DP2E-AI 2025

1st International Workshop on Distributed and Parallel Programming for Extreme-scale AI

June 16-17th, 2025 | Paris, France

MINES PARIS | PSL

High-Performance Computing and Responsibly Reckless Algorithms

Jack Dongarra

University of Tennessee

Oak Ridge National Laboratory

University of Manchester

An Accidental Benchmarker

LINPACK was an NSF Project w/ ANL, UNM, UM, & UCSD
We worked independently and came to Argonne in the summers

Top 23 List from 1977
Performance of solving *Ax=b* using LINPACK software

UNIT = 10**6 TIME/(1/3 100**3 + 100**2)

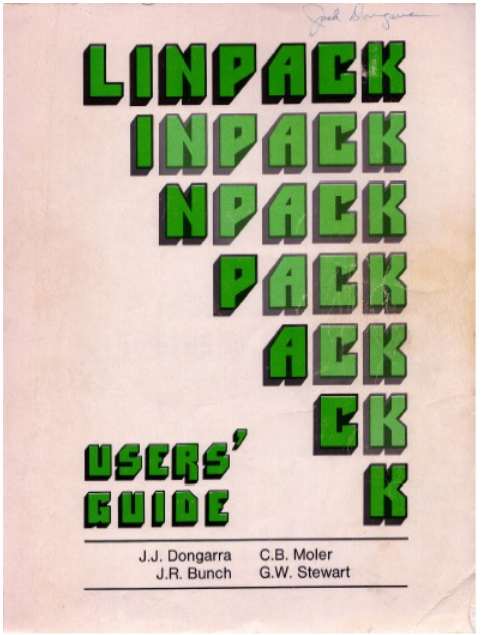
Facility	TIME N=100 secs.	UNIT micro-secs.	Computer	Type	Compiler
NCAR	.049	0.14	CRAY-1	S	CFT, Assembly BLAS
LASL	.148	0.43	CDC 7600	S	FTN, Assembly BLAS
NCAR	.192	0.56	CRAY-1	S	CFT
LASL	.210	0.61	CDC 7600	S	FTN
Argonne	.297	0.86	IBM 370/195	D	H
NCAR	.359	1.05	CDC 7600	S	Local
Argonne	.388	1.33	IBM 3033	D	H
NASA Langley	.489	1.42	CDC Cyber 175	S	FTN
U. Ill. Urbana	.506	1.47	CDC Cyber 175	S	Ext. 4.6
LLL	.554	1.61	CDC 7600	S	CHAT, No optimize
SLAC	.579	1.69	IBM 370/168	D	H Ext., Fast mult.
Michigan	.631	1.84	Amdahl 470/V6	D	H
Toronto	.890	2.59	IBM 370/165	D	H Ext., Fast mult.
Northwestern	1.44	4.20	CDC 6600	S	FTN
Texas	1.93*	5.63	CDC 6600	S	RUN
China Lake	1.95*	5.69	Univac 1110	S	V
Yale	2.59	7.53	DEC KL-20	S	F20
Bell Labs	3.46	10.1	Honeywell 6080	S	Y
Wisconsin	3.49	10.1	Univac 1110	S	V
Iowa State	3.54	10.2	Itel AS/5 mod3	D	H
U. Ill. Chicago	4.10	11.9	IBM 370/158	D	G1

Appendix B of the Linpack Users' Guide

- Designed to help users estimate the run time for solving systems of equation using the Linpack software.

First benchmark report from 1977;

- Cray 1 to DEC PDP-10



LINPACK Benchmark ⟶ Top500

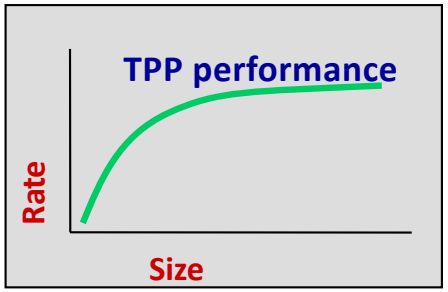
- Since 1978 I maintained a LINPACK Benchmark list.
- Hans Meuer and Erich Strohmaier had a list of fastest computers ranked by peak performance.
- Since 1993 listing of the 500 most powerful computers using 64-bit floating point arithmetic.
- Yardstick: Performance for

  Ax=b, dense problem

Maintained and updated twice a year:

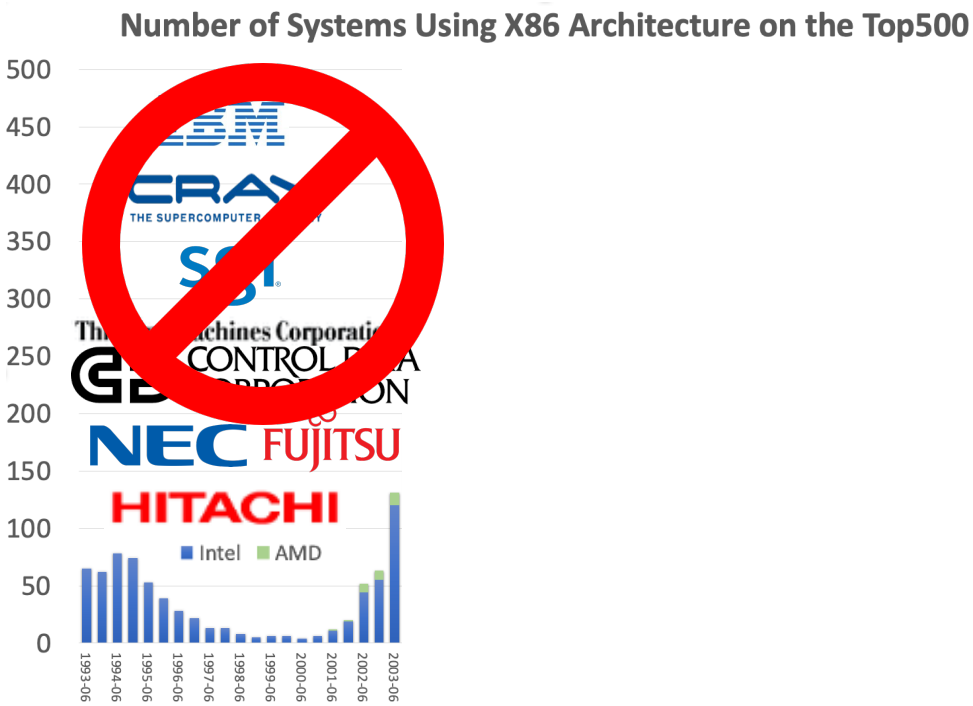
SC‘xy in the States in November

Meeting in Germany in June



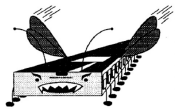
TOP500 The List.

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

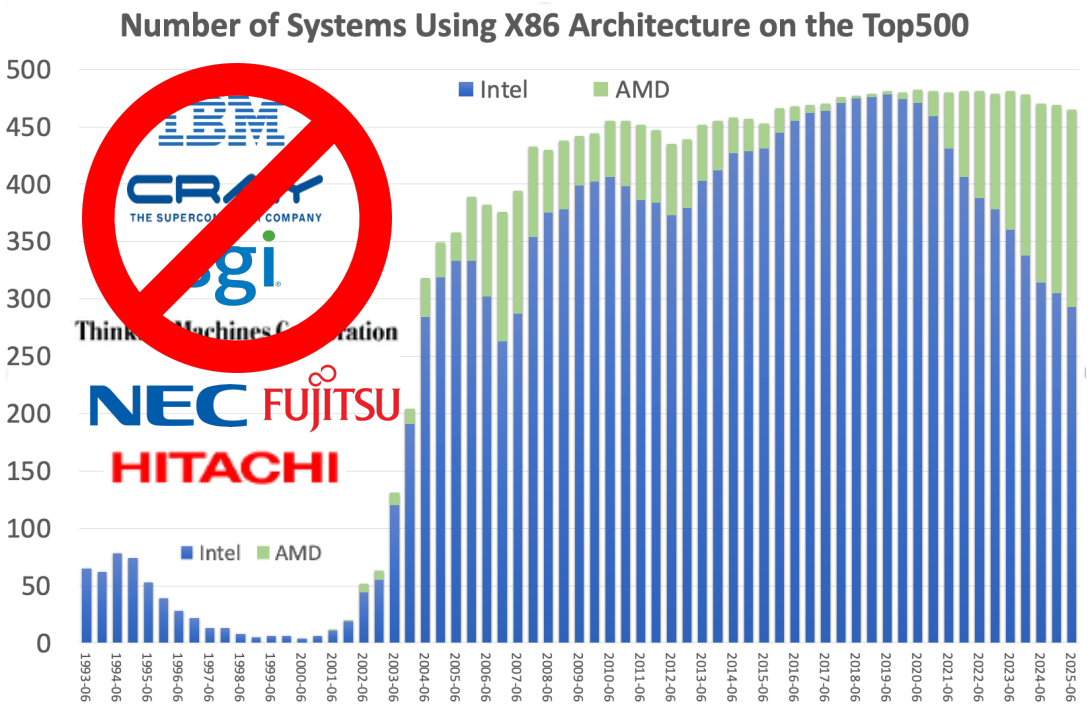


Scientific High Performance Computing based on Commodity Processors

Attach of the Killer Micros



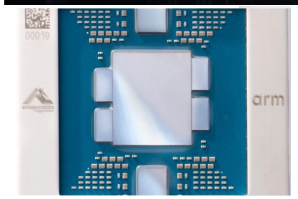
- TOP500 list began in 1993
  - 65 systems used Intel's i860 architecture
  - Remainder had specialized architectures, mainly vector based
- Today's TOP500 list
  - 59% of systems used Intel processors
  - Another 34% used AMD processors
- **93% of the systems use x86-64 architecture**
  - Many use GPU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



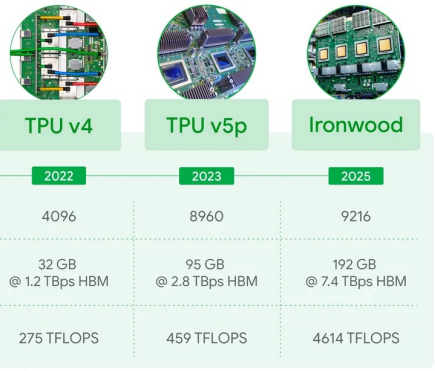
Cloud Vendors are Designing and Using Their Own Processors

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

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



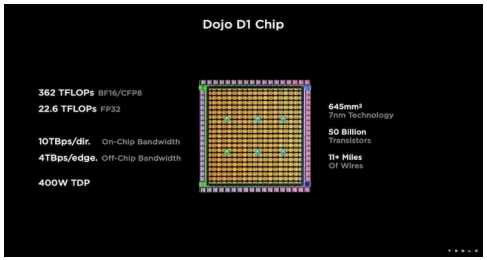
- Google TPU7
  - 2X TPU3 performance
  - 4096 units per "pod"
  - Reconfigurable optical interconnect



- 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

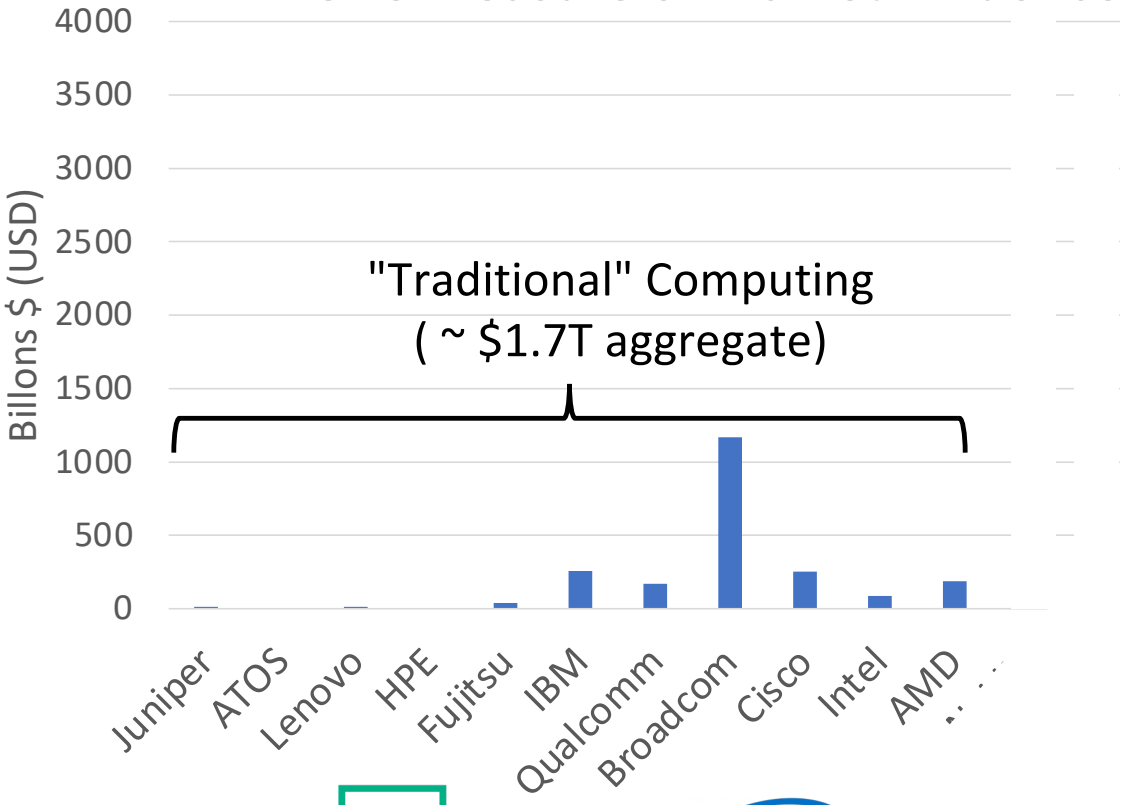
GRAPHCORE

groq

Market Capitalizations

June 15,2025

One measure of market influence



Hewlett Packard Enterprise

intel

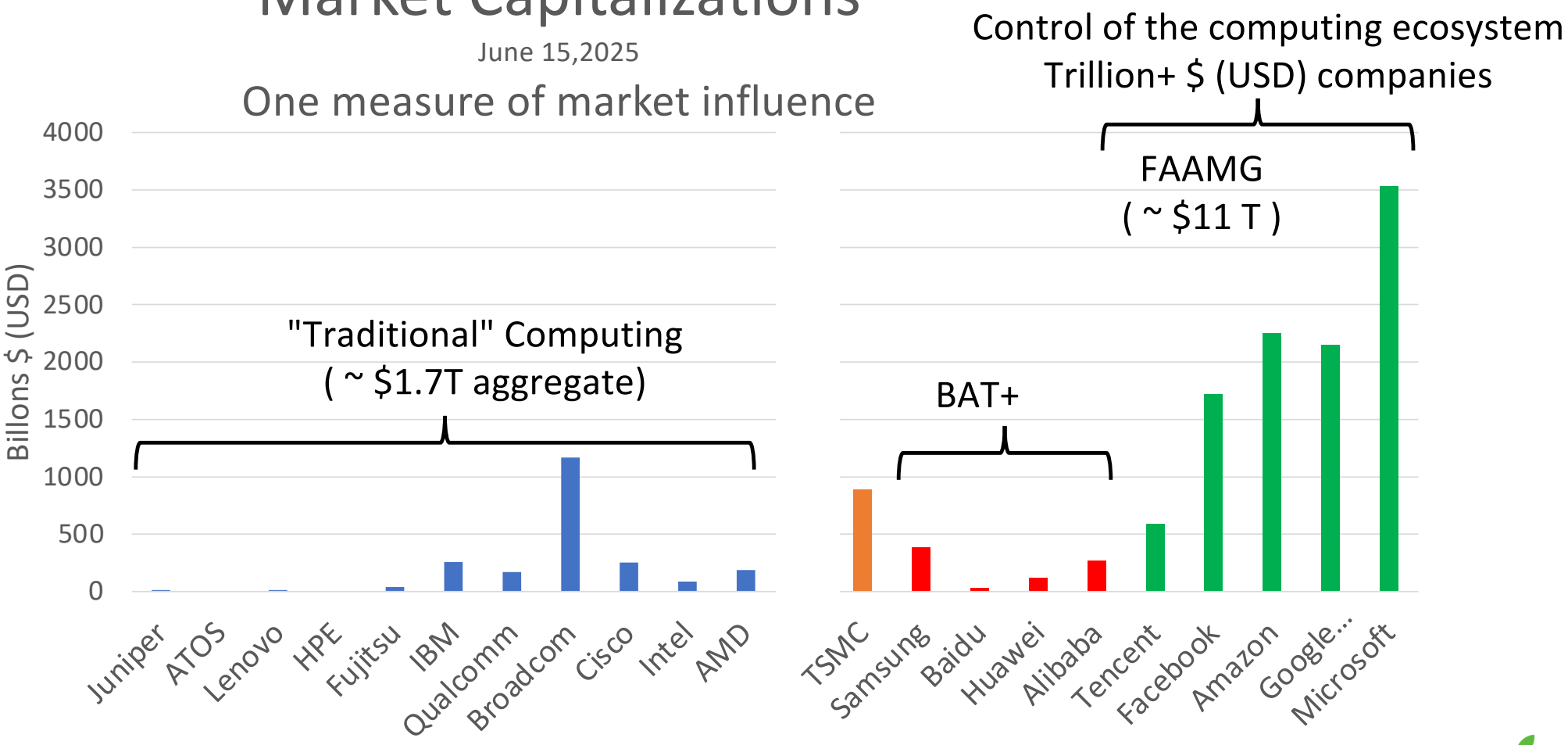
IBM

AMD

Market Capitalizations

June 15,2025

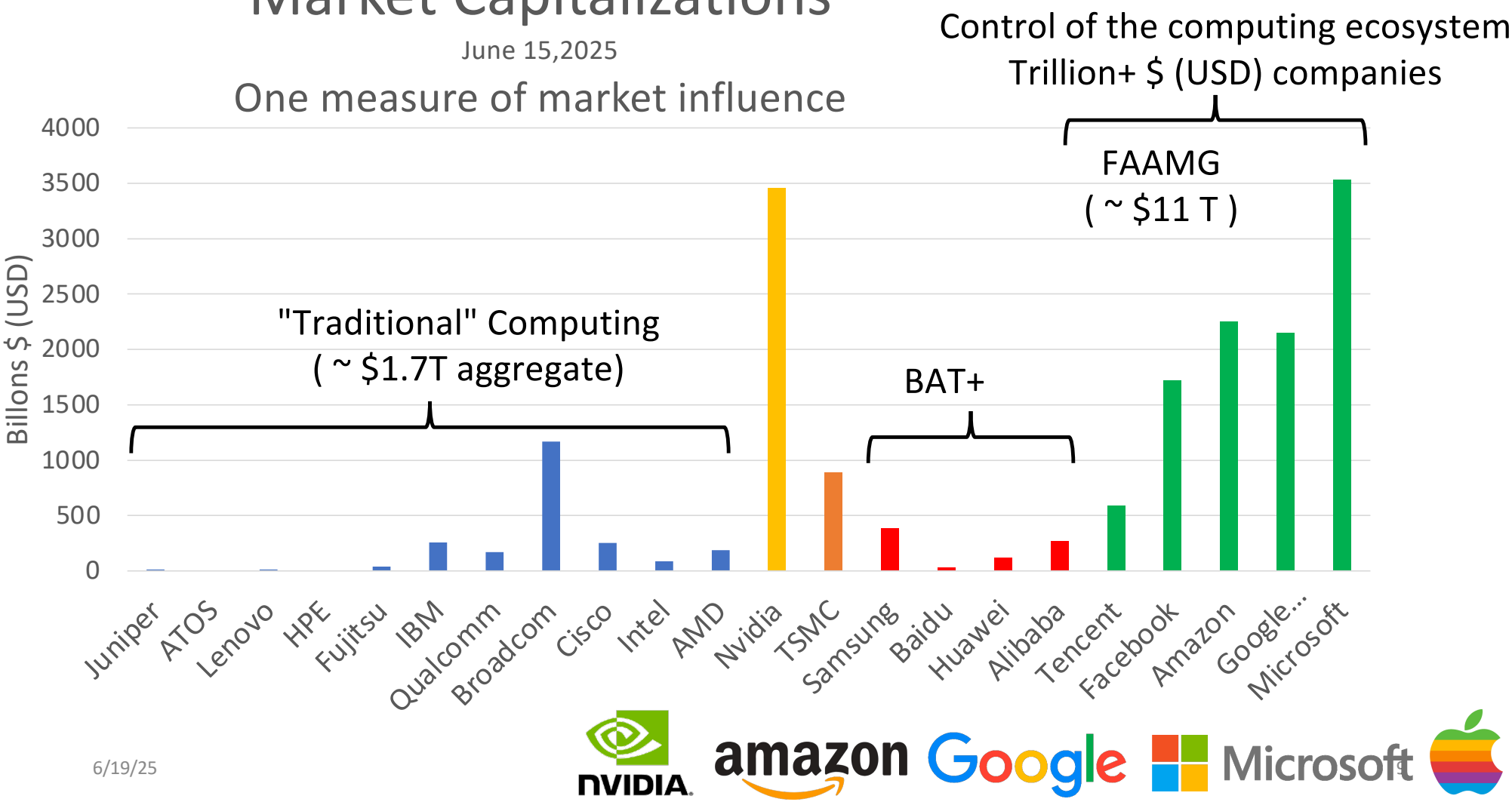
One measure of market influence



Market Capitalizations

June 15,2025

One measure of market influence



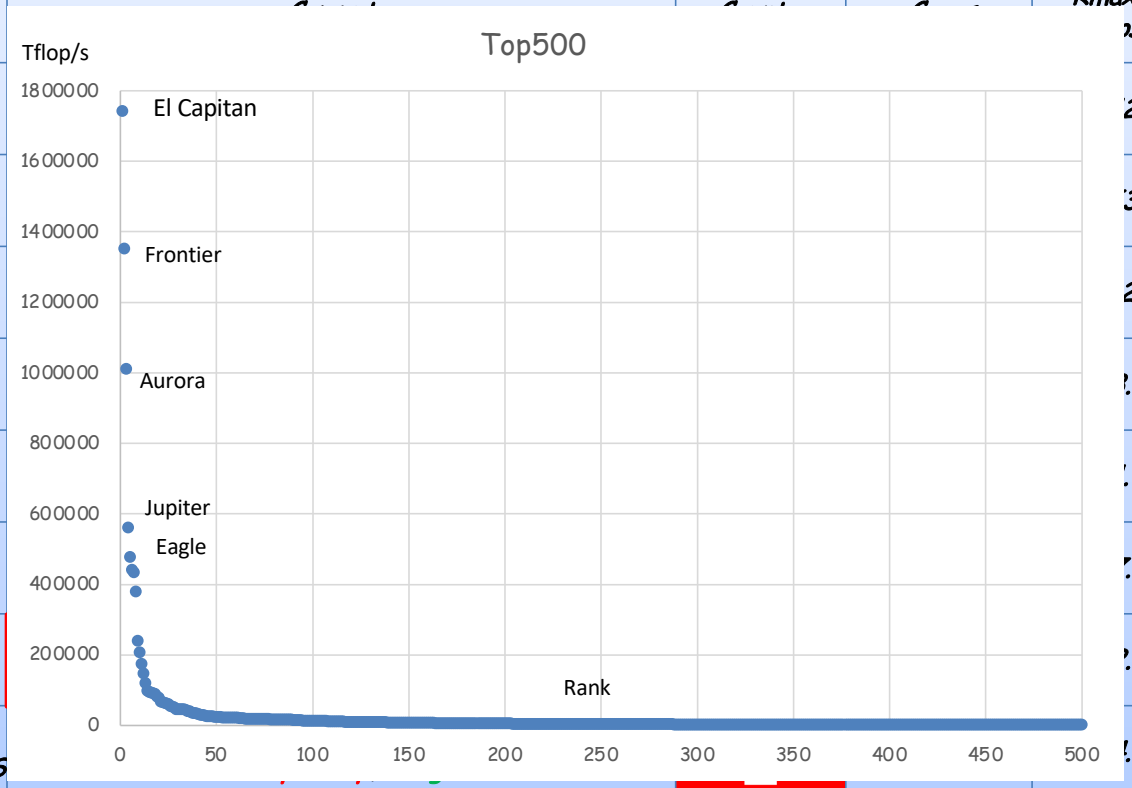
Our HPC Systems are Based on Commodity Parts

- **Commodity Processors**
  - 93% of the Top500 system use X86 (Intel & AMD) instruction set
- **Commodity Accelerators**
  - 86% of accelerated systems use NVIDIA
- **Commodity Interconnect**
  - 87% of the Top500 systems use Ethernet or Infinaband
- **Commodity OS**
  - 100% of the Top500 systems run on Linux

- **Unlike the Hyperscalers**
  - They are building their own processors, accelerators, and interconnects

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

Rank	Site				Rmax [s]	% of Peak	Power [MW]	GFlops/ Watt
1	DOE / NNSA LLNL				2	63	29.5	58.9
2	DOE / OS Oak Ridge Nat Lab				3	65	24.6	55.0
3	DOE / OS Argonne Nat Lab				2	51	38.7	26.1
4	EuroHPC/FZL				.	85	13.1	60.5
5	Microsoft, Azure Cloud				.	66	-	
6	Eni S.p.A.				.	78	8.46	56.5
7	RIKEN Center for Computational Science				.	82	29.9	14.8
8	Swiss National Supercomputing Center CSCS				.	76	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



Elon Musk's xAI Colossus System Used for Training Grok, Musk's LLM for Their Chatbot for X/Twitter

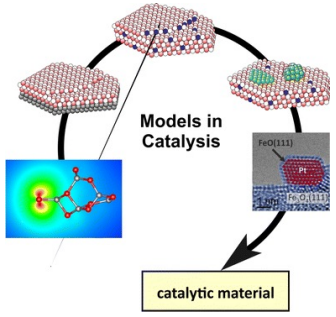
- Built on Nvidia's H100
  - 67 Tflop/s each 64 bit fl pt
    - 495 Tflop/s 32 bit fl pt
    - 990 Tflop/s 16 bit fl pt
    - 1980 Tflop/s 8 bit fl pt
  - 64 GPUs + 16 CPUs / rack
    - 2 CPUs for 8 GPUs
  - 8 racks / group (512 GPUs)
  - 1,500 racks in total
  - Integrated by Supermicro
    - Ethernet 400 Gb/s
- 200,000 H100's
  - 13.4 Eflop/s 64 bit fl pt
    - 100 Eflop/s 32 bit fl pt
    - 200 Eflop/s 16 bit fl pt
    - .4 Zflop/s 8 bit fl pt 10^{21} Ops/s
  - $3-4B Cost
  - 300 MW Power

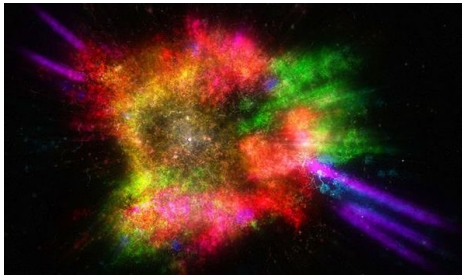
Performance and Benchmarking Evaluation Tools

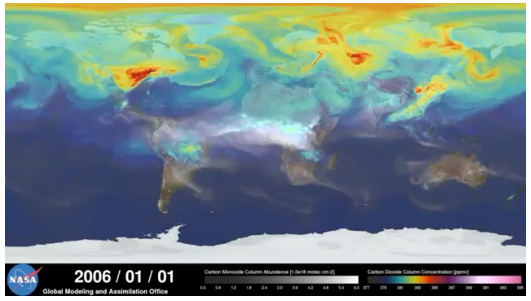
- **Linpack Benchmark - Longstanding benchmark started in 1979**
  - **Lots of positive features; easy to understand and run; shows trends**
- **However, much has changed since 1979**
  - **Arithmetic was expensive then and today it is over-provisioned and inexpensive**
- **Linpack performance of computer systems is no longer strongly correlated to real application performance**
  - **Linpack benchmark based on dense matrix multiplication**
  - **Not "typical" of scientific HPC applications, distorts the field**
- **Designing a system for good Linpack performance can lead to design choices that are wrong for today's applications**

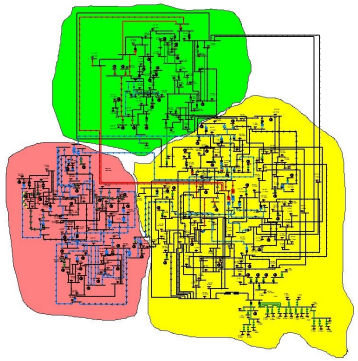
Today's Top HPC Systems Used to do Simulations

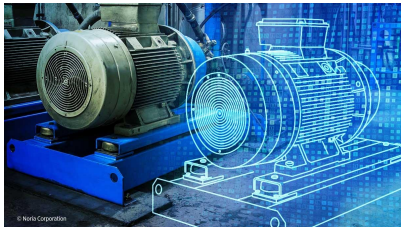
- *Climate*
- *Combustion*
- *Nuclear Reactors*
- *Catalysis*
- *Electric Grid*
- *Fusion*
- *Stockpile*
- *Supernovae*
- *Materials*
- *Digital Twins*
- *Accelerators*
- …

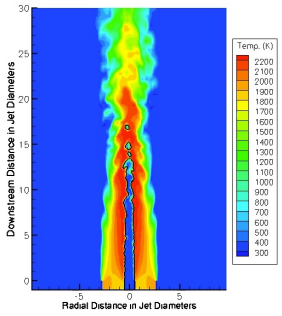


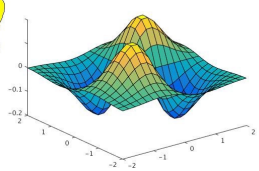


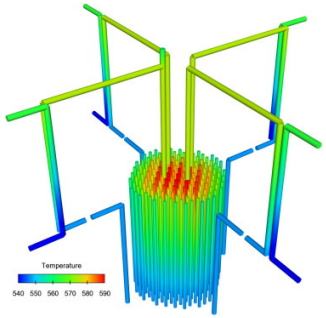


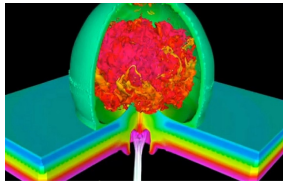


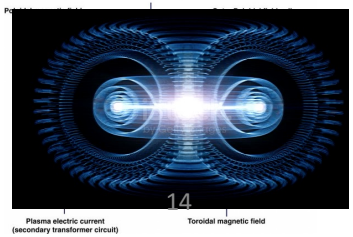










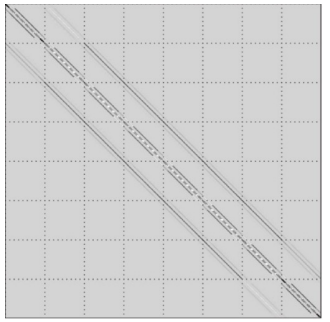


- Usually 3-D PDE's
  - Sparse matrix computations, not dense

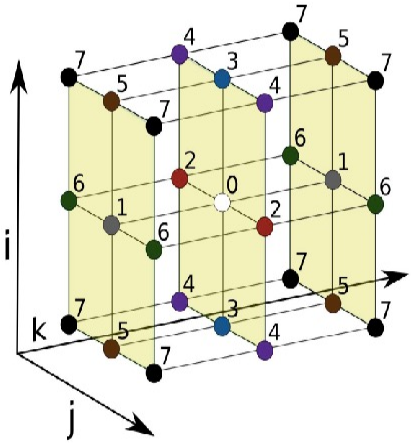
hpcg-benchmark.org With Piotr Luszczek and Mike Heroux

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

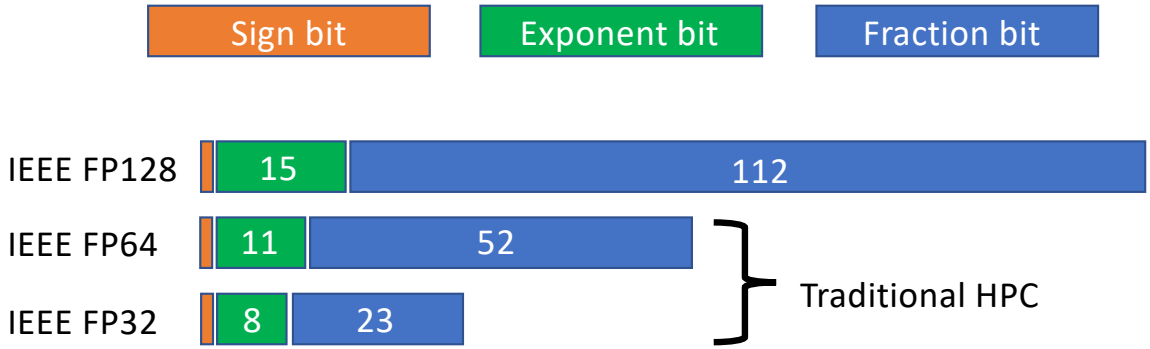
Rank	Site	Computer	Cores	HPL Rmax (Pflop/s)	TOP500 Rank	HPCG (Pflop/s)	Fraction of Peak
1	DOE/SC/LLNL **USA**	**El Capitan**, HPE Cray 255a, AMD 4th Gen EPYC 24C 1.8 GHz, AMD Instinct MI300A, Slingshot-11	11,039,616	1742	1	17.4	**0.6%**
2	RIKEN Center for Computational Science **Japan**	**Fugaku**, Fujitsu 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%**
8	AIST	**ABCI 3.0**, HPE Cray XD670, Xeon Platinum 8558 48C					**1.3%**
9	**USA**	2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-10	888,832	79	25	1.9	**%**
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!

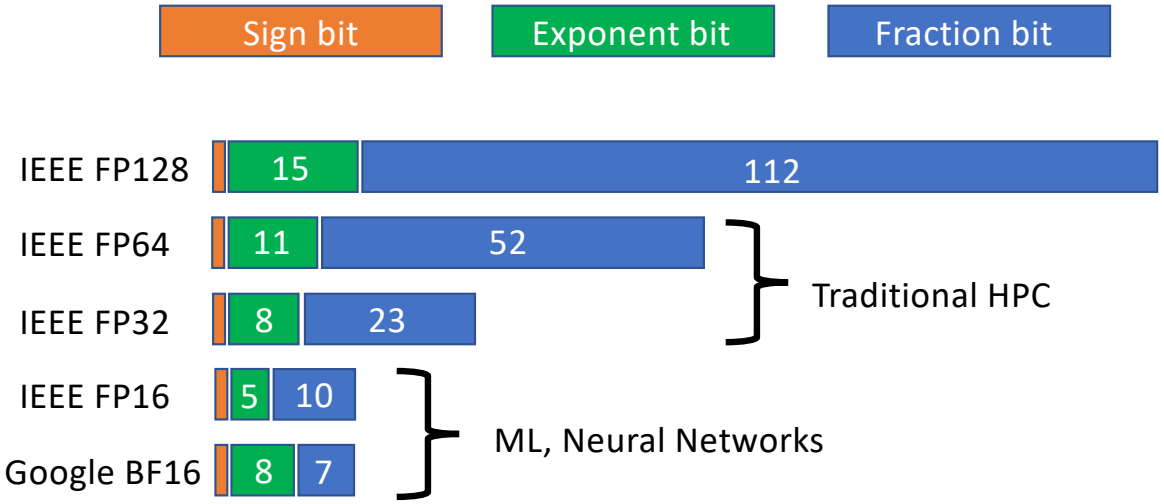
“Responsibly Reckless” Algorithms

- Try a fast algorithm (that may be unstable) and 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

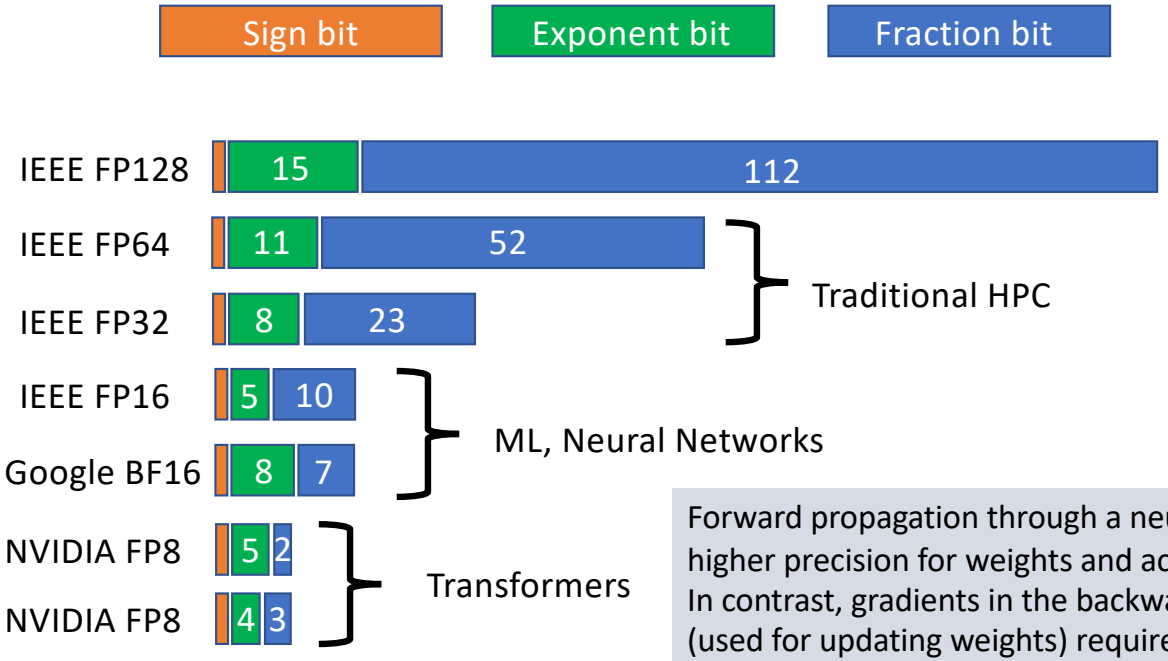


Floating Point Representation



Can we leverage the short precision in our “traditional” 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” numerical computations?

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**
    - **Integers to emulate floating point**
  - **Suitable numerical properties for the algorithm & problems.**

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

Can We Take Advantage of the Hardware? Basically, There are Three Approaches with Mixed Precision

1. Use a mathematical technique
   - Get an approximation in lower precision then use something like Newton's method to enhance accuracy.
   - Emulate floating point in integer arithmetic.
2. Transfer less bytes, data transfer is expensive
   - Store data in primary storage in full precision.
   - Transfer the data in short precision.
     - Could also use data compression techniques
   - Compute using full precision.
3. Use a combination of 1. & 2.

One Idea for Using Mixed Precision Goes Something Like This…

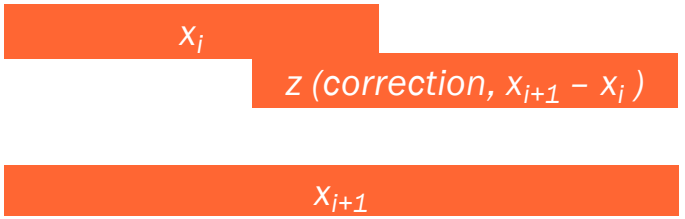
- Exploit lower arithmetic as much as possible.
  - Especially for the bulk of the computation
- Correct or update the solution with selective use of higher floating point arithmetic to provide a “refined results” (more accurate).
- Intuitively:
  - Compute a 32 bit result,
  - Calculate a correction to 32 bit result using selected higher precision (64 bit) and,
  - Perform the update of the 32 bit results with the correction using high precision (64 bit).

Intriguing Potential

- Exploit lower precision as much as possible
  - Payoff in performance
    - Faster floating point
    - Less data to move
  - Use integer arithmetic to emulate floating point
- Automatically switch between SP and DP to match the desired accuracy
  - Compute solution in SP and then a correction to the solution in DP
- Potential for GPU, FPGA, special purpose processors
  - Use as little precision as you can get away with and improve the accuracy
- Linear systems and Eigenvalue, optimization problems, where Newton's method is used.

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

$$x_{i+1} - x_i = -\frac{f(x_i)}{f'(x_i)}$$



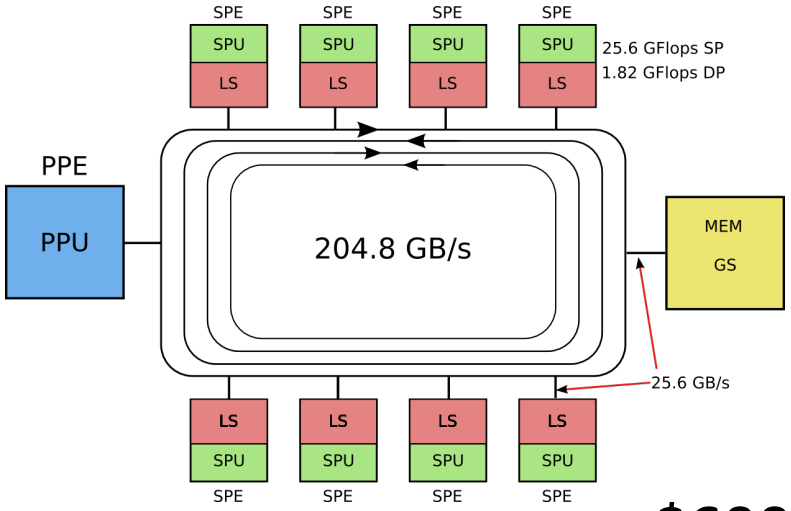
C.T. Kelley, "Newton's Method in Mixed Precision," SIAM Review, Vol 64, No. 1, pp 191-211, 2022

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 $f'(x) = -A$
  - $x_+ = x + A \setminus (b - Ax)$; $r = b - Ax$
  - $(x_+ - x) = A^{-1} * r$
  - $\Delta = (L*U)^{-1} * r$

IBM's Cell Processor - 2004



- 9 Cores
  - Power PC at 3.2 GHz
  - 8 SPEs
- 32 bit fl pt peak at 204.8 Gflop/s peak!
- 64 bit fl pt peak at 14.6 Gflop/s
  - 14 times slower that SP;
    - Factor of 2 from ratio SP:DP and factor of 7 because of latency issues

$600

- Cell Processor was used in the LANL Roadrunner Supercomputer which was #1 in 2008 and the Sony Playstation 3

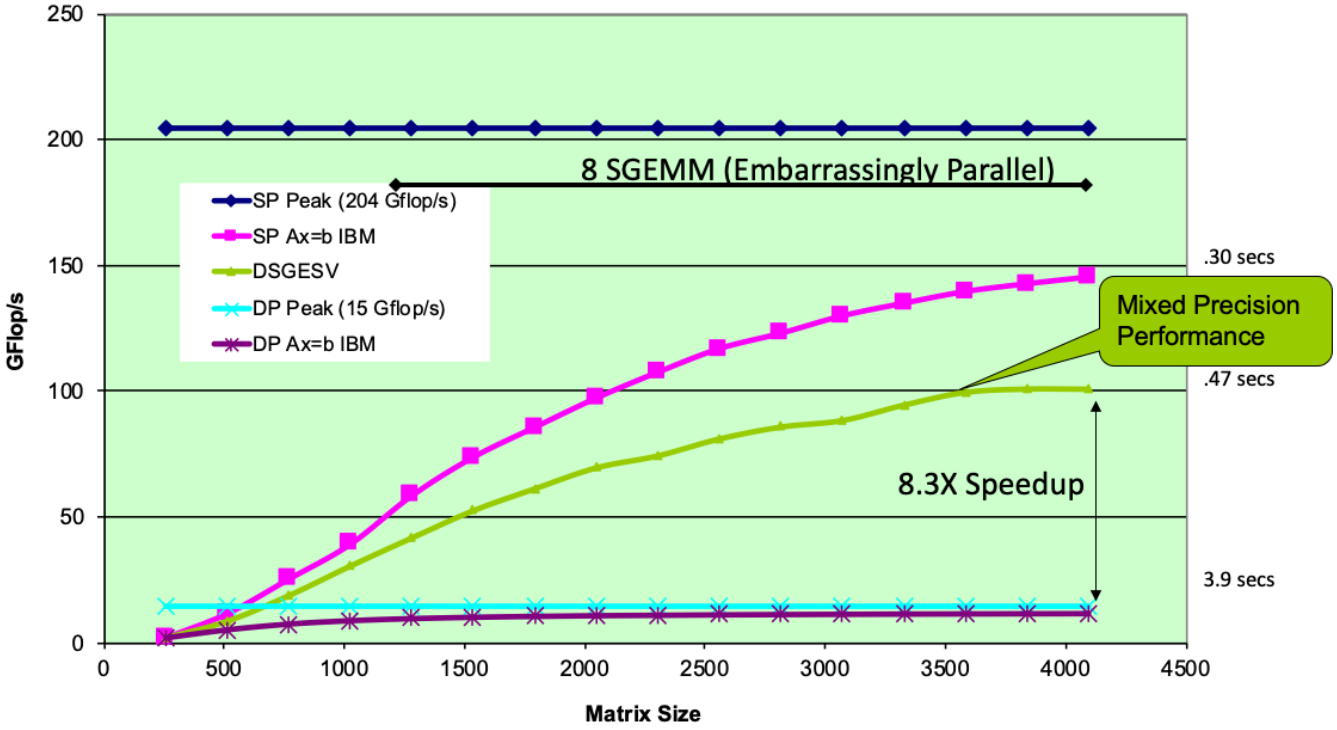
The Cell Processors were fully IEEE-754 compliant in double precision.
In single precision, they only implement round-towards-zero, denormalized numbers are flushed to zero and NaNs are treated like normal numbers.

Leveraging Mixed Precision on Cell Processor

Idea: use low precision to co
($O(n^2)$) the solution in order

```
Iterative refinement for dense
L U = lu(A)
x = U\(L\b)
r = b – Ax (with original A)

WHILE || r || not small enough
      1.  find a correction "z" f
      2.  x = x + z
      3.  r = b – Ax (with origin
END
```



- Wilkinson, Moler, Stewart, & Higham pr
- It can be shown that using this approacl
- Need a copy of the original matrix to compute residual (r) and matrix cannot be too badly conditioned

J. Langou, J. Langou, P. Luszczek, J. Kurzak, A. Buttari, and J. J. Dongarra. Exploiting the performance of 32 bit floating point arithmetic in obtaining 64 bit accuracy. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, 2006.

Leveraging Mixed Precision for Linear Algebra

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: ➢ 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.

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

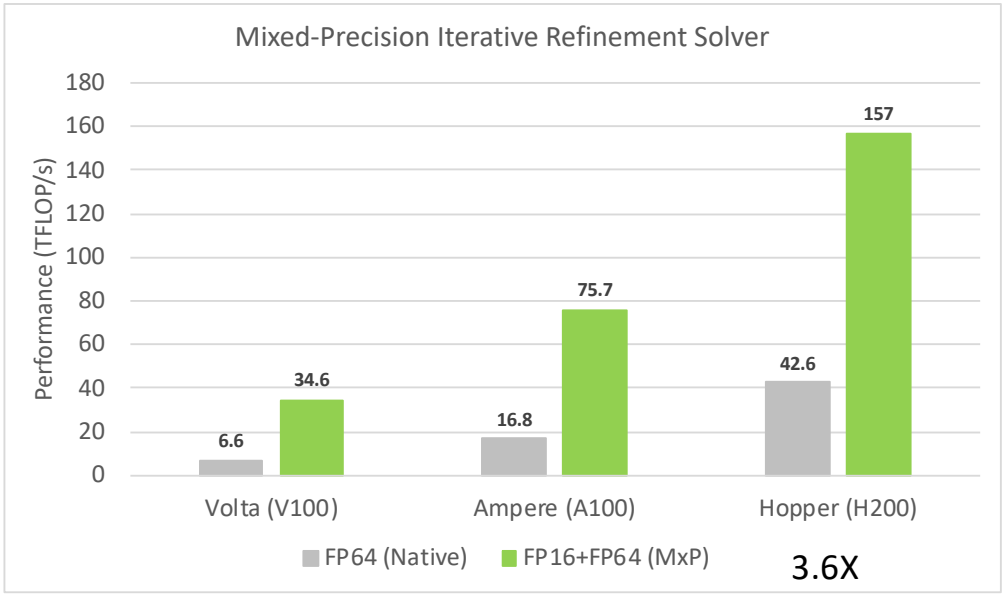
J. Langou, et al., Exploiting the Performance of 32 bit fl-pt Arithmetic in Obtaining 64 bit Accuracy, in: Proc. of SC06

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

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

Mixed-Precision Iterative Refinement Solver

Performance and Efficiency Improvements Across Three Generations



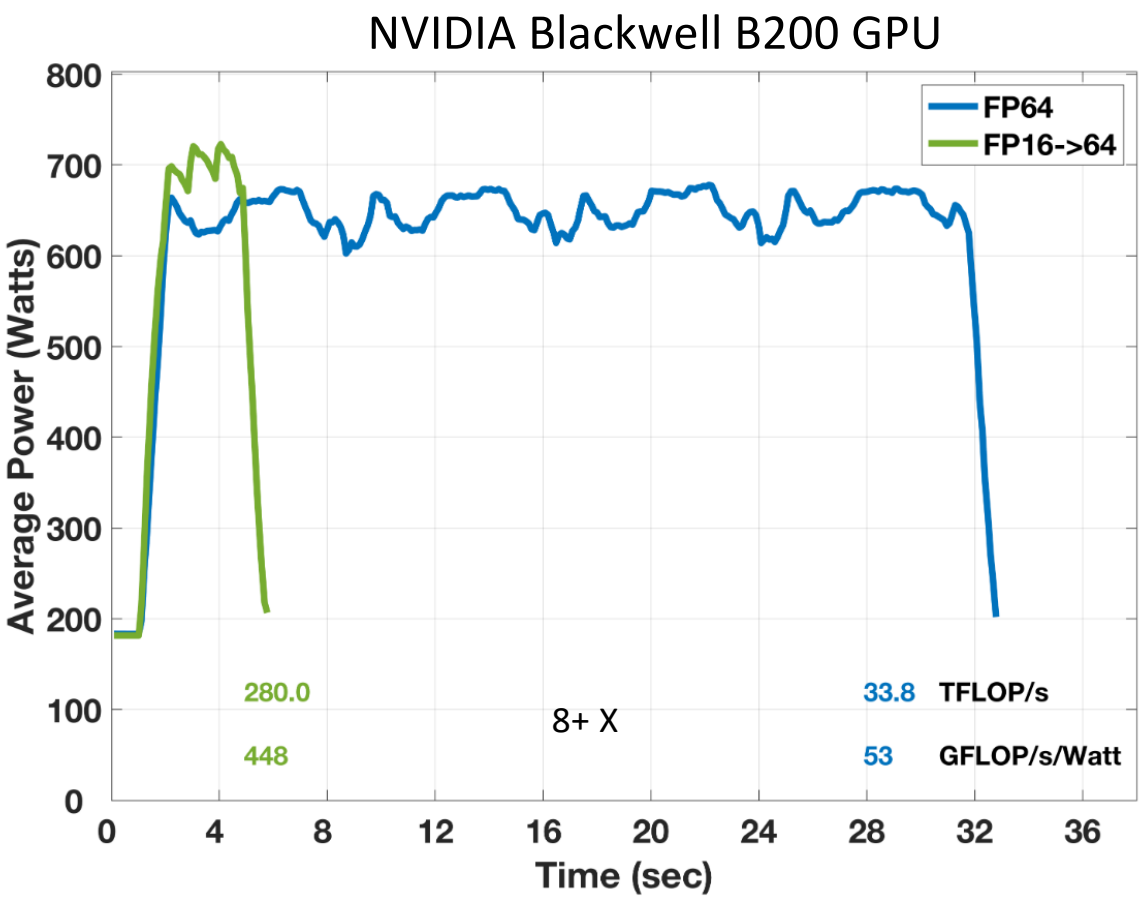
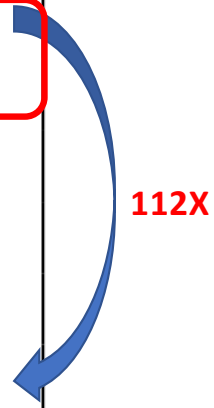

NVIDIA Blackwell B200 GPU
Average Power (Watts)
Time (sec)
FP64
FP16->64
280.0
448
8+ X
33.8 TFLOP/s
53 GFLOP/s/Watt
0
100
200
300
400
500
600
700
800
0
4
8
12
16
20
24
28
32
36

Figure of Merit	Volta 2017 (V100)	Ampere 2020 (A100)	Hopper 2022 (H200)
FP64 FMA (TFLOP/s)	7.8	9.75	33.5
FP64 Tensor (TFLOP/s)	N/A	19.5	67
FP32 FMA (TFLOP/s)	15.7	19.5	67
FP16 Tensor (TFLOP/s)	125	312	989
BF16 Tensor (TFLOP/s)	125	312	989
INT8 Tensor (TOP/s)	N/A	624	1979
Memory BW (TB/s)	0.9	2.0	4.8

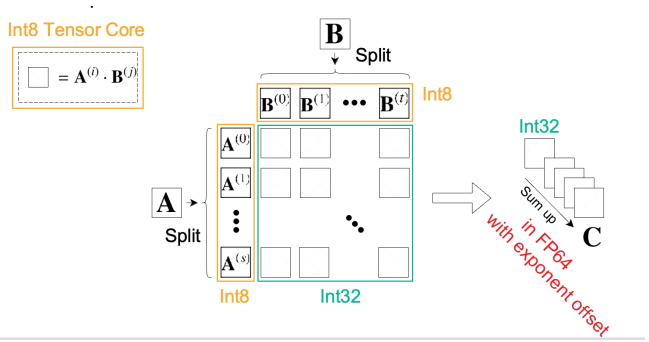
Figure of Merit	Volta 2017 (V100)	Ampere 2020 (A100)	Hopper 2022 (H200)	Blackwell 2024 (B200)
FP64 FMA (TFLOP/s)	7.8	9.75	33.5	40
FP64 Tensor (TFLOP/s)	N/A	19.5	67	40
FP32 FMA (TFLOP/s)	15.7	19.5	67	80
FP16 Tensor (TFLOP/s)	125	312	989	2250
BF16 Tensor (TFLOP/s)	125	312	989	2250
INT8 Tensor (TOP/s)	N/A	624	1979	4500
Memory BW (TB/s)	0.9	2.0	4.8	8.0



Opportunity Breeds Innovation, 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

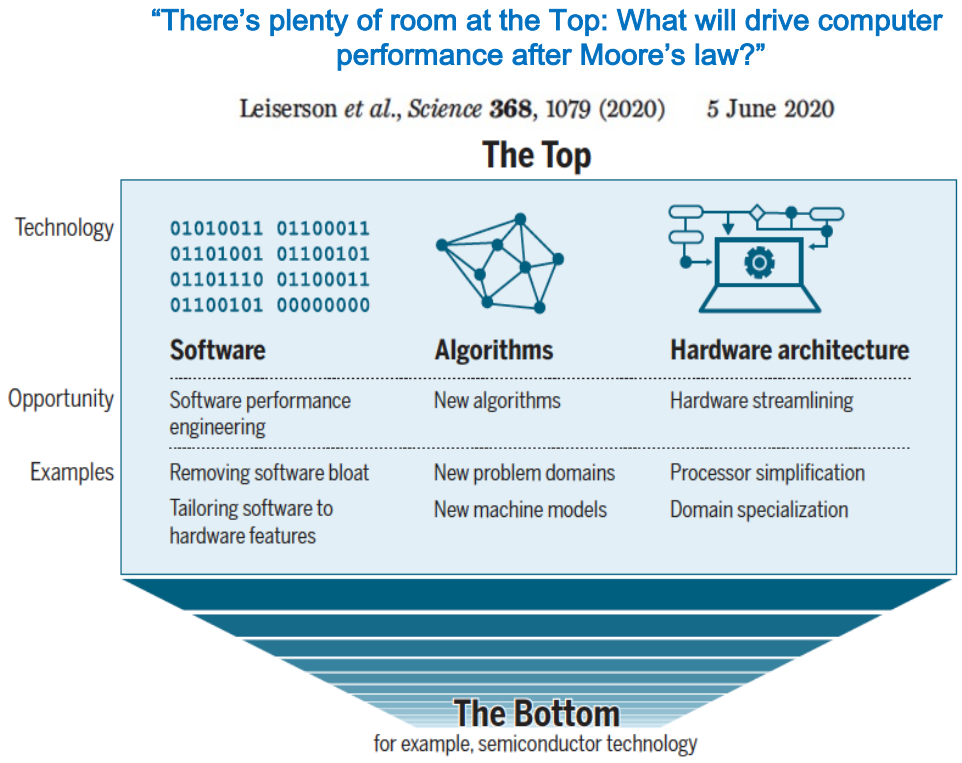
Analysis of Floating-Point Matrix Multiplication Computed via Integer Arithmetic

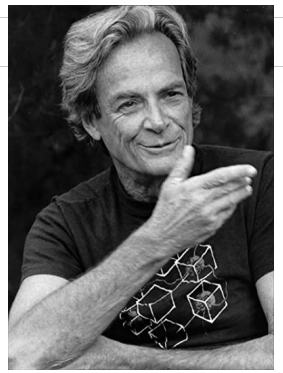
Ahmad Abdelfattah, Jack Dongarra, Massimiliano Fasi, Mantas Mikaitis, Françoise Tisseur

http://arxiv.org/abs/2506.11277

The Take Away

- HPC Hardware is Constantly Changing
  - Scalar
  - Vector
  - Distributed
  - Accelerated
  - Mixed precision
- Algorithm / Software advances follows hardware.
  - And there is “plenty of room at the top”





Feynman’s 1959 Lecture @ CalTech