Reduced Precision and Data Compression in Scientific Applications
DAREXA-F Workshop, December 2025

MAX PLANCK
COMPUTING & DATA FACILITY

# Mixed Feeling about Mixed Precision:
# Can we adapt Numerical Algorithms to AI Hardware?

Hartwig Anzt, TU Munich & University of Tennessee

# What are your feelings about mixed precision?

What are your feelings about mixed precision?

- ○ positive
- ○ negative
- ○ mixed
- ○ Indifferent, I work only on the theoretical side.

# What are your feelings about mixed precision?

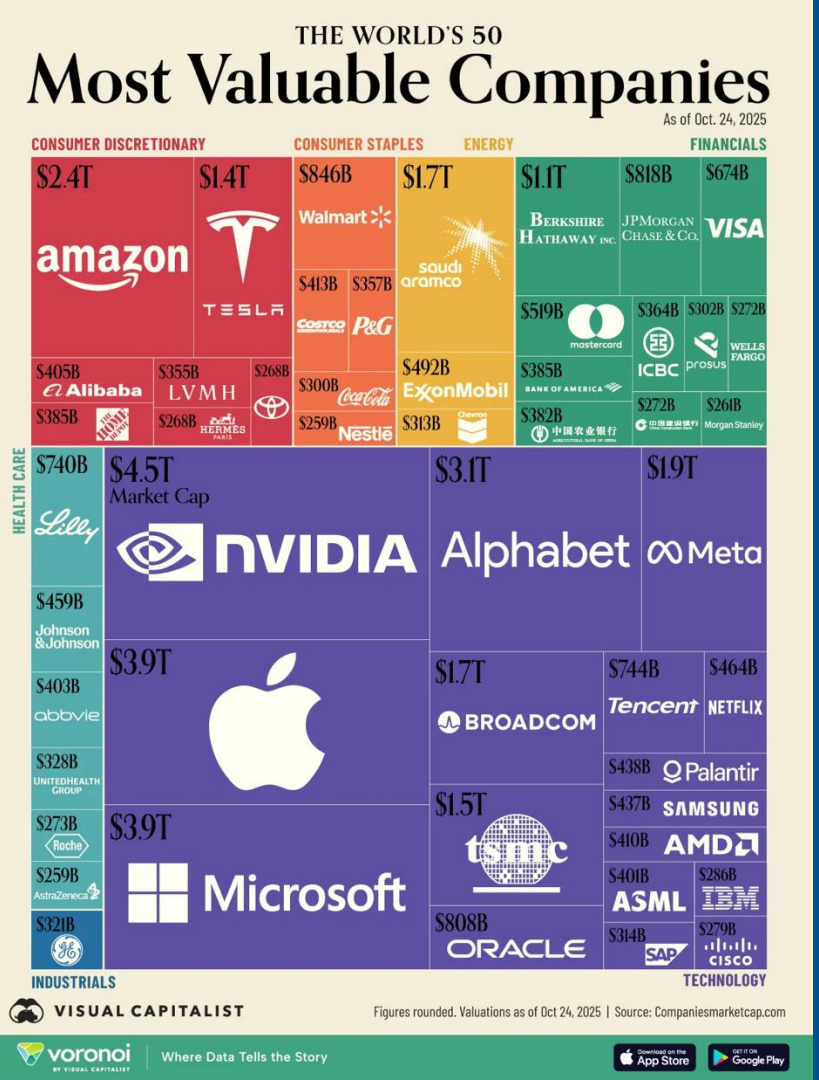
Which floating point formats do you use?

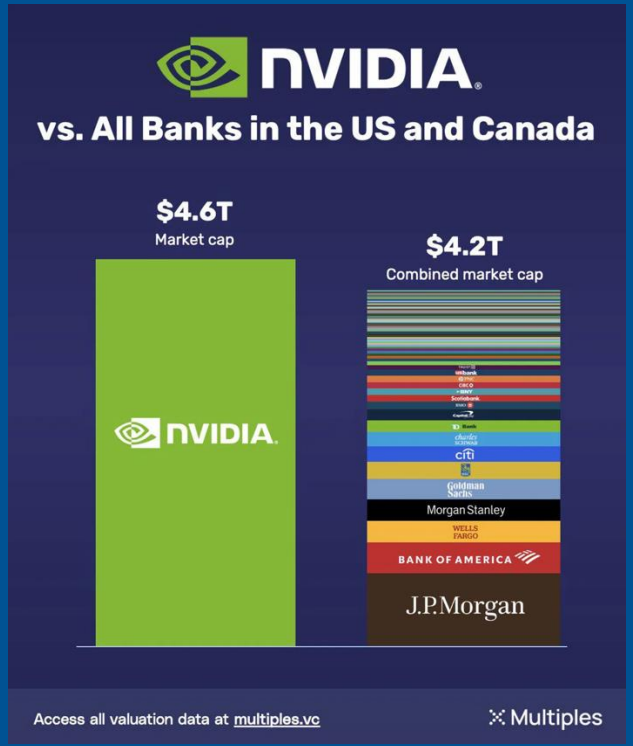
- [ ] IEEE FP64 (double precision)
- [ ] IEEE FP32 (single precision)
- [ ] TF32 (Tensor float 32)
- [ ] lower than FP32
- [ ] What is a floating point format? The only values I need are π, i, and e.

# What are your feelings about mixed precision?

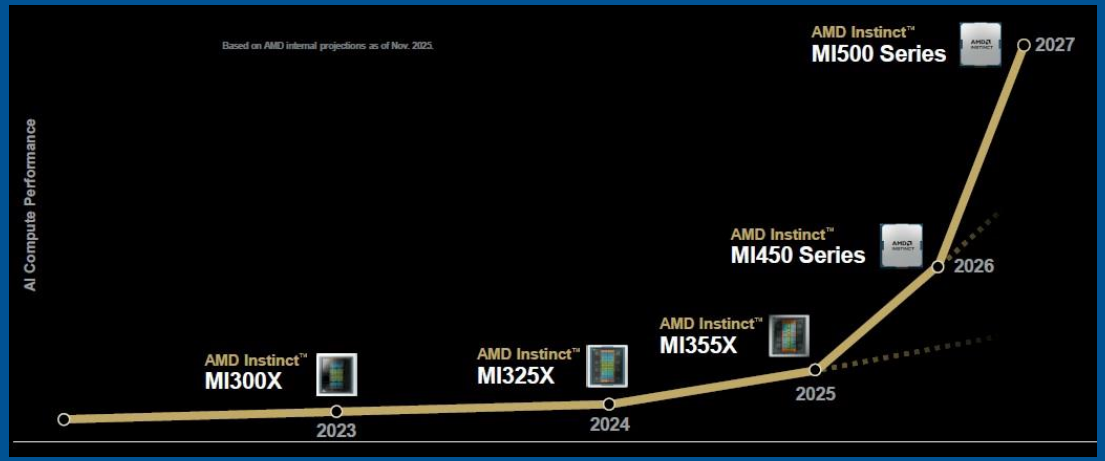
What is the biggest perk of mixed precision?

- ○ Faster compute
- ○ Faster memory access and communication
- ○ Smaller memory footprint (also on disk)
- ○ All of the above, sign me up for the community effort!

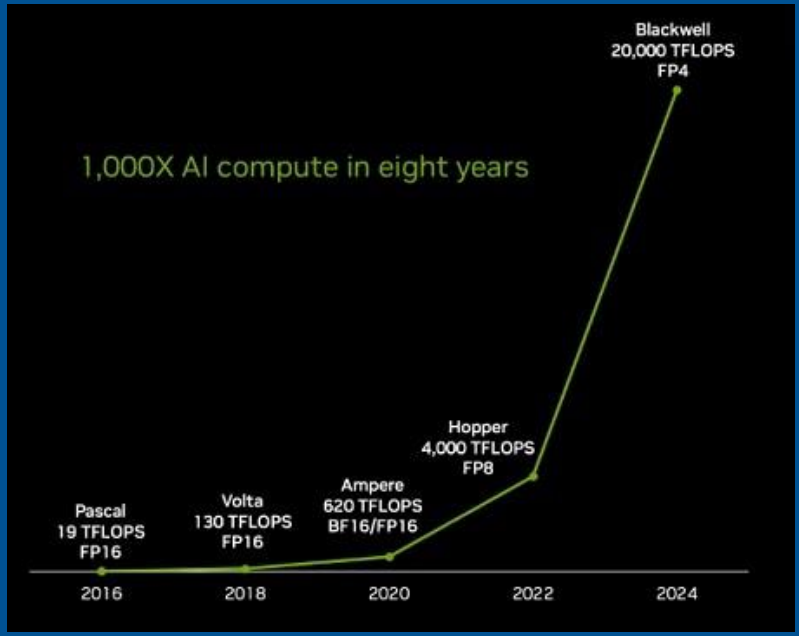

THE WORLD'S 50
Most Valuable Companies
As of Oct. 24, 2025
CONSUMER DISCRETIONARY
CONSUMER STAPLES
ENERGY
FINANCIALS
$2.4T
amazon
$1.4T
TESLA
$846B
Walmart
$1.7T
saudi aramco
$1.1T
BERKSHIRE HATHAWAY INC.
$818B
JPMORGAN CHASE & CO.
$674B
VISA
$413B
$357B
Costco
P&G
$519B
mastercard
$364B
ICBC
$302B
prosus
$272B
WELLS FARGO
$405B
Alibaba
$355B
LVMH
$268B
$492B
ExxonMobil
$385B
BANK OF AMERICA
$300B
Coca-Cola
$385B
$268B
HERMES
$259B
Nestlé
$313B
Chevron
$382B
$272B
$261B
Morgan Stanley
HEALTH CARE
$740B
Lilly
$4.5T
Market Cap
NVIDIA
$3.1T
Alphabet
$1.9T
Meta
$459B
Johnson & Johnson
$3.9T
$1.7T
BROADCOM
$744B
Tencent
$464B
NETFLIX
$403B
abbvie
$328B
UNITEDHEALTH GROUP
$438B
Palantir
$437B
SAMSUNG
$1.5T
TSMC
$410B
AMD
$273B
Roche
$3.9T
Microsoft
$401B
ASML
$286B
IBM
$259B
AstraZeneca
$808B
ORACLE
$314B
SAP
$279B
cisco
$321B
INDUSTRIALS
TECHNOLOGY
VISUAL CAPITALIST
Figures rounded. Valuations as of Oct 24, 2025 | Source: Companiesmarketcap.com
voronoi
BY VISUAL CAPITALIST
Where Data Tells the Story
Download on the App Store
GET IT ON Google Play


NVIDIA
vs. All Banks in the US and Canada
$4.6T
Market cap
NVIDIA
$4.2T
Combined market cap
citi
Goldman Sachs
Morgan Stanley
WELLS FARGO
BANK OF AMERICA
J.P.Morgan
Access all valuation data at multiples.vc
Multiples

# Evolution of GPU Hardware

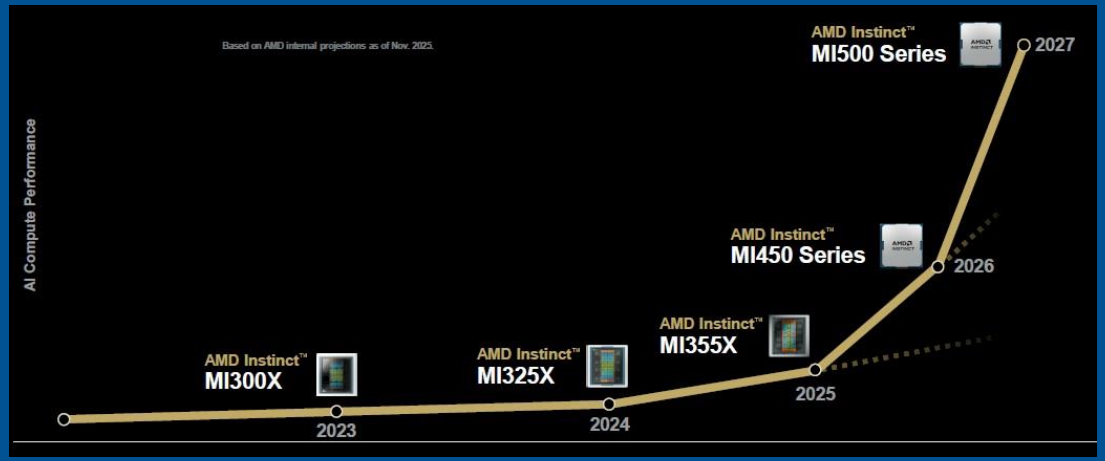



AMD

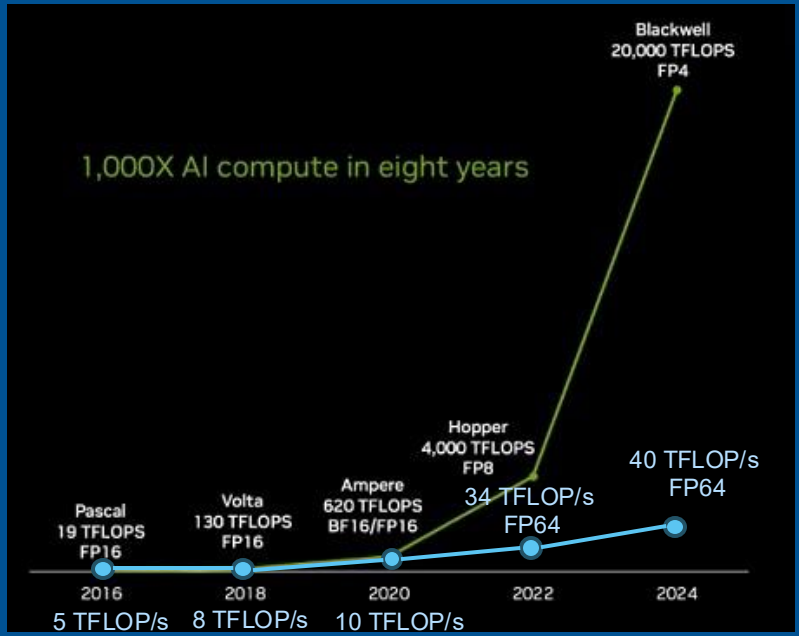



NVIDIA

# Evolution of GPU Hardware

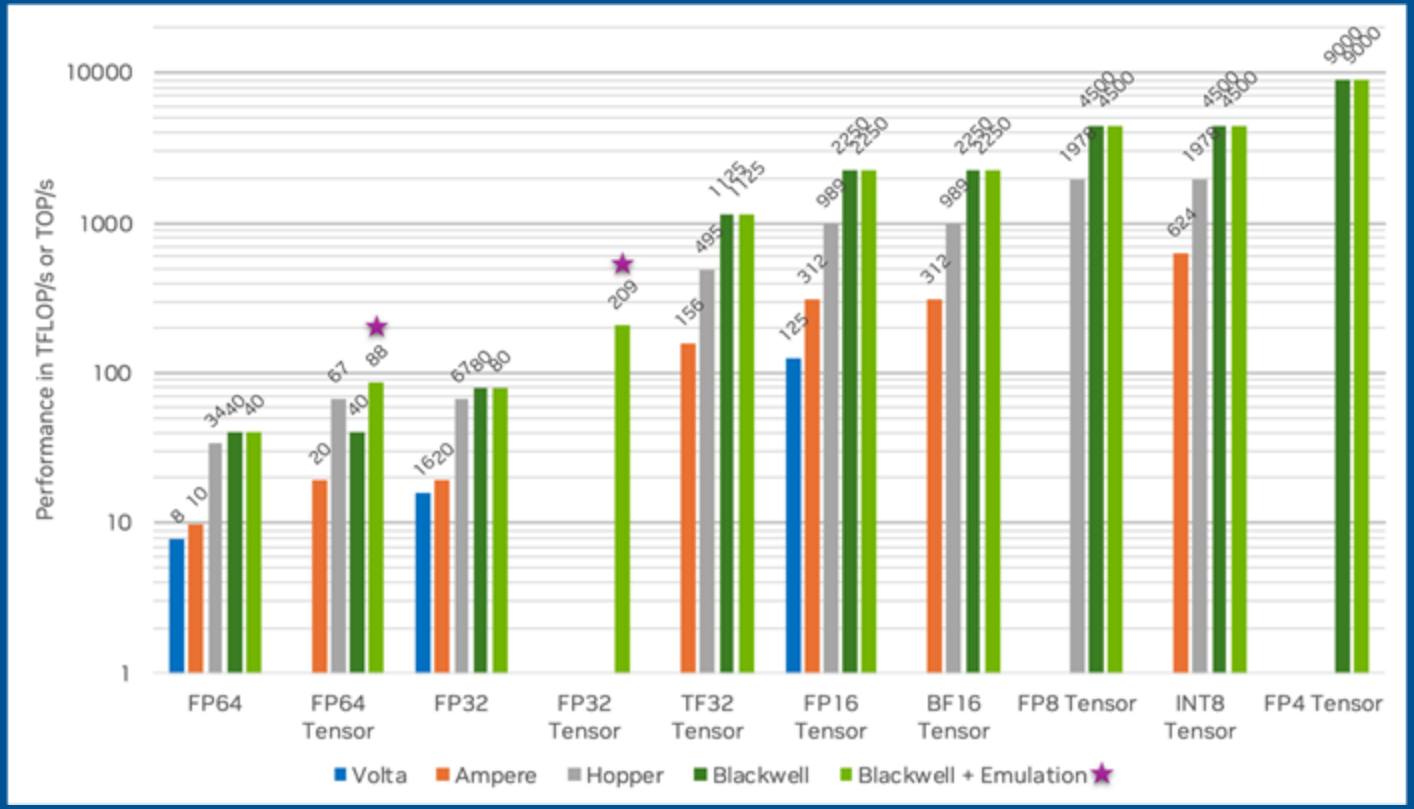


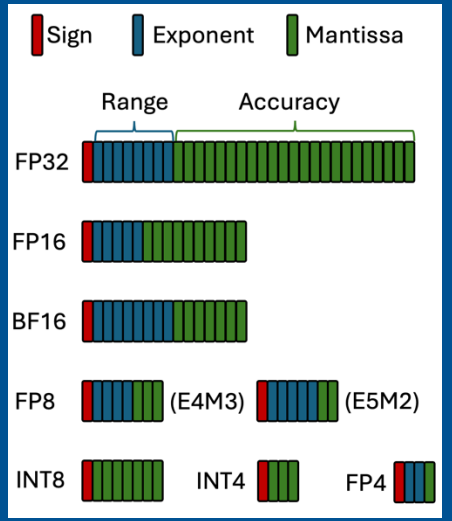

AMD




NVIDIA

# Evolution of GPU Hardware




Blackwell + Emulation
(1)FP64 Tensor uses Ozaki-I on GB200 with 10% ESC+ADP overhead
(2)FP32 Tensor uses BF16x9

# Ozaki Scheme (Ozaki I)

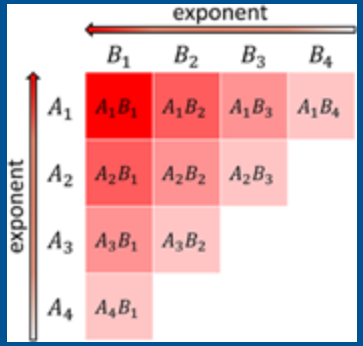
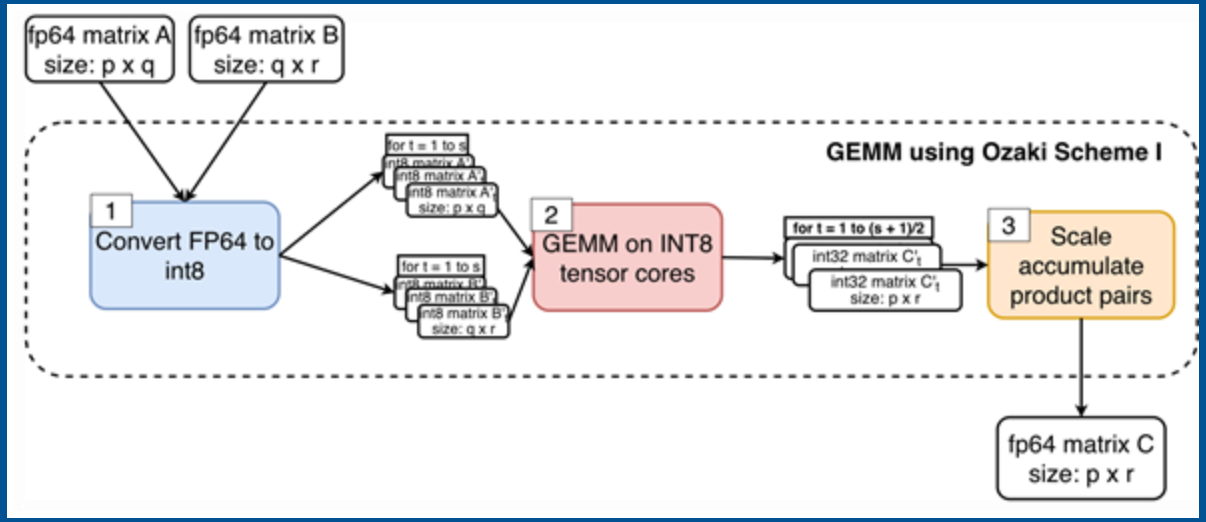
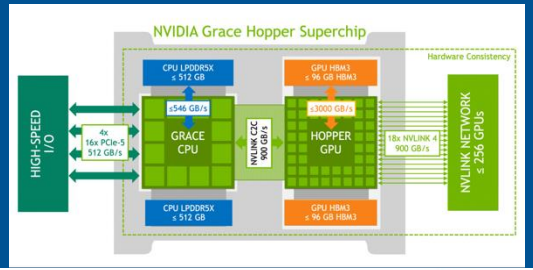



**Table 2.** Relation between the number of slices and the number of matrix multiplications

| slices | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| muls. | 3 | 6 | 10 | 15 | 21 | 28 | 36 | 45 | 55 |



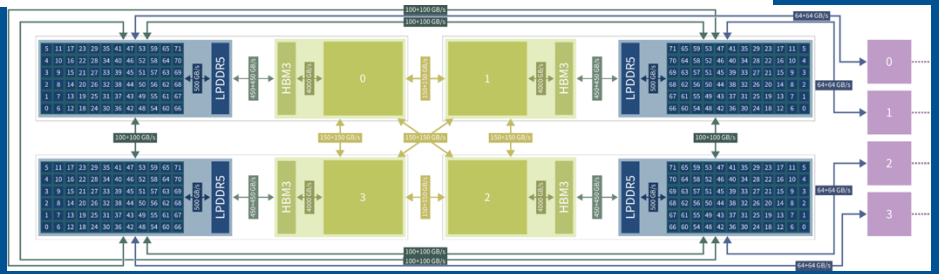

# Solving a real-world problem




| | |
|---|---|
| **FP64** | 34 teraFLOPS |
| **FP64 Tensor Core** | 67 teraFLOPS |
| **FP32** | 67 teraFLOPS |
| **TF32 Tensor Core** | 989 teraFLOPS* \| 494 teraFLOPS |
| **BFLOAT16 Tensor Core** | 1,979 teraFLOPS* \| 990 teraFLOPS |
| **FP16 Tensor Core** | 1,979 teraFLOPS* \| 990 teraFLOPS |
| **FP8 Tensor Core** | 3,958 teraFLOPS* \| 1,979 teraFLOPS |
| **INT8 Tensor Core** | 3,958 TOPS* \| 1,979 TOPS |
| **High-bandwidth memory (HBM) size** | 96GB HBM3 \| 144GB HBM3e |
| **Memory bandwidth** | Up to 4TB/s \| Up to 4.9TB/s |
| **NVIDIA NVLink-C2C CPU-to-GPU bandwidth** | 900 GB/s |

* With sparsity

JUPITER



# Performance Limiters

*Assumption: Compute and Communicate can overlap*

$$\text{runtime} = \max\{T_{\text{compute}}, \quad T_{\text{communicate}}\}$$

$$= \max\{\frac{\text{algorithm work}}{\text{compute performance}}, \quad \frac{\text{data volume}}{\text{bandwidth}}\}$$

# Performance Limiters

*Assumption: Compute and Communicate can overlap*

$$\text{runtime} = \max\{T_{\text{compute}}, \quad T_{\text{communicate}}\}$$

$$= \max\{\frac{\text{algorithm work}}{\text{compute performance}}, \quad \frac{\text{data volume}}{\text{bandwidth}}\}$$

$$\text{performance} = \text{execution rate} = \frac{1}{\text{runtime}} = \min\{\frac{\text{compute performance}}{\text{algorithm work}}, \quad \frac{\text{bandwidth}}{\text{data volume}}\}$$

# Performance Limiters

*Assumption: Compute and Communicate can overlap*

$$\text{runtime} = \max\{T_{\text{compute}}, \quad T_{\text{communicate}}\}$$

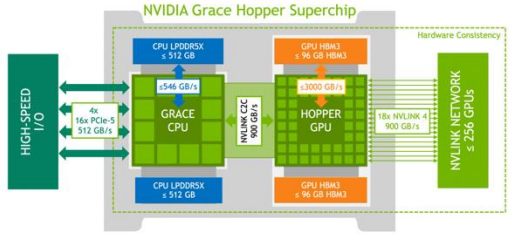
$$= \max\{\frac{\text{algorithm work}}{\text{compute performance}}, \quad \frac{\text{data volume}}{\text{bandwidth}}\}$$

$$\text{performance} = \text{execution rate} = \frac{1}{\text{runtime}} = \min\{\frac{\text{compute performance}}{\text{algorithm work}}, \quad \frac{\text{bandwidth}}{\text{data volume}}\}$$

$$= \min\{\underbrace{\text{compute performance}}_{\hat{P}}, \quad \underbrace{\text{bandwidth}}_{B} \cdot \underbrace{\frac{\text{algorithm work}}{\text{data volume}}}_{\text{arithmetic intensity (AI)}}\}$$

$$\text{performance} = \min\{\hat{P}, \quad B \cdot AI\}$$

# NVIDIA GH200




| | |
|---|---|
| FP64 | 34 teraFLOPS |
| FP64 Tensor Core | 67 teraFLOPS |
| FP32 | 67 teraFLOPS |
| TF32 Tensor Core | 989 teraFLOPS* \| 494 teraFLOPS |
| BFLOAT16 Tensor Core | 1,979 teraFLOPS* \| 990 teraFLOPS |
| FP16 Tensor Core | 1,979 teraFLOPS* \| 990 teraFLOPS |
| FP8 Tensor Core | 3,958 teraFLOPS* \| 1,979 teraFLOPS |
| INT8 Tensor Core | 3,958 TOPS* \| 1,979 TOPS |
| High-bandwidth memory (HBM) size | 96GB HBM3 \| 144GB HBM3e |
| Memory bandwidth | Up to 4TB/s \| Up to 4.9TB/s |
| NVIDIA NVLink-C2C CPU-to-GPU bandwidth | 900 GB/s |

67 teraFLOPS

34 teraFLOPS

Compute Performance [GFLOP/s]

$10^4$

$10^3$

Peak fp64 perf.

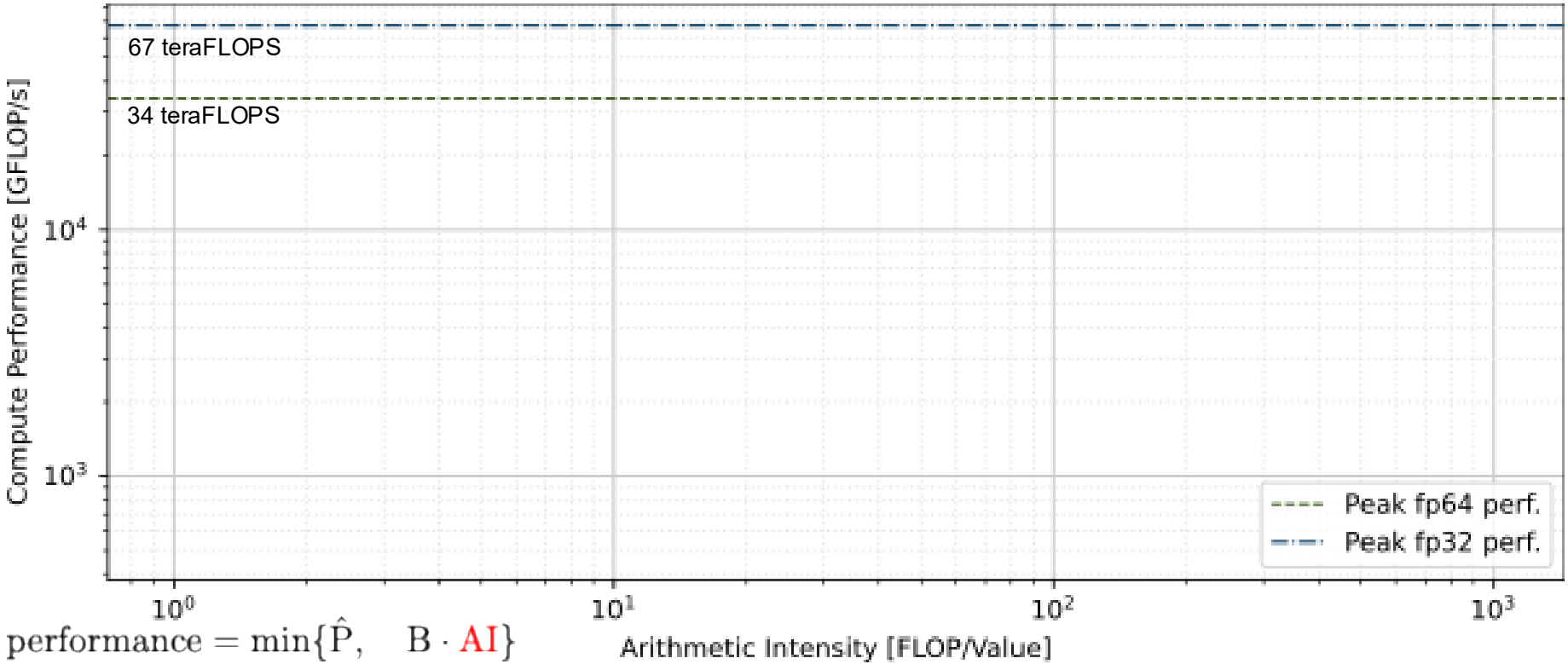
Peak fp32 perf.

$10^0$ $10^1$ $10^2$ $10^3$

$$\text{performance} = \min\{\hat{P}, \quad B \cdot AI\}$$

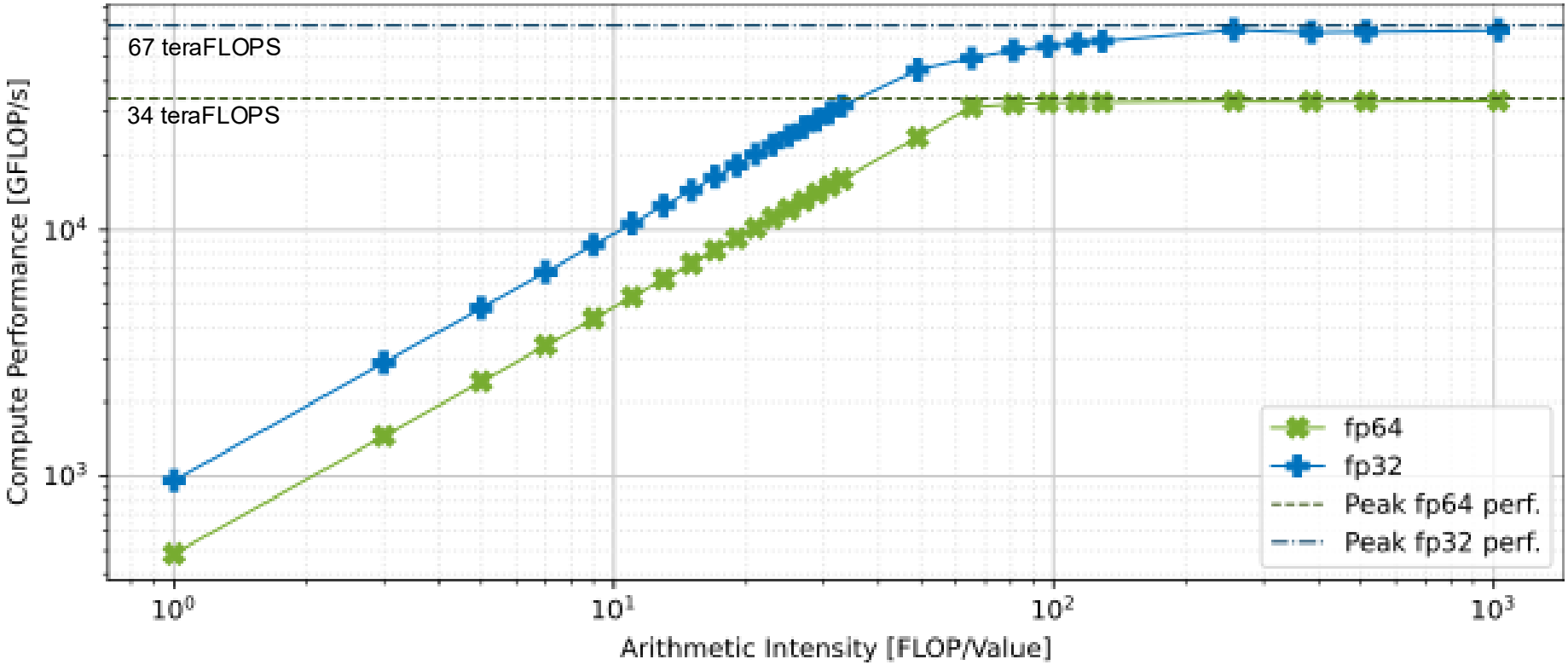
Arithmetic Intensity [FLOP/Value]

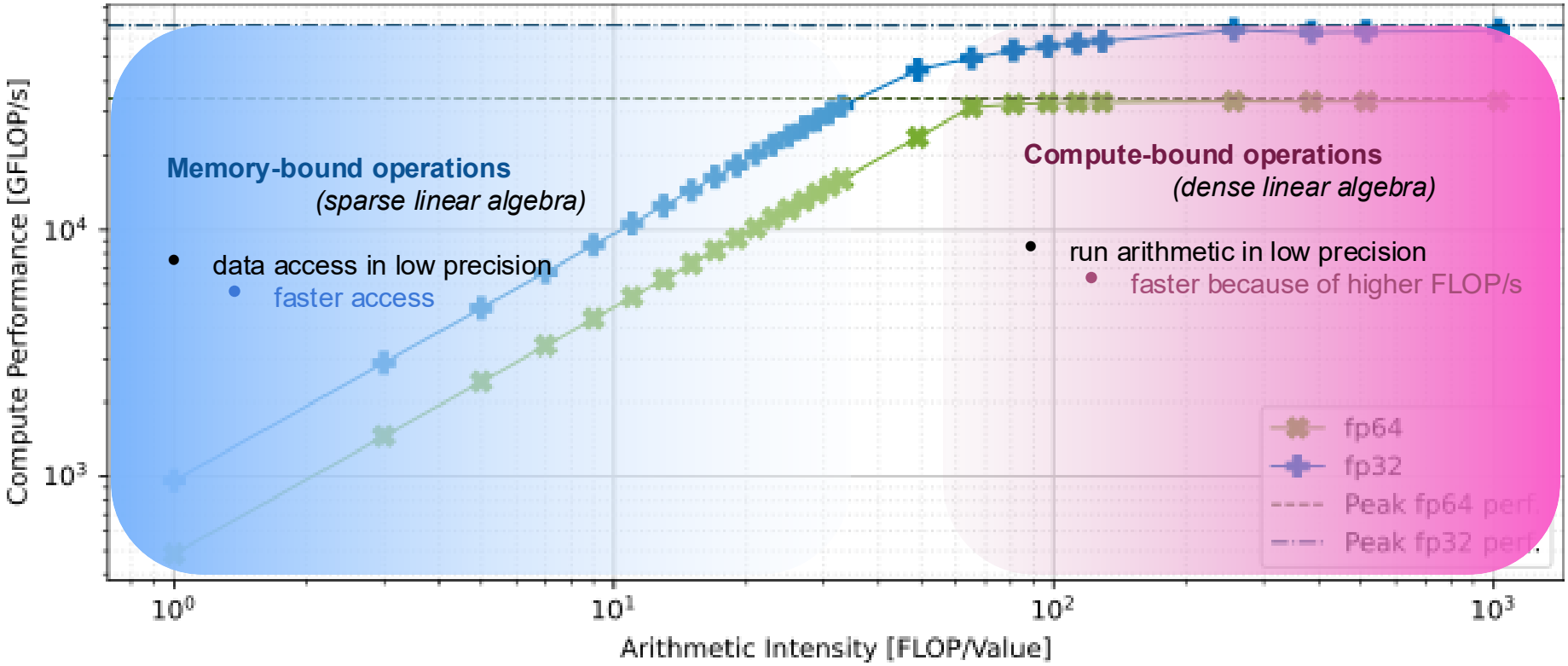
# NVIDIA GH200




$$\text{performance} = \min\{\hat{P}, \quad B \cdot AI\}$$

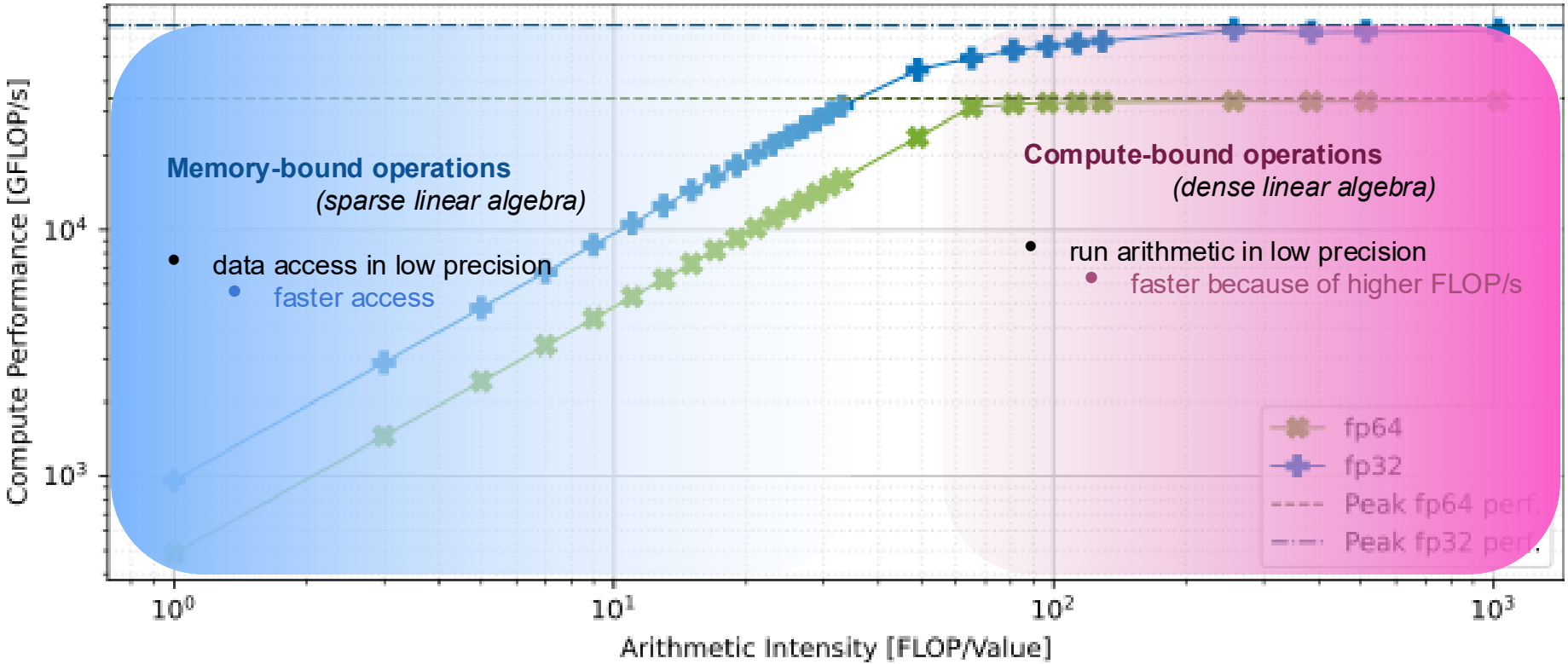
# NVIDIA GH200




Memory-bound operations
(sparse linear algebra)
data access in low precision
faster access
Compute-bound operations
(dense linear algebra)
run arithmetic in low precision
faster because of higher FLOP/s
fp64
fp32
Peak fp64 perf.
Peak fp32 perf.
Compute Performance [GFLOP/s]
$10^4$
$10^3$
$10^0$
$10^1$
$10^2$
$10^3$
Arithmetic Intensity [FLOP/Value]

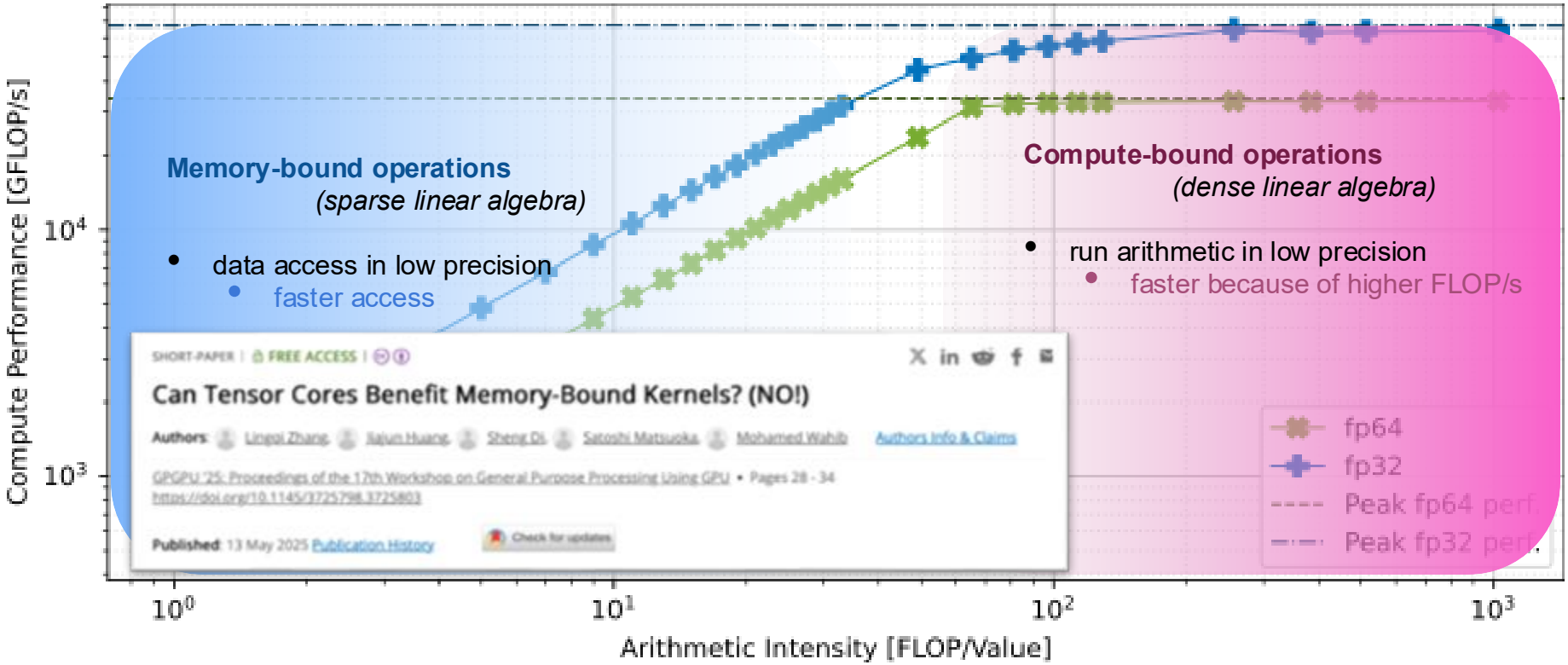
Matrix fp32

Matrix fp64

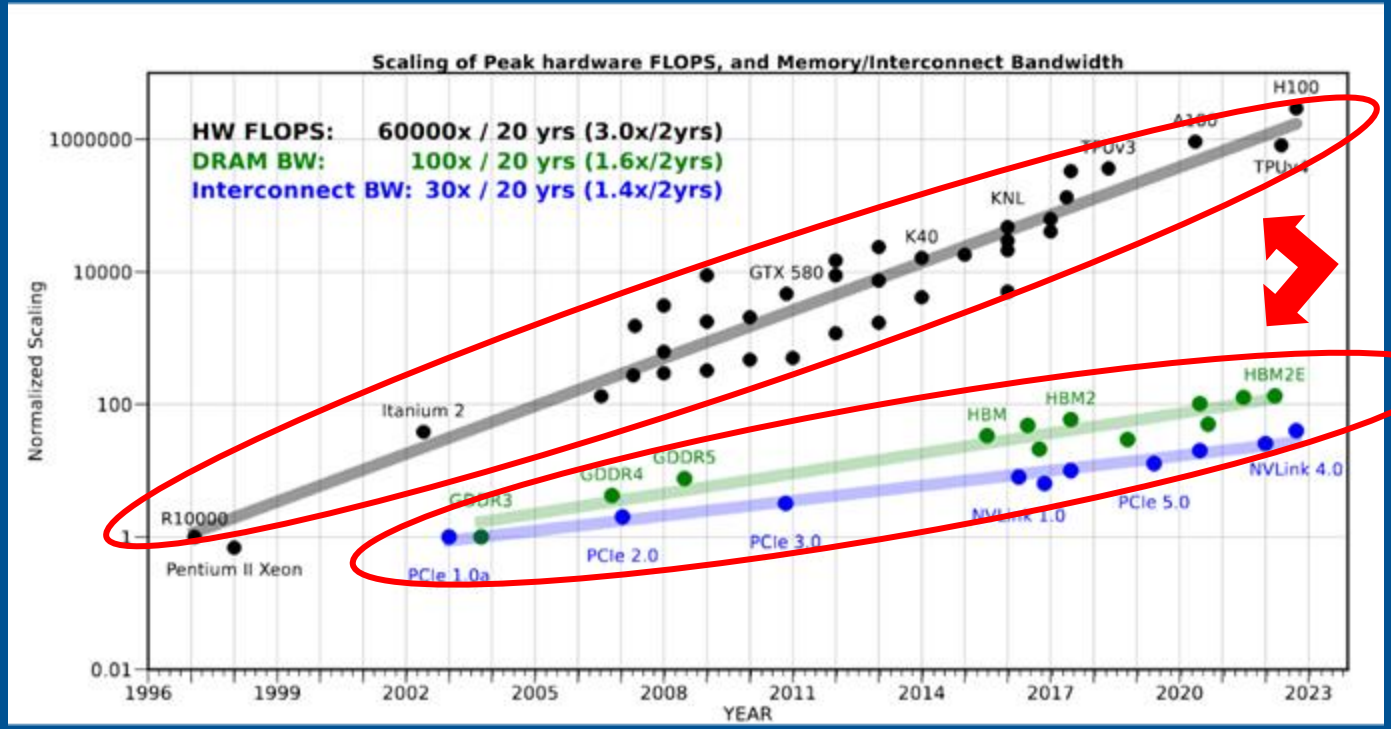


Matrix fp32

Matrix fp64



# Evolution of Performance and Memory Bandwidth



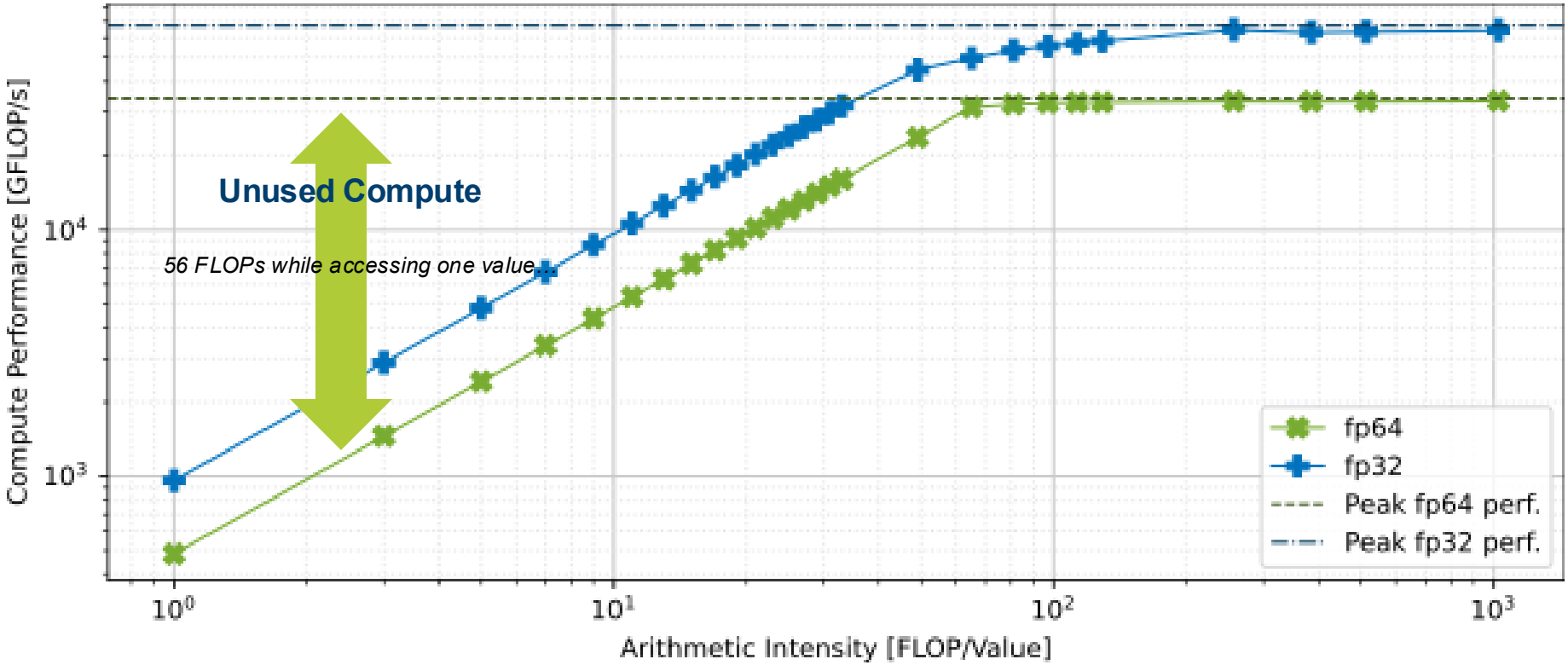

**Widening performance gap**

- **FLOPS: 3.0×/2yrs, Total: 60,000x**
- **DRAM bandwidth: 1.6x/2yrs, Total: 100x**
- **Interconnect bandwidth: 1.4x/2yrs, Total: 30x**

Jack Dongarra and David Keyes. "The co-evolution of computational physics and high-performance computing." *Nature Reviews Physics* (2024).

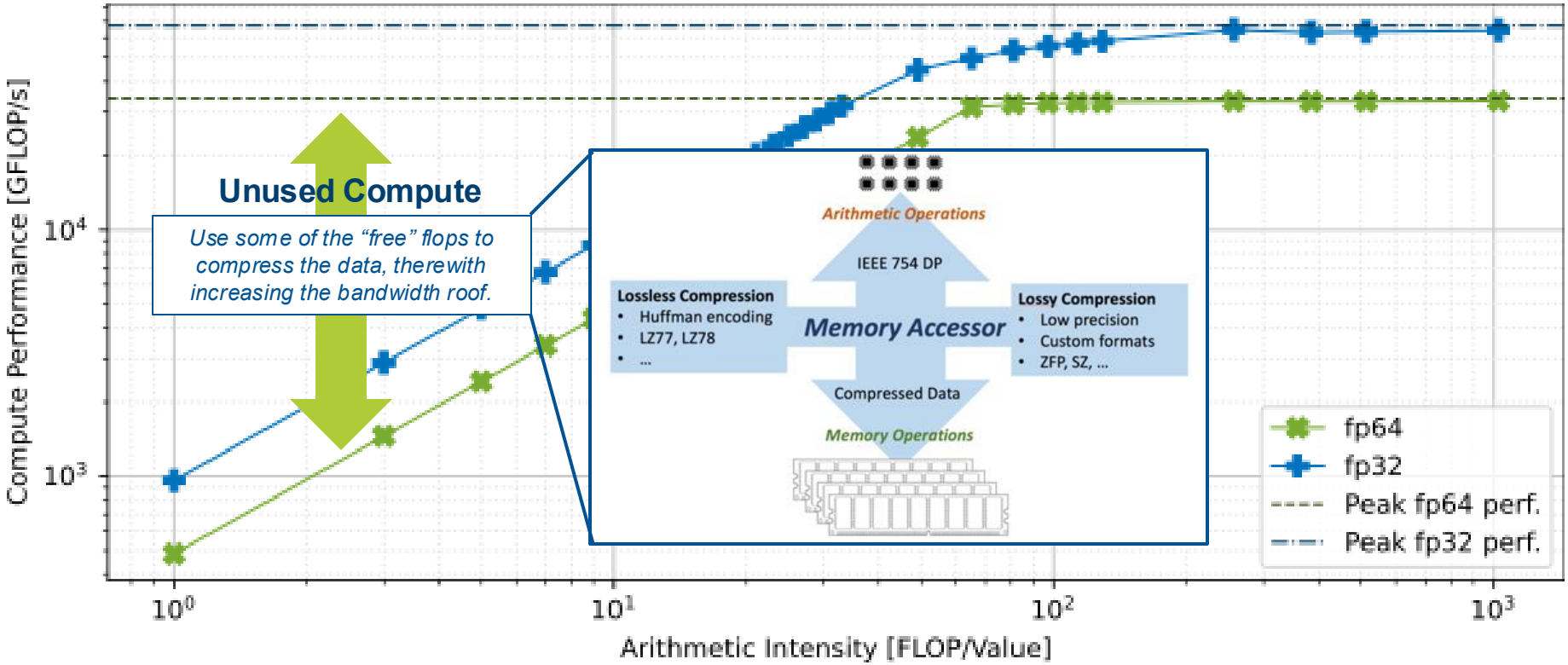
Matrix fp32

Matrix fp64

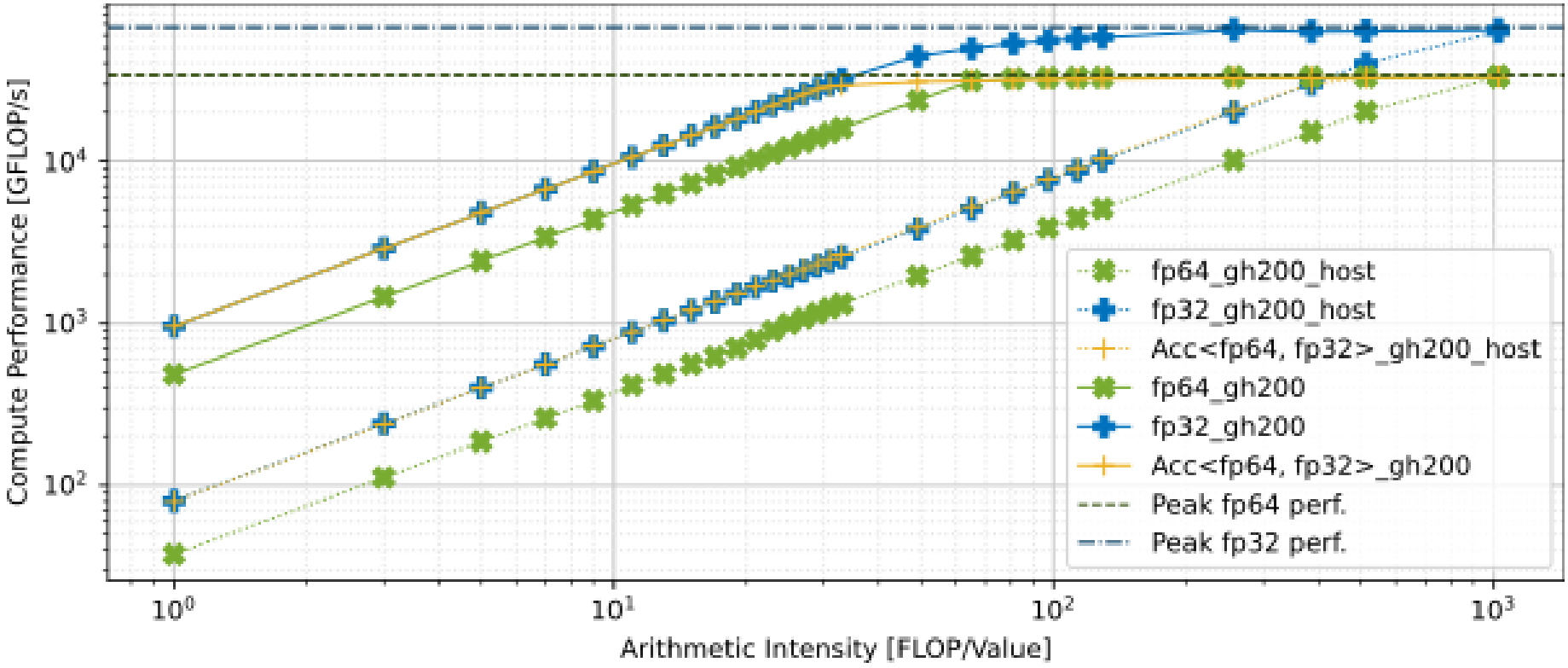


Matrix fp32

Matrix fp64



# A simple Compression in the Memory Accessor: Mixed Precision

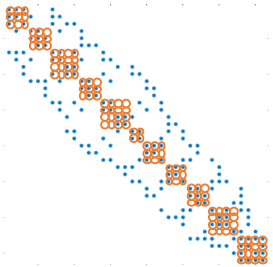


# Compressing Approximate Operators

- **Preconditioning iterative solvers**
  - Idea: Approximate inverse of system matrix to make the system “easier to solve”: $P^{-1} \approx A^{-1}$

    and solve $Ax = b \quad \Leftrightarrow \quad P^{-1}Ax = P^{-1}b \quad \Leftrightarrow \quad \tilde{A}x = \tilde{b}$ .

- **Block-Jacobi preconditioner** is based on **block-diagonal scaling**: $P = diag_B(A)$
  - Each block corresponds to one (small) linear system.
    - *Larger* blocks typically improve convergence.
    - *Larger* blocks make block-Jacobi more expensive.

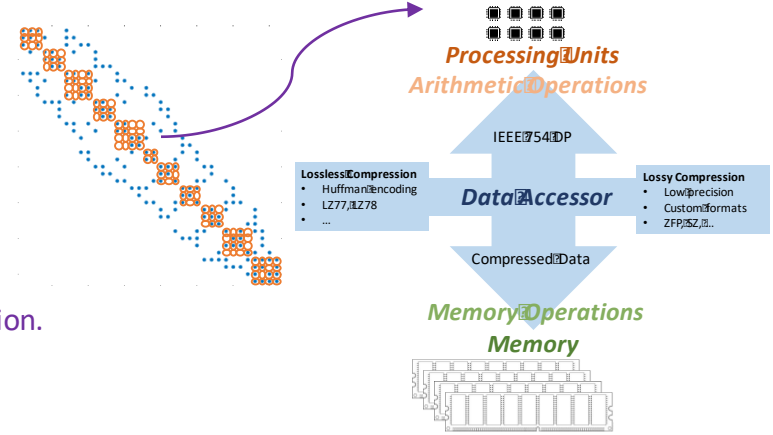


# Compressing Approximate Operators

- **Preconditioning iterative solvers**
  - Idea: Approximate inverse of system matrix to make the system "easier to solve": $P^{-1} \approx A^{-1}$

    and solve $Ax = b \quad \Leftrightarrow \quad P^{-1}Ax = P^{-1}b \quad \Leftrightarrow \quad \tilde{A}x = \tilde{b}$ .

- **Block-Jacobi preconditioner** is based on **block-diagonal scaling**: $P = diag_B(A)$
  - Each block corresponds to one (small) linear system.
    - *Larger* blocks typically improve convergence.
    - *Larger* blocks make block-Jacobi more expensive.

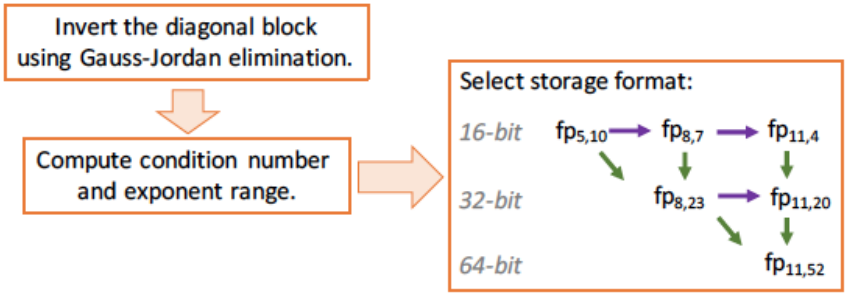



- *Can we compress the preconditioner matrix $P^{-1}$ ?*

- Use the memory accessor to store the inverted diagonal blocks in lower precision.
  - *Be careful to preserve the regularity of each inverted diagonal block!*

# Approximate Operators using the Memory Accessor

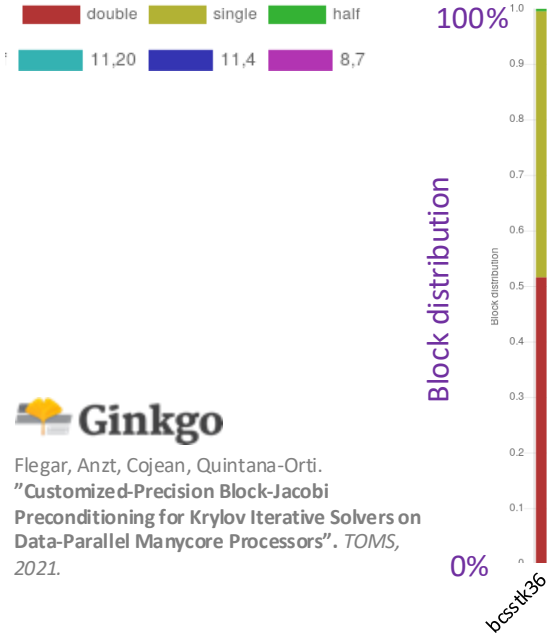
- Choose how much accuracy of the preconditioner should be preserved in the selection of the storage format.
- All computations use double precision, but store blocks in lower precision.




+ **Regularity preserved;**
+ Flexibility in the accuracy;
+ ”Not a low precision preconditioner”
  + Preconditioner is a constant operator;
  + No flexible Krylov solver needed ;

- **Overhead** of the **precision detection** (condition number calculation);
- **Overhead** from storing **precision information** (need to additionally store/retrieve flag);
- Speedups / preconditioner quality **problem-dependent**;

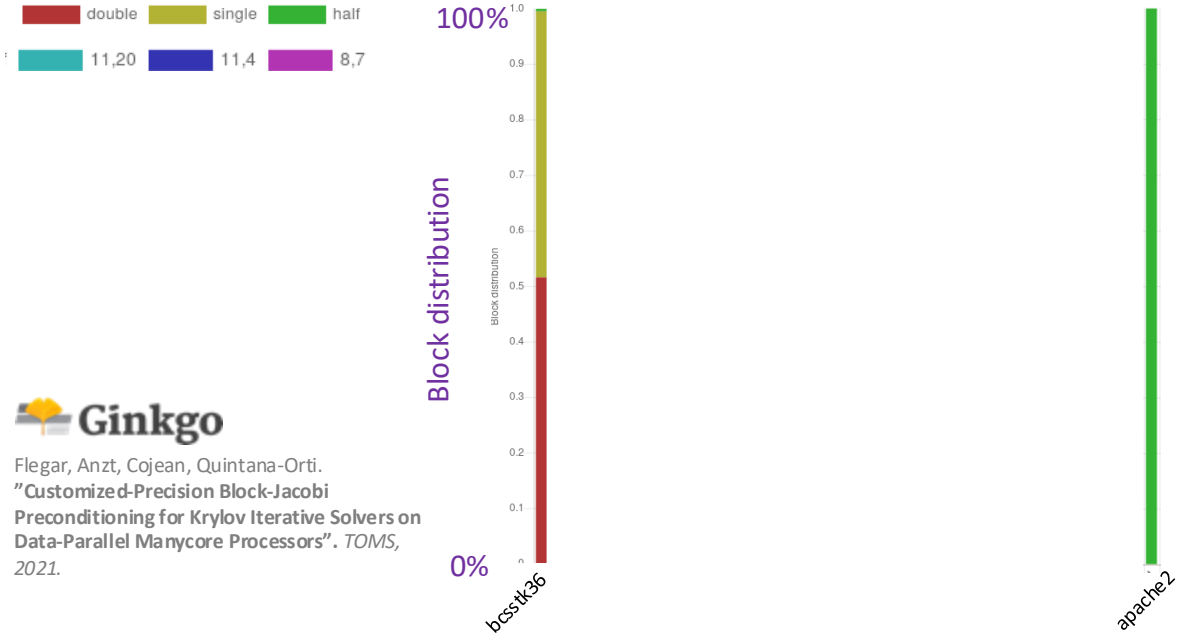
# Performance




Ginkgo

Flegar, Anzt, Cojean, Quintana-Orti.
**"Customized-Precision Block-Jacobi Preconditioning for Krylov Iterative Solvers on Data-Parallel Manycore Processors".** *TOMS, 2021.*

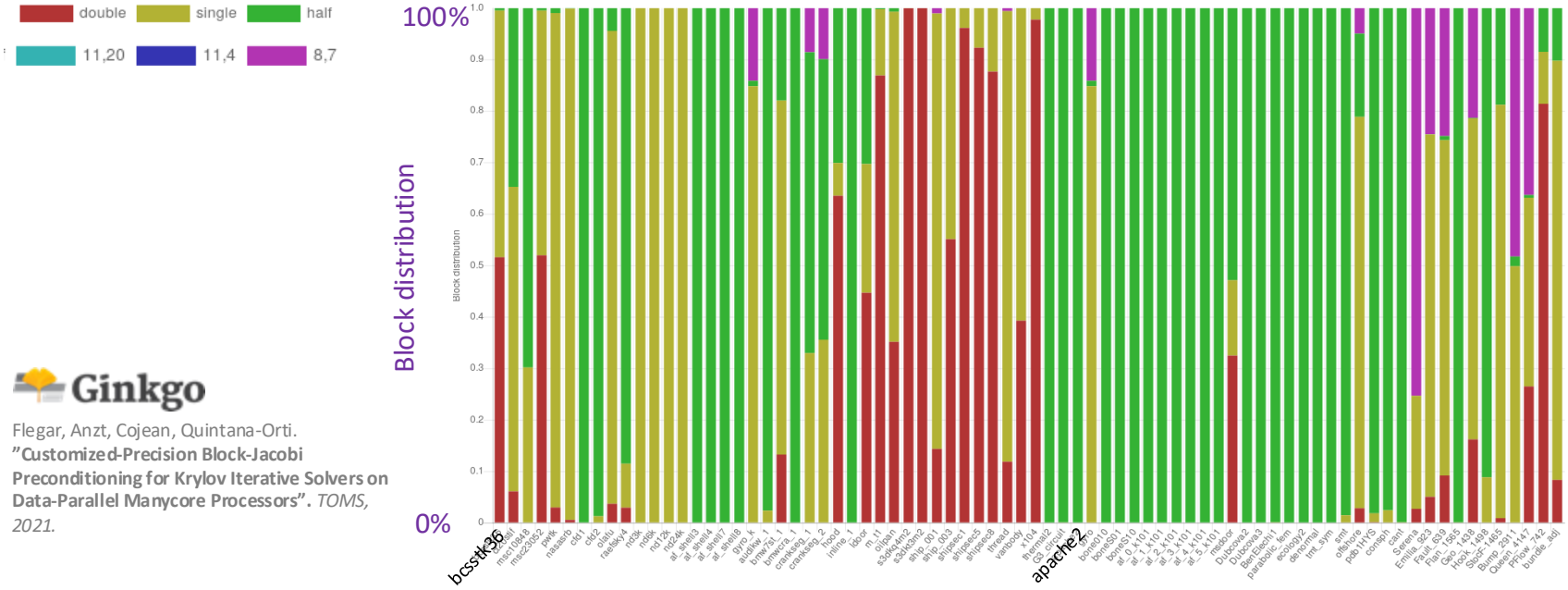
# Performance




Ginkgo

Flegar, Anzt, Cojean, Quintana-Orti.
**"Customized-Precision Block-Jacobi Preconditioning for Krylov Iterative Solvers on Data-Parallel Manycore Processors".** *TOMS, 2021.*

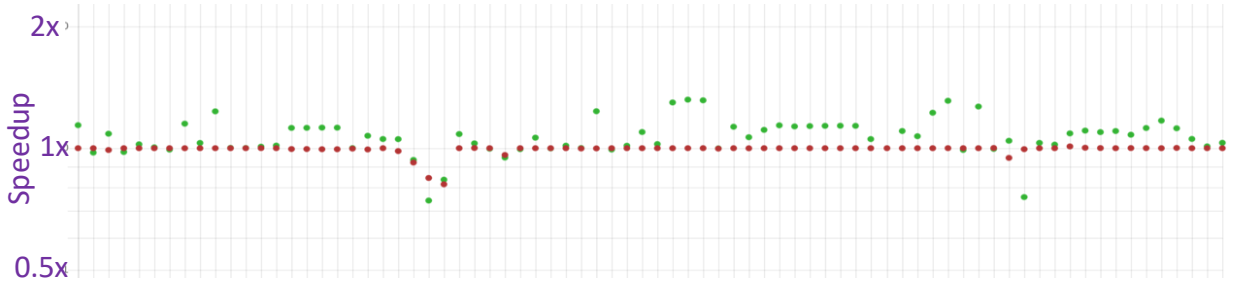
# Performance



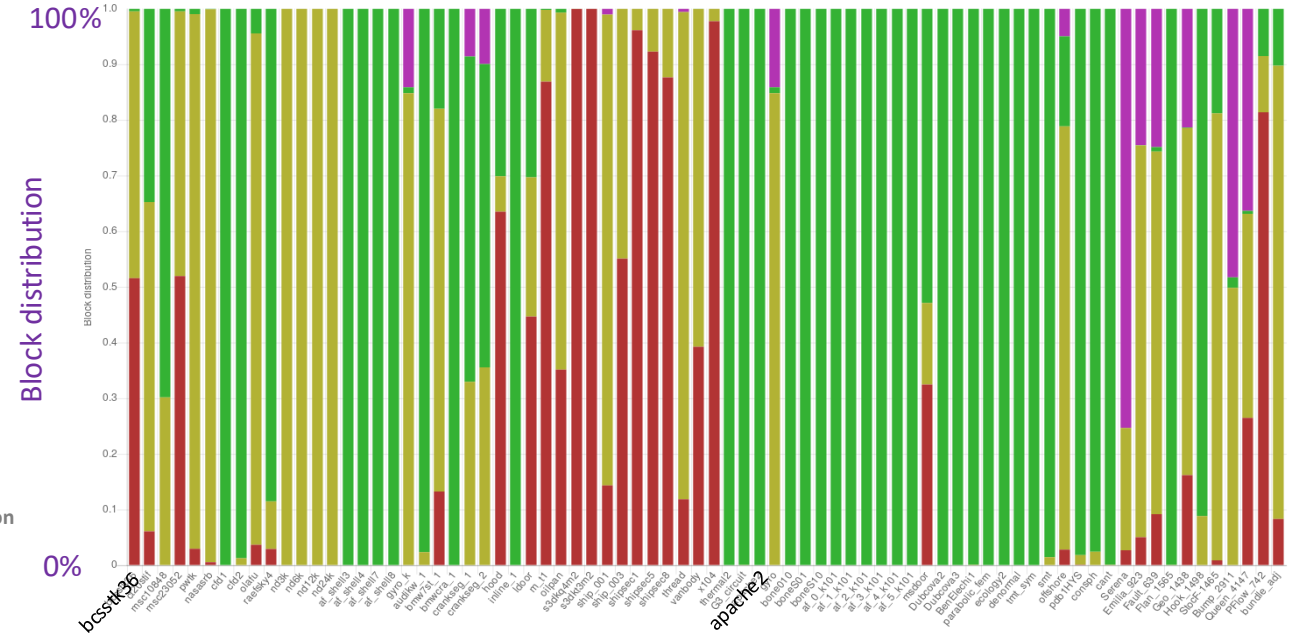

Ginkgo

Flegar, Anzt, Cojean, Quintana-Orti.
**"Customized-Precision Block-Jacobi Preconditioning for Krylov Iterative Solvers on Data-Parallel Manycore Processors".** *TOMS, 2021.*

# Performance








Ginkgo

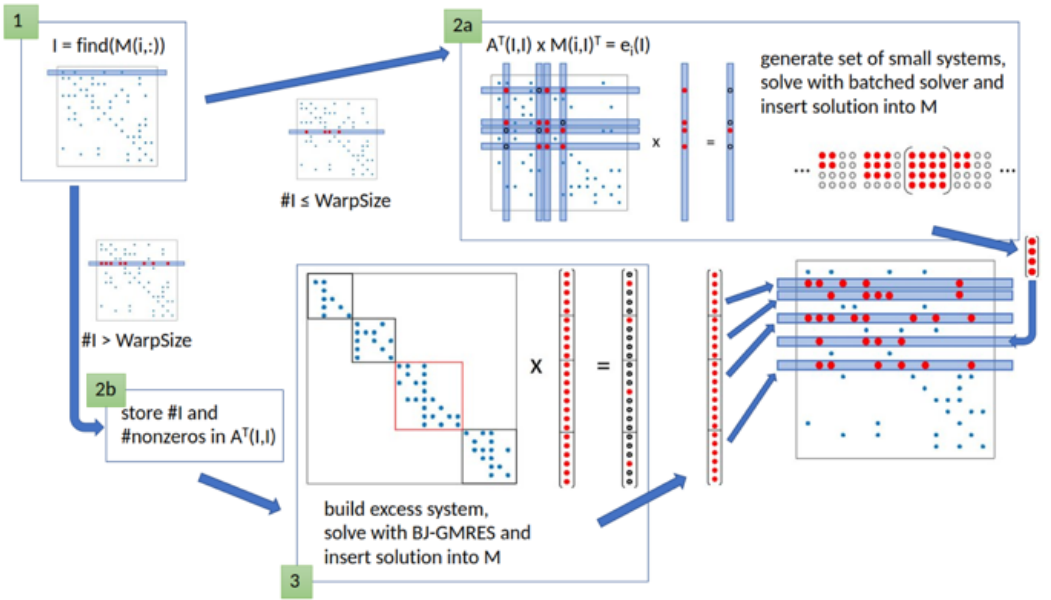
Flegar, Anzt, Cojean, Quintana-Orti.
**"Customized-Precision Block-Jacobi Preconditioning for Krylov Iterative Solvers on Data-Parallel Manycore Processors".** *TOMS, 2021.*

# Approximate Operators using the Memory Accessor

- **Preconditioning iterative solvers**
  - Idea: Approximate inverse of system matrix to make the system “easier to solve”: $P^{-1} \approx A^{-1}$

    and solve $Ax = b \quad \Leftrightarrow \quad P^{-1}Ax = P^{-1}b \quad \Leftrightarrow \quad \tilde{A}x = \tilde{b}$ .

- **Incomplete Sparse Approximate Inverse Preconditioner**
  - $M \approx A^{-1}$ with $(M \cdot A)|_A = I$
  - uses sparsity pattern of $A$ ;

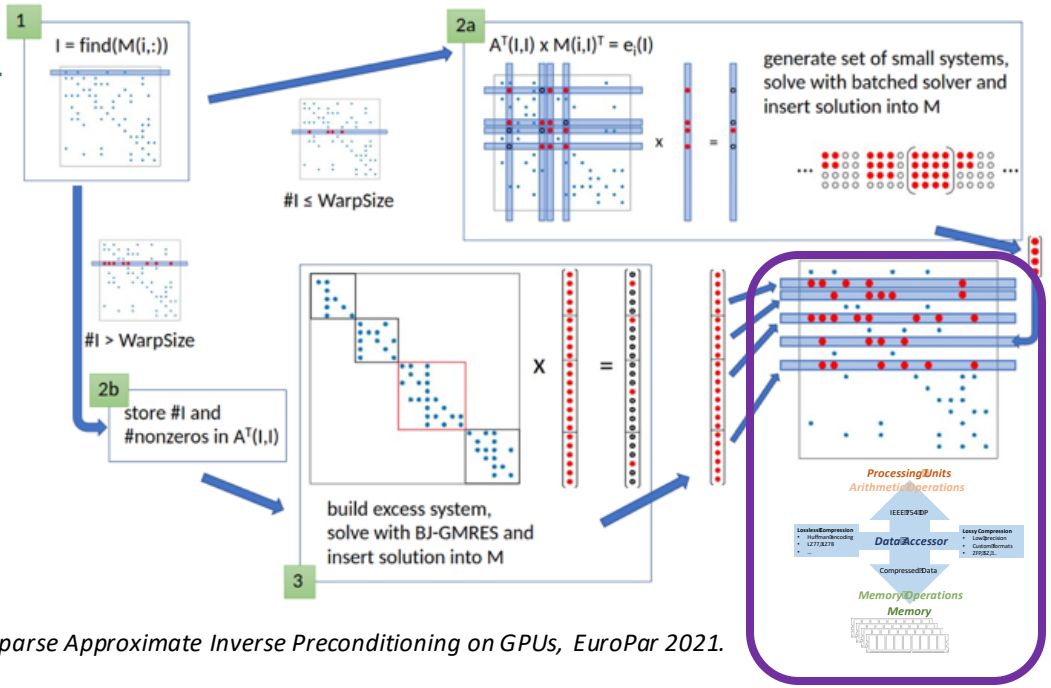


# Approximate Operators using the Memory Accessor

- **Preconditioning iterative solvers**
  - Idea: Approximate inverse of system matrix to make the system "easier to solve": $P^{-1} \approx A^{-1}$

    and solve $Ax = b \quad \Leftrightarrow \quad P^{-1}Ax = P^{-1}b \quad \Leftrightarrow \quad \tilde{A}x = \tilde{b}$ .

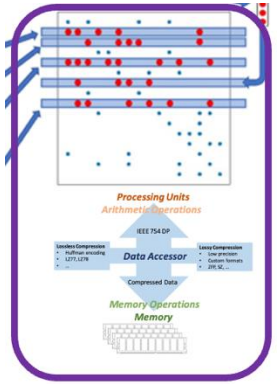
- **Incomplete Sparse Approximate Inverse Preconditioner**
  - $M \approx A^{-1}$ with $(M \cdot A)|_A = I$
  - uses sparsity pattern of $A$ ;




*Göbel at al: Mixed Precision Incomplete and Factorized Sparse Approximate Inverse Preconditioning on GPUs, EuroPar 2021.*

# Memory Accessor Research

***How do we preserve numerical stability?***

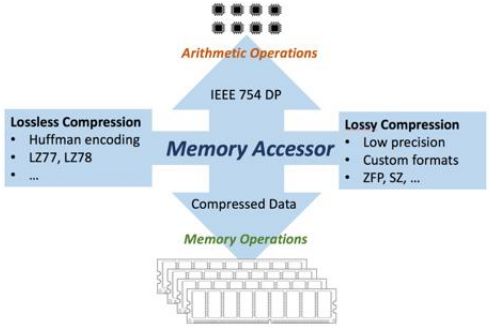



- Preserve regularity of preconditioner
- Avoid over/underflow

BUT:

- Global preconditioners are costly to analyze..
- Ideas?
- Quick estimate of eigenvalues / condition number?

***How can we achieve higher compression rates?***




- Compression often use pattern recognition/interpolation
- Larger blocks of data allow for higher compression rates

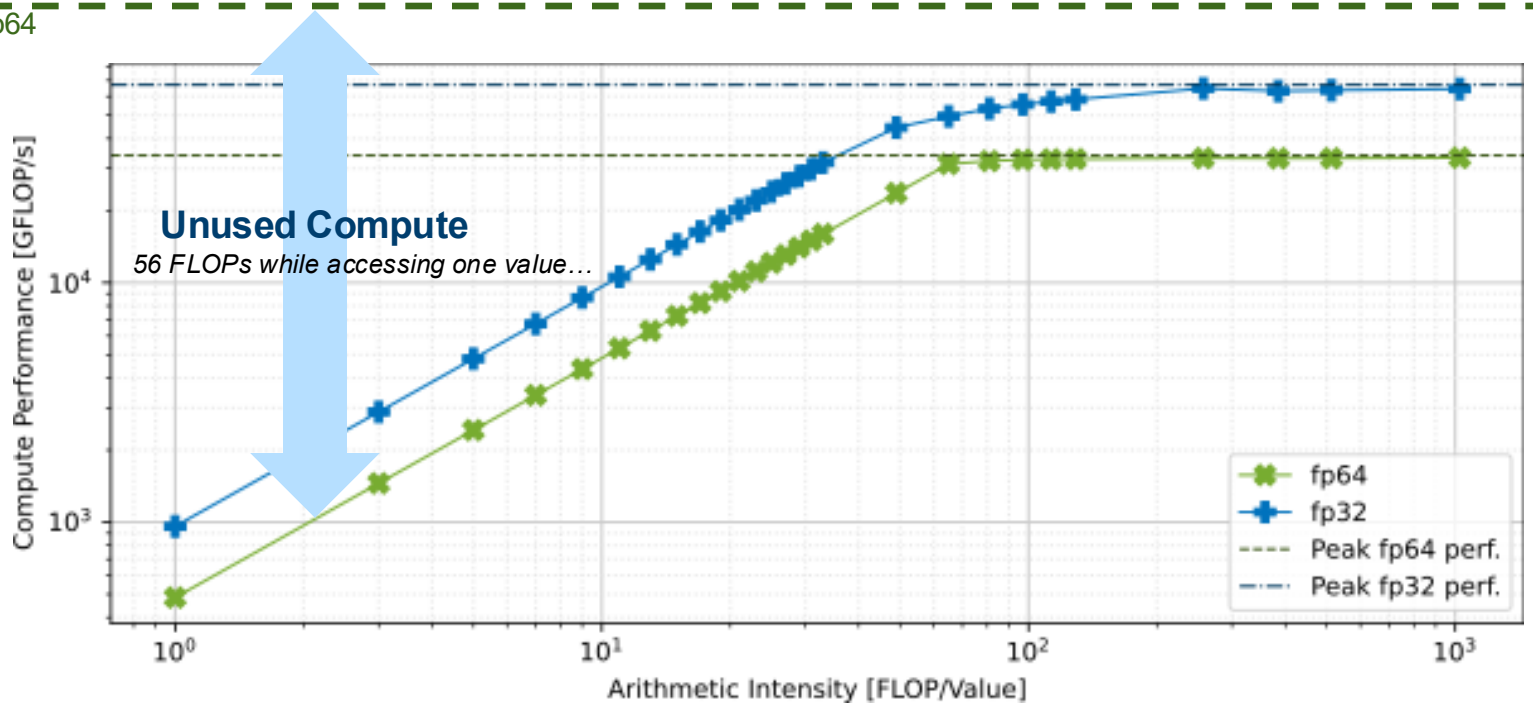
BUT:

- Memory accessor can only operate in registers
- Often, algorithms need Random Memory Access (RMA)

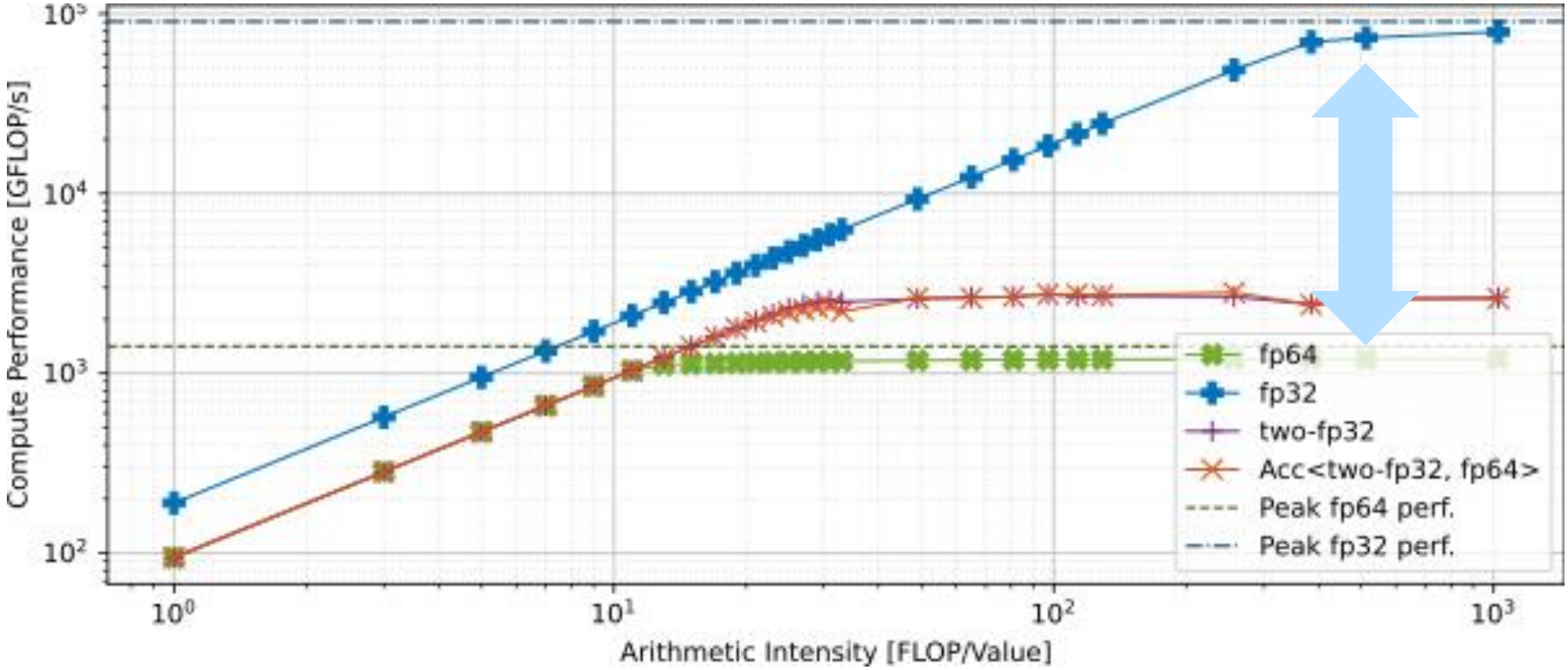
# Activities going forward: There's plenty of room at the Top

Matrix fp32

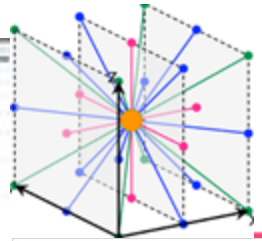
Matrix fp64



# Activities going forward: There's plenty of room at the Top



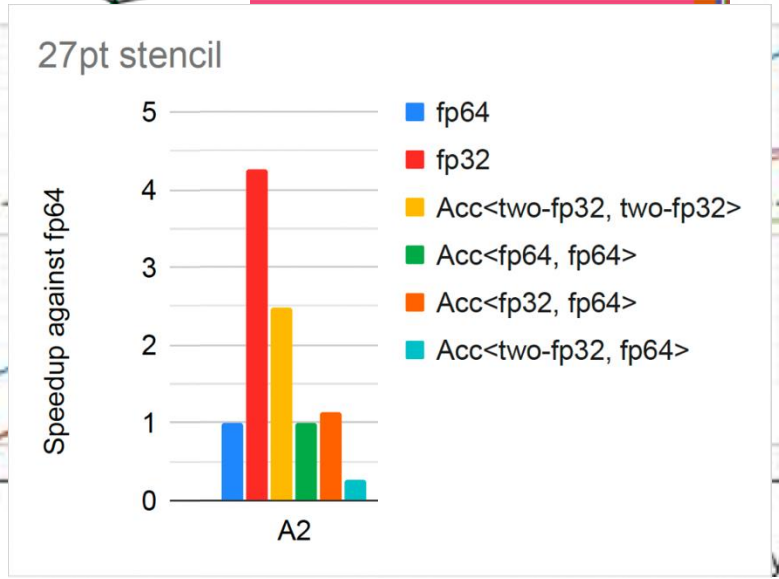
# Activities going forward: There's plenty of room at the Top



NCAS-CMS/
**hpcg_27ptStencil**

Matrix free version of HPCG benchmark source code

0 Contributors 0 Issues 0 Stars 0 Forks

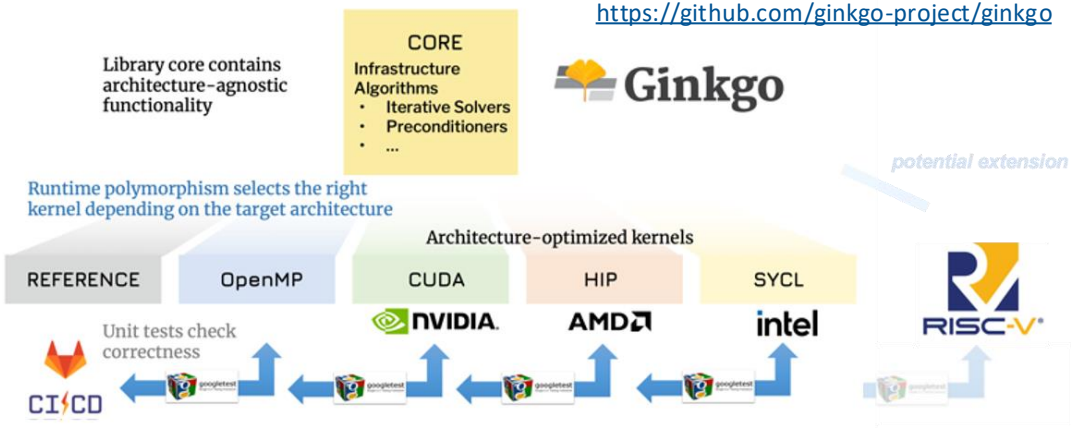


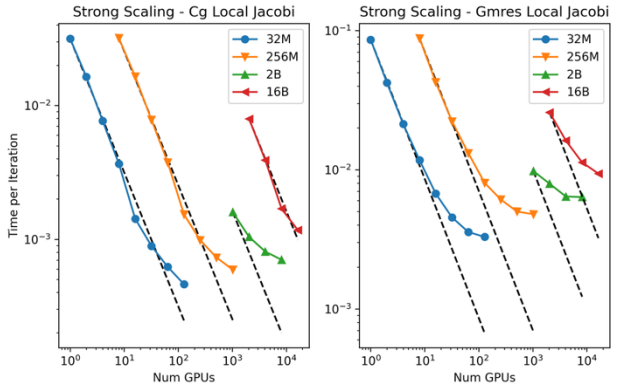
# What will the future bring?

- **AI is dominating the hardware market.**
  - Matrix multiplication in low precision.
  - **Low precision** will become the **major format supported by hardware** (16 bit? 8bit? 4bit?).
  - We will likely **emulate higher precision**.
    (Ozaki scheme etc. – tailored for GEMM)
- **Bandwidth and Latency can not keep up with growth in compute power.**
  - **We need use compression on all levels** – likely hierarchically combined.
  - **New algorithms** need to **favor computation over communication**
    (including repeated computation over memory access).
- **Power consumption.**
  - **Data centers as major consumers can compensate variations in the power generation.**
  - **Scientific computing will become a malleable load.**

# Ginkgo High Performance Math Library

https://github.com/ginkgo-project/ginkgo




Strong scaling of Ginkgo for 3D Laplace problem (27pt stencil) on ORNL's Frontier system (AMD MI200 GPUs).

| | | OMP | CUDA | HIP | SYCL |
|---|---|---|---|---|---|
| Basic | SpMV | ✓ | ✓ | ✓ | ✓ |
| | SpMM | ✓ | ✓ | ✓ | ✓ |
| | SpGeMM | ✓ | ✓ | ✓ | ✓ |
| Krylov solvers | BiCG | ✓ | ✓ | ✓ | ✓ |
| | BiCGSTAB | ✓ | ✓ | ✓ | ✓ |
| | CG | ✓ | ✓ | ✓ | ✓ |
| | CGS | ✓ | ✓ | ✓ | ✓ |
| | GCR | ✓ | ✓ | ✓ | ✓ |
| | GMRES | ✓ | ✓ | ✓ | ✓ |
| | FCG | ✓ | ✓ | ✓ | ✓ |
| | FGMRES | ✓ | ✓ | ✓ | ✓ |
| | IR | ✓ | ✓ | ✓ | ✓ |
| | IDR | ✓ | ✓ | ✓ | ✓ |
| | Chebyshev | ✓ | ✓ | ✓ | ✓ |
| | MinRES | ✓ | ✓ | ✓ | ✓ |
| Preconditioners | Block-Jacobi | ✓ | ✓ | ✓ | ✓ |
| | Gauss-Seidel/SSOR | ✓ | ✓ | ✓ | ✓ |
| | ILU/IC | ✓ | ✓ | ✓ | |
| | Parallel ILU/IC | ✓ | ✓ | ✓ | ✓ |
| | Parallel ILUT/ICT | ✓ | ✓ | ✓ | ✓ |
| | ISAI | ✓ | ✓ | ✓ | ✓ |
| Batched | Batched BiCGSTAB | ✓ | ✓ | ✓ | ✓ |
| | Batched CG | ✓ | ✓ | ✓ | ✓ |
| | Batched GMRES | ✓ | ✓ | ✓ | ✓ |
| | Batched ILU | ✓ | ✓ | ✓ | ✓ |
| | Batched ISAI | ✓ | ✓ | ✓ | ✓ |
| | Batched Block-Jacobi | ✓ | ✓ | ✓ | ✓ |
| AMG | AMG preconditioner | ✓ | ✓ | ✓ | ✓ |
| | AMG solver | ✓ | ✓ | ✓ | ✓ |
| | Parallel Graph Match | ✓ | ✓ | ✓ | ✓ |
| Sparse direct | Symbolic Cholesky | ✓ | ✓ | ✓ | ✓ |
| | Numeric Cholesky | ✓ | ✓ | ✓ | |
| | Symbolic LU | ✓ | ✓ | ✓ | ✓ |
| | Numeric LU | ✓ | ✓ | ✓ | |
| | Sparse TRSV | ✓ | ✓ | ✓ | |
| Utilities | On-Device Matrix Assembly | ✓ | ✓ | ✓ | ✓ |
| | MC64/RCM reordering | ✓ | ✓ | ✓ | ✓ |
| | Wrapping user data | ✓ | ✓ | ✓ | ✓ |
| | Logging | ✓ | ✓ | ✓ | ✓ |
| | PAPI counters (SDE) | ✓ | ✓ | ✓ | ✓ |
| | Config file support | ✓ | ✓ | ✓ | ✓ |

✓ MPI Support ✓ Single-GPU Support