{j+1,j} \leftarrow \|\boldsymbol{w}\|_2$

$\boldsymbol{V}_{:,j+1} \leftarrow \boldsymbol{w}/\|\boldsymbol{w}\|_2$

$\boldsymbol{H}_{:,j} \leftarrow \boldsymbol{G_0}\,\boldsymbol{G_1}\ldots\boldsymbol{G_{j-1}}\,\boldsymbol{H}_{:,j}$

$\boldsymbol{G_j} \leftarrow rotation\_matrix(\boldsymbol{H}_{:,j})$

$\boldsymbol{H}_{:,j} \leftarrow \boldsymbol{G_j}\,\boldsymbol{H}_{:,j}$

$\boldsymbol{s} \leftarrow \boldsymbol{G_j}\,\boldsymbol{s}$

$\boldsymbol{u_k} \leftarrow \boldsymbol{VH}^{-1}\,\boldsymbol{s}$

$\boldsymbol{x_{k+1}} \leftarrow \boldsymbol{x_k} + \boldsymbol{u_k}$

A Mixed Precision GMRES Algorithm

GMRES$_{res}(\boldsymbol{A}, \boldsymbol{x_0}, \boldsymbol{b}, \boldsymbol{M}^{-1})$ — Computing $\boldsymbol{Ax}=\boldsymbol{b}$. $\boldsymbol{A}^{-1}\approx\boldsymbol{M}^{-1}$

for k=0, 1, 2,… — Restarts

Double:

$\boldsymbol{r_k} \leftarrow \boldsymbol{b}-\boldsymbol{Ax_k}$

$\boldsymbol{z_k} \leftarrow \boldsymbol{M}^{-1}\,\boldsymbol{r_k}$

$\beta \leftarrow \|\boldsymbol{z_k}\|_2$

$\boldsymbol{V}_{:,0} \leftarrow \boldsymbol{z_k}/\beta$

$\mathbf{s} \leftarrow [\beta, 0, 0, \ldots, 0]^T$

for j=0, 1, 2, … — Iteration count

Single:

$\boldsymbol{w} \leftarrow \boldsymbol{M}^{-1}\,\boldsymbol{AV}_{:,j}$

$\boldsymbol{w}, \boldsymbol{H}_{:,j} \leftarrow orthogonalize(\boldsymbol{w}, \boldsymbol{V}_{:,j})$

$\boldsymbol{H}_{j+1,j} \leftarrow \|\boldsymbol{w}\|_2$

$\boldsymbol{V}_{:,j+1} \leftarrow \boldsymbol{w}/\|\boldsymbol{w}\|_2$

$\boldsymbol{H}_{:,j} \leftarrow \boldsymbol{G_0}\,\boldsymbol{G_1}\ldots\boldsymbol{G_{j-1}}\,\boldsymbol{H}_{:,j}$

$\boldsymbol{G_j} \leftarrow rotation\_matrix(\boldsymbol{H}_{:,j})$

$\boldsymbol{H}_{:,j} \leftarrow \boldsymbol{G_j}\,\boldsymbol{H}_{:,j}$

$\boldsymbol{s} \leftarrow \boldsymbol{G_j}\,\boldsymbol{s}$

$\boldsymbol{u_k} \leftarrow \boldsymbol{V}\boldsymbol{H}^{-1}\,\boldsymbol{s}$

Double:

$\boldsymbol{x_{k+1}} \leftarrow \boldsymbol{x_k} + \boldsymbol{u_k}$

Simplified Mixed Precision GMRES Algorithm

GMRES$_{res}(\boldsymbol{A}, \boldsymbol{x_0}, \boldsymbol{b}, \boldsymbol{M}^{-1})$

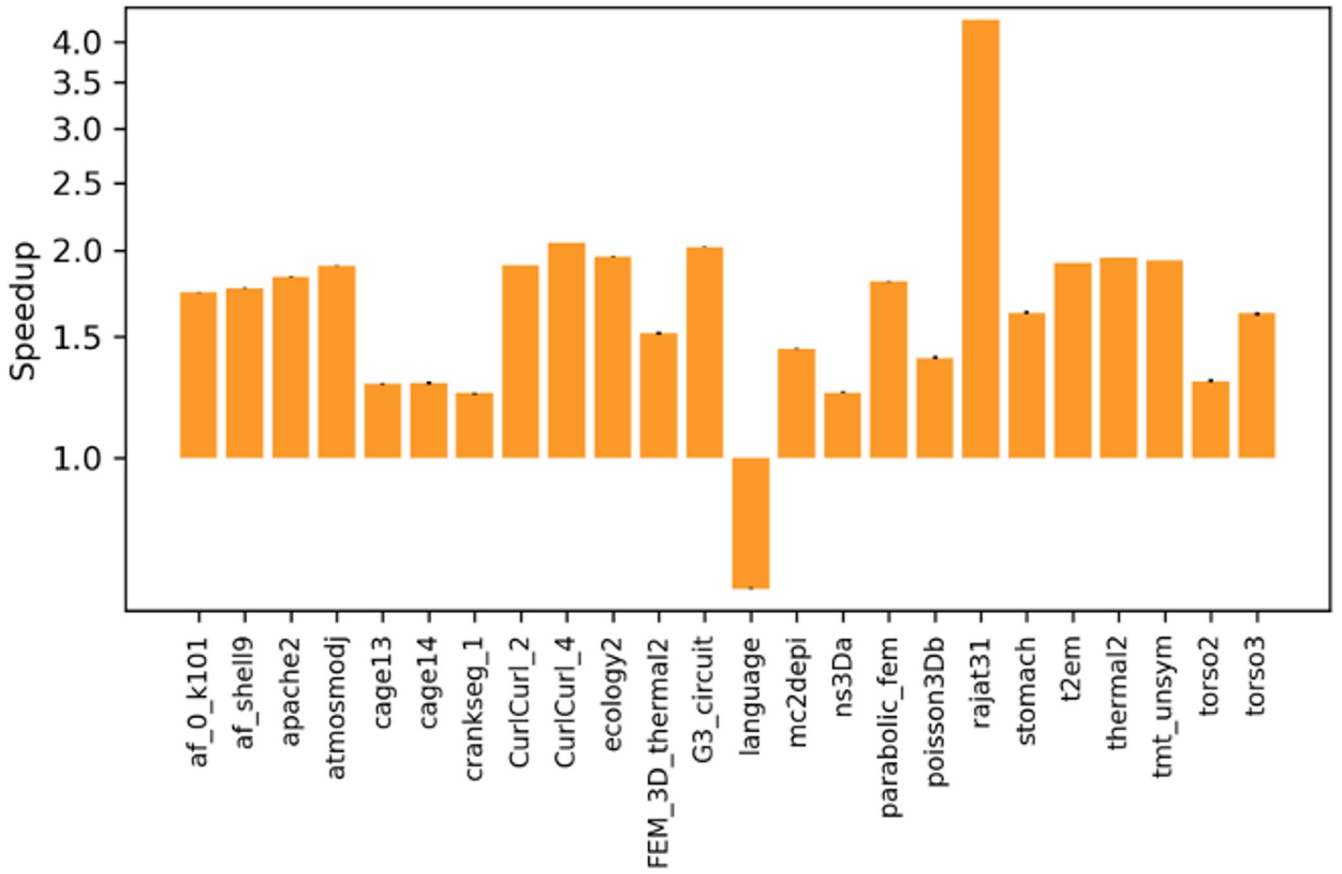
for k=0, 1, 2,…

Double: $\boldsymbol{r_k} \leftarrow \boldsymbol{b} - \boldsymbol{A x_k}$

Single: $\boldsymbol{u_k} \leftarrow$ GMRES$_{\text{no_restart}}(\boldsymbol{A}^{-1}, \boldsymbol{0}, \boldsymbol{r_k}, \boldsymbol{M}^{-1})$

Double: $\boldsymbol{x_{k+1}} \leftarrow \boldsymbol{x_k} + \boldsymbol{u_k}$

Performance Evaluation: No Preconditioner

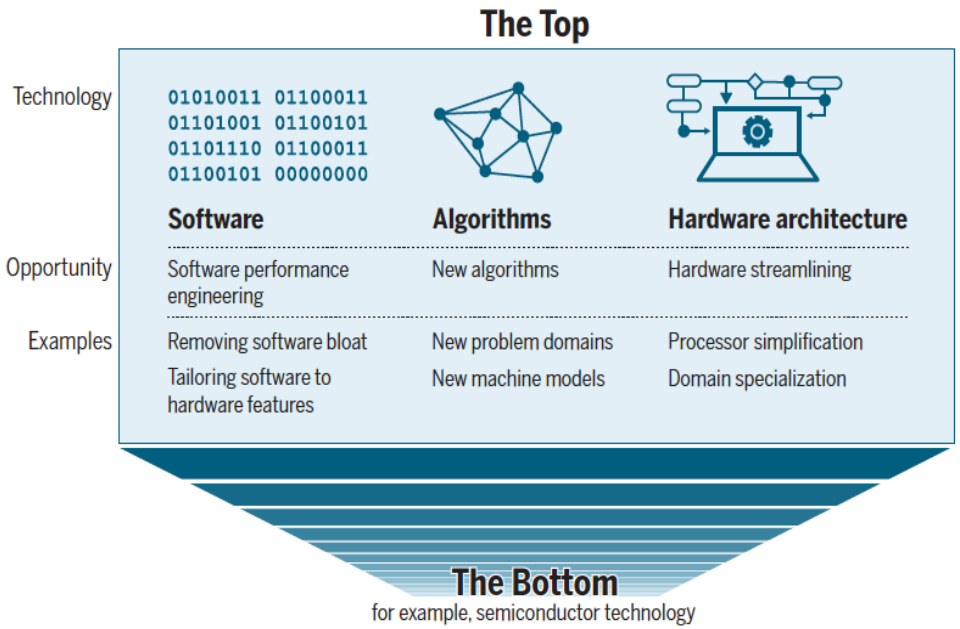


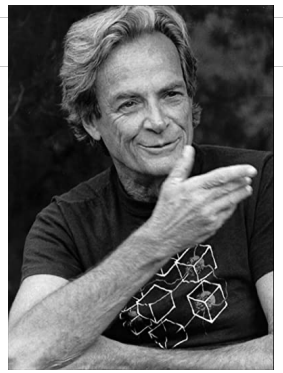
The Take Away

- HPC Hardware is Constantly Changing
  - Scalar
  - Vector
  - Distributed
  - Accelerated
  - Mixed precision
- Three computer revolutions
  - High performance computing
  - Deep learning
  - Edge & AI
- Algorithm / Software advances follows hardware.
  - And there is “plenty of room at the top”

“There’s plenty of room at the Top: What will drive computer performance after Moore’s law?”

Leiserson *et al.*, *Science* **368**, 1079 (2020) 5 June 2020





Feynman’s 1959 Lecture @ CalTech