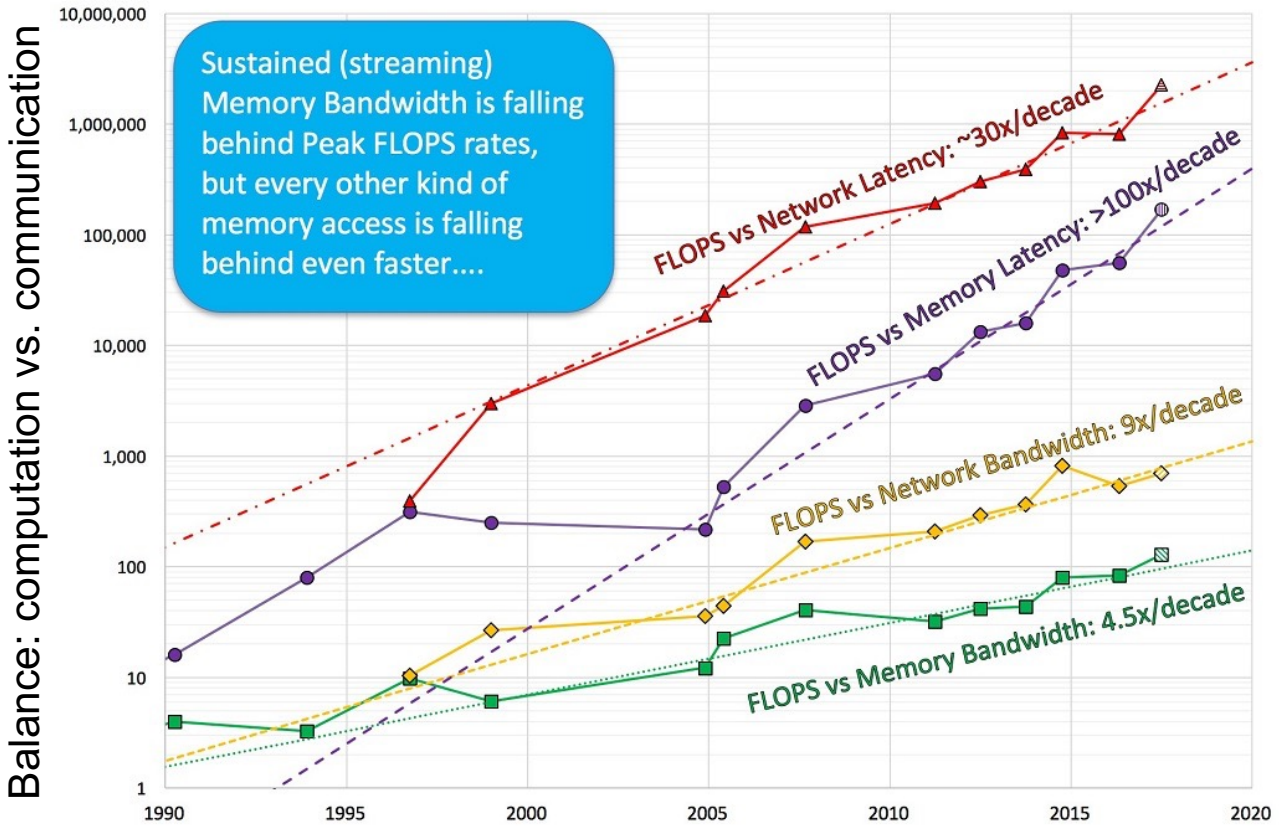
Can Mixed Precision Accelerate Sparse Solvers?

Hartwig Anzt
Technical University of Munich
University of Tennessee

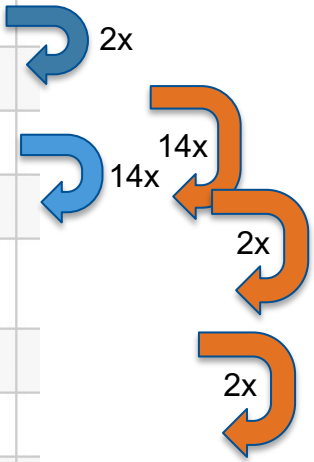
This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.



Trends in the relative performance of floating-point arithmetic and several classes of data access for select HPC servers over the past 25 years. Source: John McCalpin

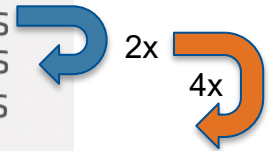
NVIDIA

Form Factor	H100 SXM
FP64	34 teraFLOPS
FP64 Tensor Core	67 teraFLOPS
FP32	67 teraFLOPS
TF32 Tensor Core	989 teraFLOPS[2]
BFLOAT16 Tensor Core	1,979 teraFLOPS[2]
FP16 Tensor Core	1,979 teraFLOPS[2]
FP8 Tensor Core	3,958 teraFLOPS[2]
INT8 Tensor Core	3,958 TOPS[2]
GPU memory	80GB
GPU memory bandwidth	3.35TB/s

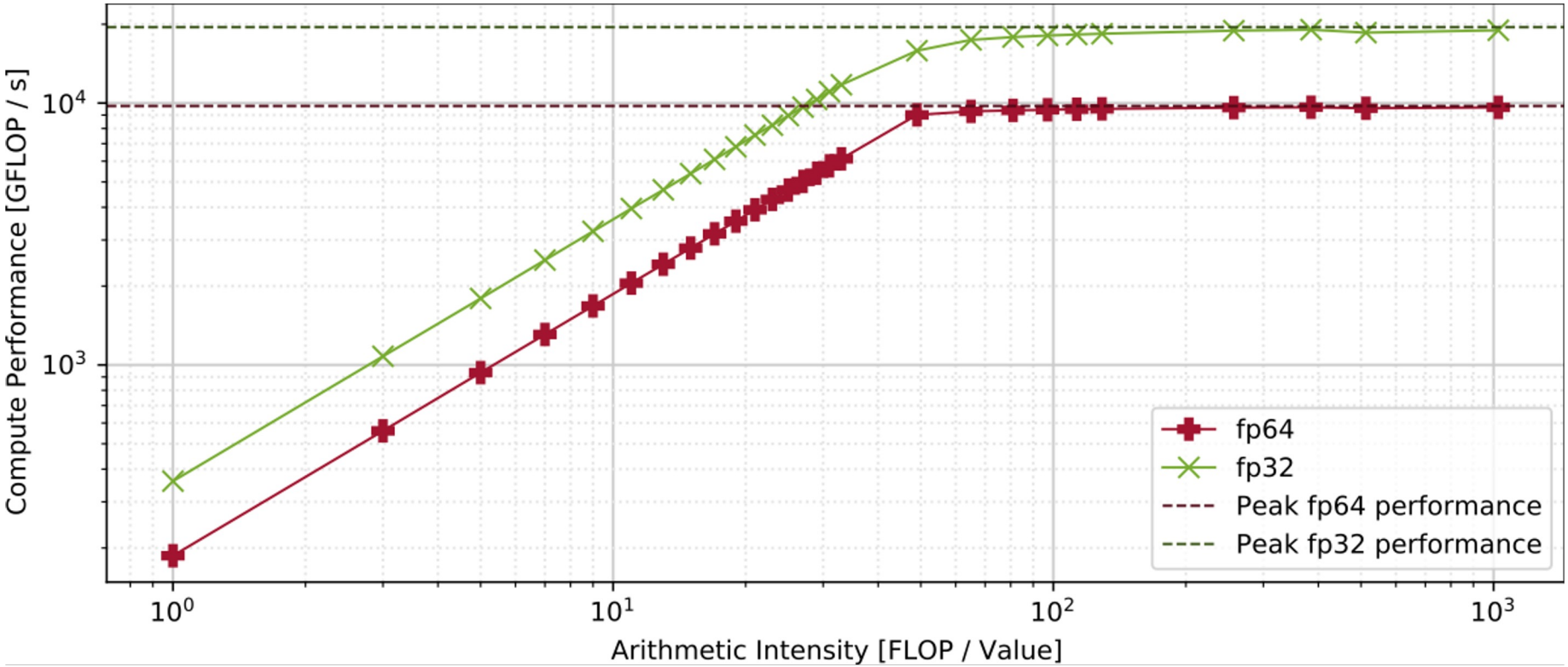


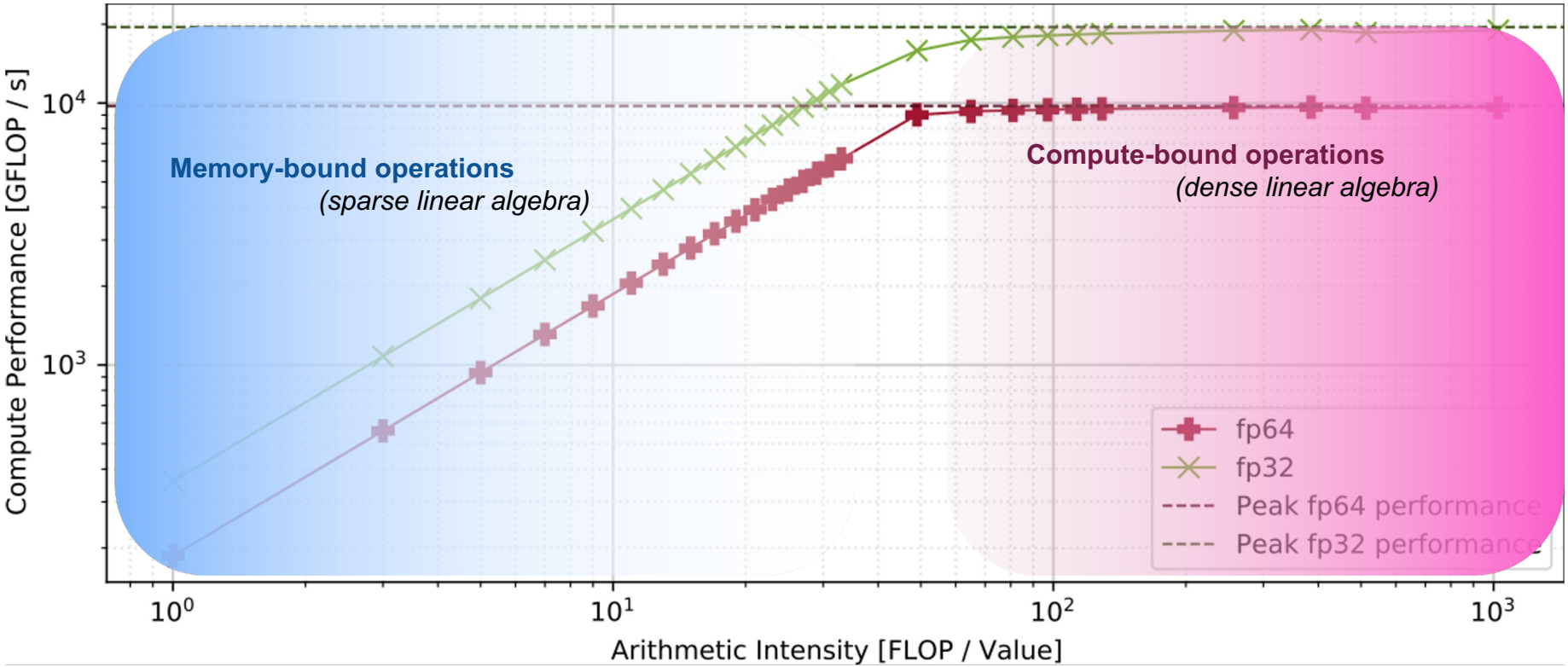
AMD

PERFORMANCE	MI250
Compute Units	208CU
Stream Processors	13,312
Peak FP64/FP32 Vector	45.3 TFLOPS
Peak FP64/FP32 Matrix	90.5 TFLOPS
Peak FP16/BF16	362.1 TFLOPS
Peak INT4/INT8	362.1 TOPS
MEMORY	
Memory Size	128GB HBM2e
Memory Interface	8,192 bits
Memory Clock	1.6GHz
Memory Bandwidth	up to 3.2TB/sec[2]



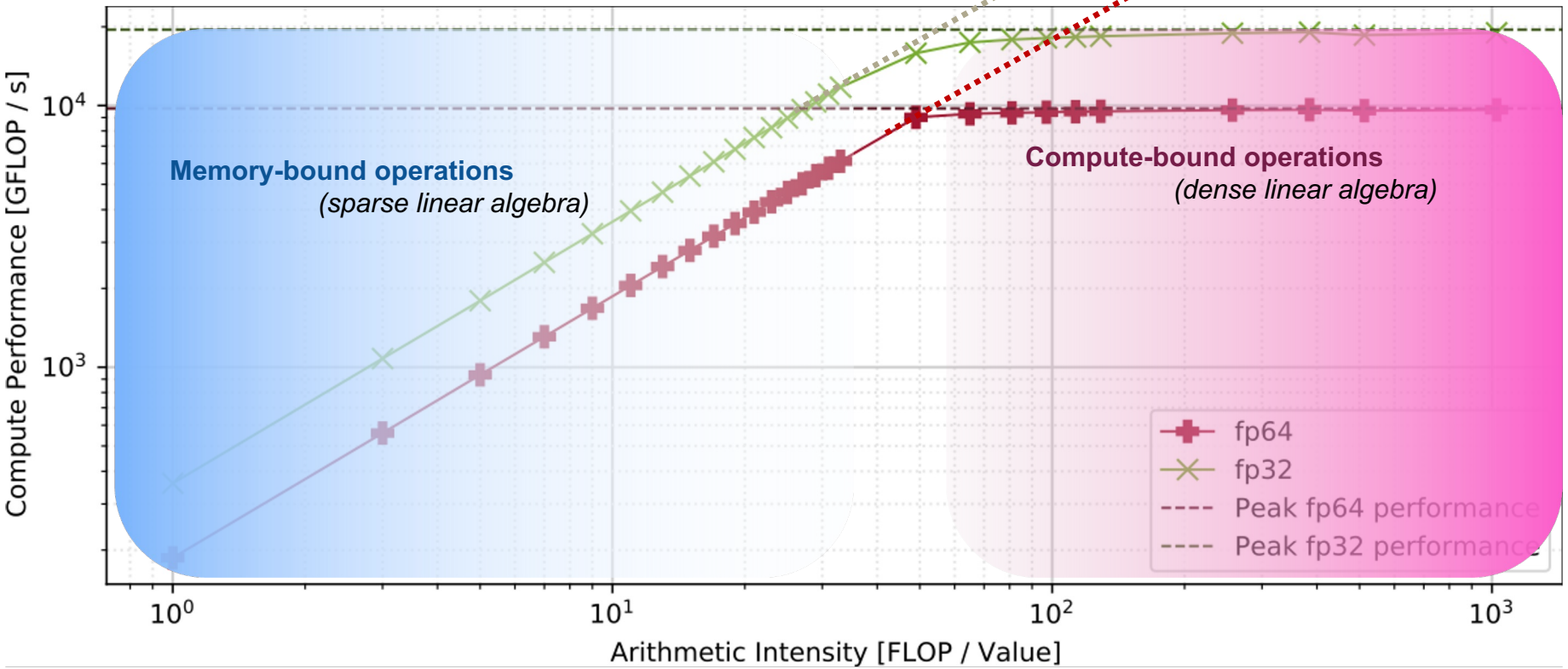
- (Dense) Matrix Performance >> Vector Operation Performance
- Low Precision Performance >> High Precision Performance


Compute Performance [GFLOP / s]
10^4
10^3
10^0
10^1
10^2
10^3
Arithmetic Intensity [FLOP / Value]
fp64
fp32
Peak fp64 performance
Peak fp32 performance


Memory-bound operations
(sparse linear algebra)
Compute-bound operations
(dense linear algebra)
fp64
fp32
Peak fp64 performance
Peak fp32 performance
Compute Performance [GFLOP / s]
Arithmetic Intensity [FLOP / Value]
10^4
10^3
10^0
10^1
10^2
10^3

Matrix fp32

Matrix fp64



Linear System Ax=b with cond(A) ≈ 10^7

(apache2 from SuiteSparse) **NVIDIA V100 GPU**

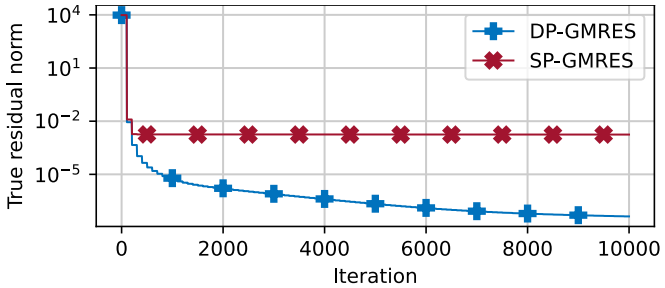
```
Double precision GMRES
Initial residual norm
sqrt(r^t r): 9670.36
Final residual norm
sqrt(r^T r): 9.6639e-09
GMRES iteration count: 23271
GMRES execution time: 43801 ms
```

Relative residual ~10^{-12}

```
Single precision GMRES
Initial residual norm
sqrt(r^t r): 9670.36
Final residual norm
sqrt(r^T r): 0.00175464
GMRES iteration count: 25000
GMRES execution time: 27376 ms
```

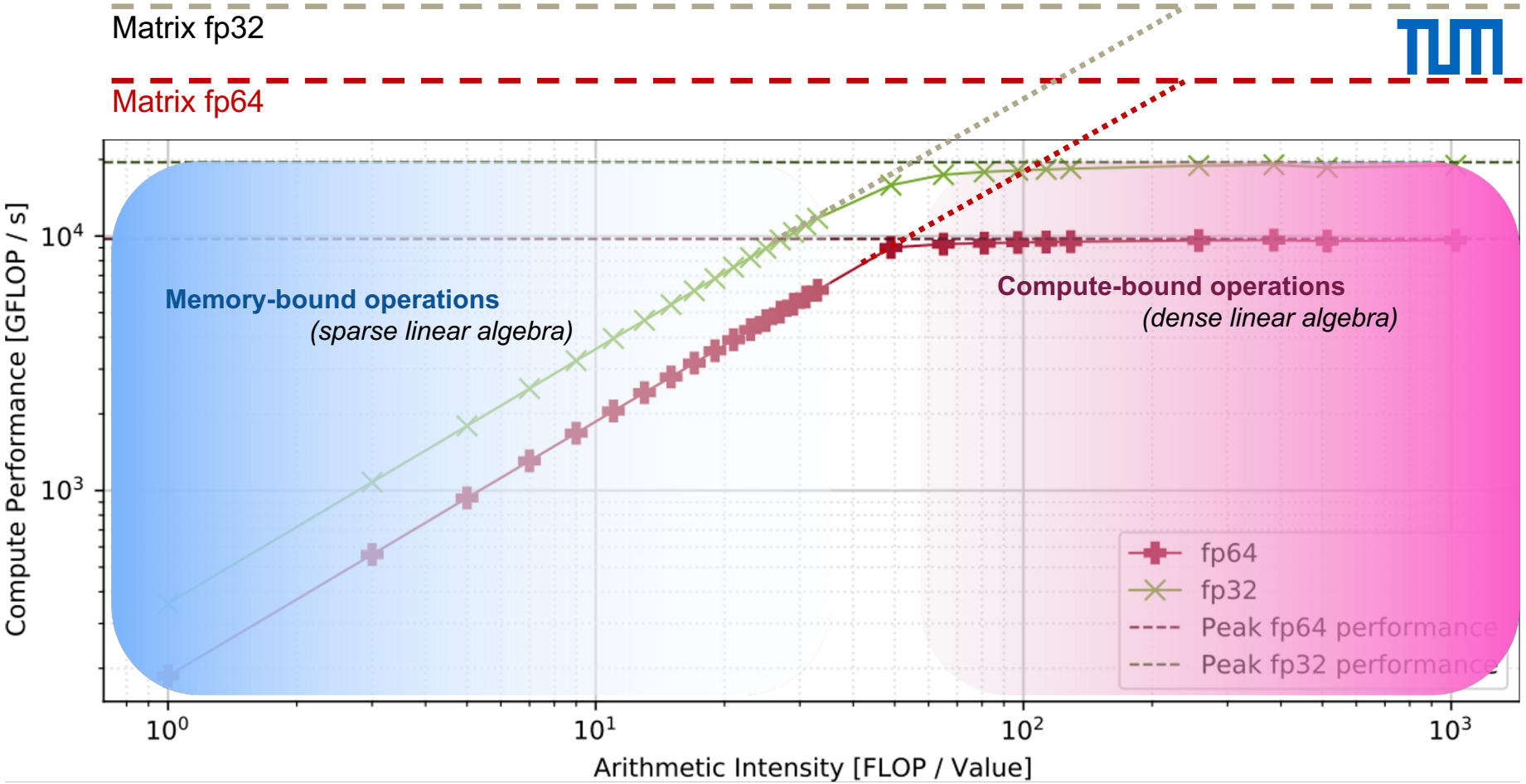
Relative residual ~10^{-7}

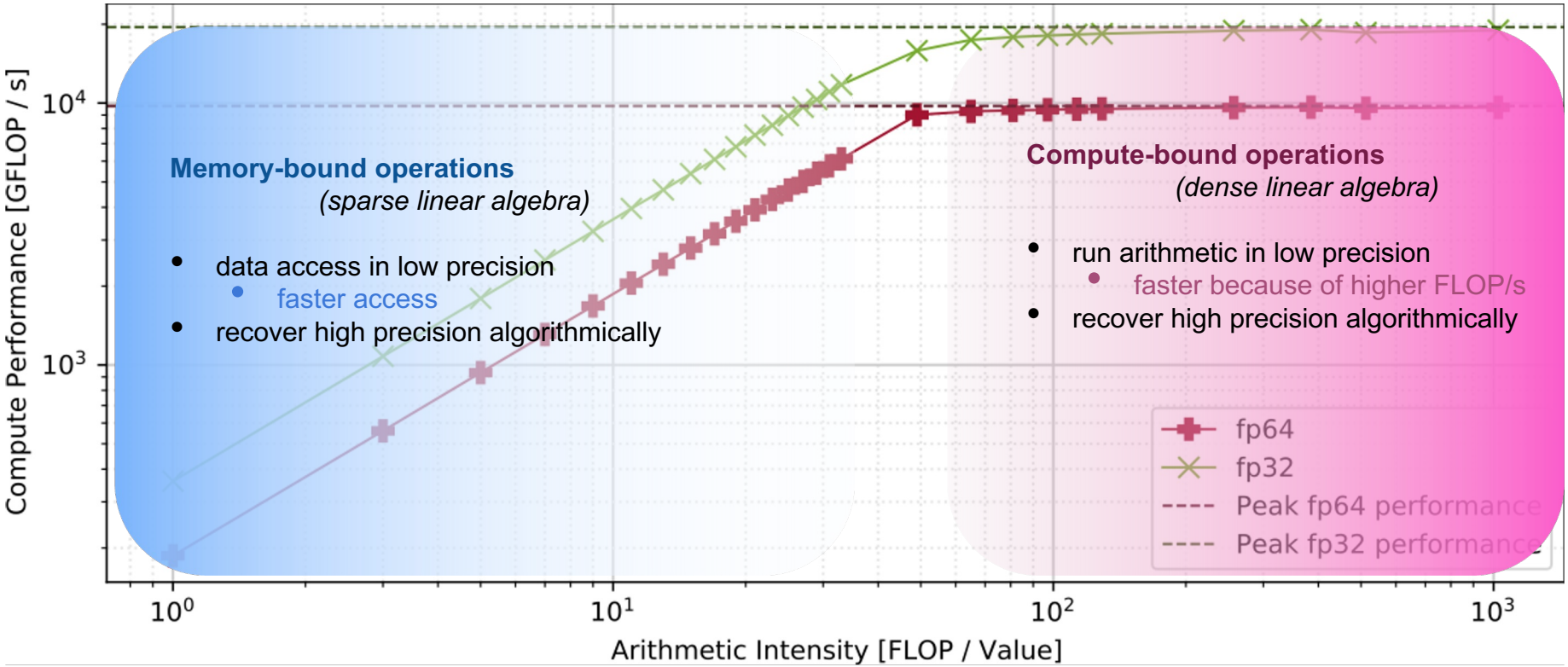
~2x faster!

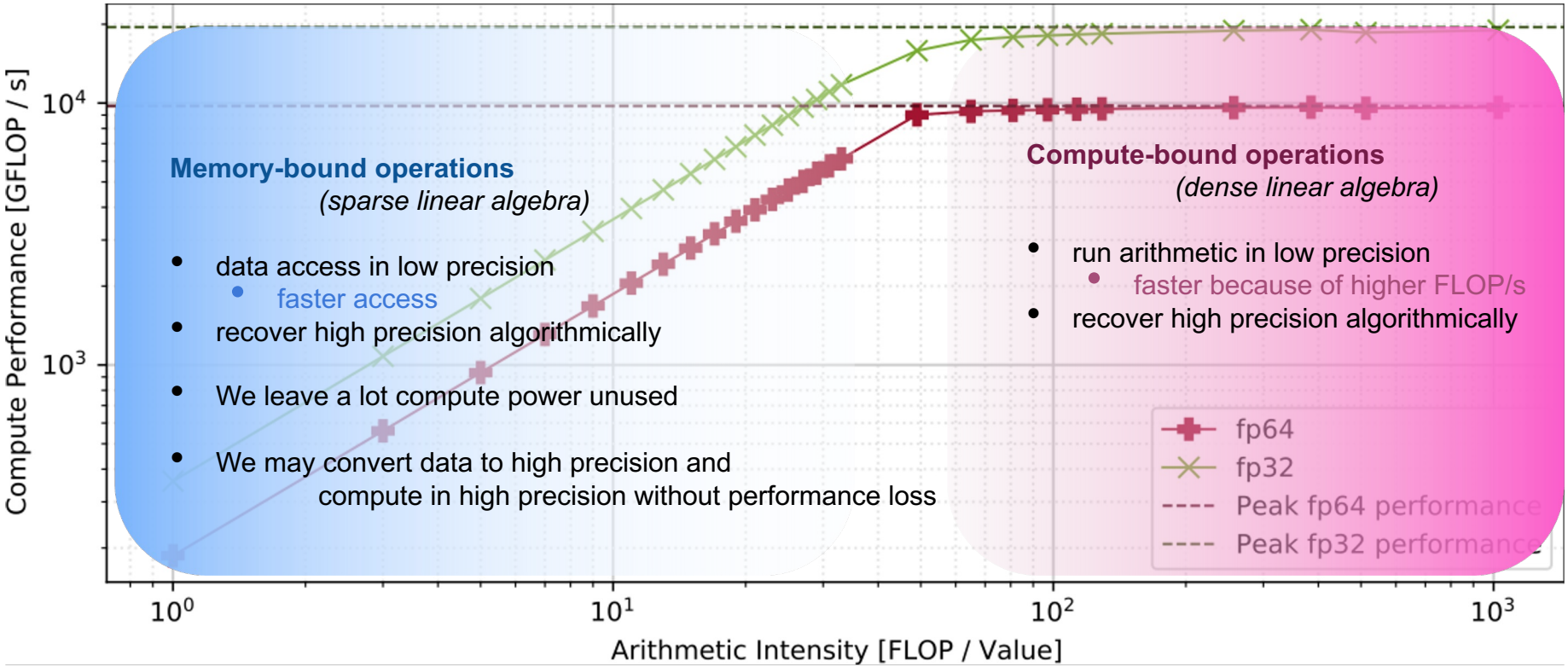


forward error ≈ (unit round-off) * (linear system's condition number)

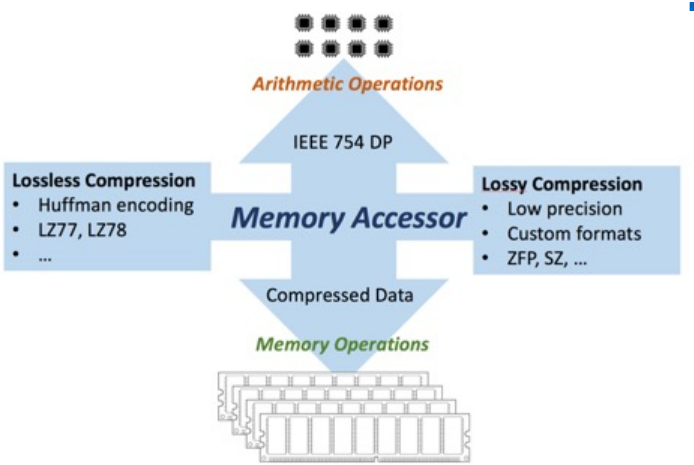
N. Higham: Accuracy and stability of numerical algorithms. SIAM, 2002.

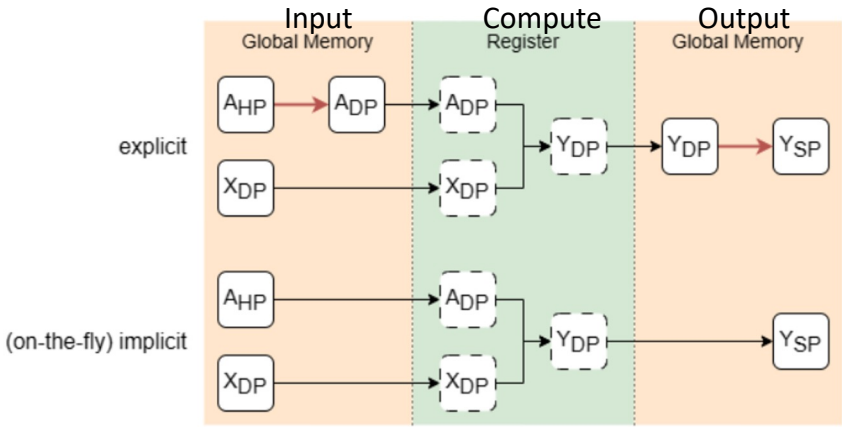

Matrix fp32
Matrix fp64
Memory-bound operations
(sparse linear algebra)
Compute-bound operations
(dense linear algebra)
fp64
fp32
Peak fp64 performance
Peak fp32 performance
Compute Performance [GFLOP / s]
10^4
10^3
10^0
10^1
10^2
10^3
Arithmetic Intensity [FLOP / Value]


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

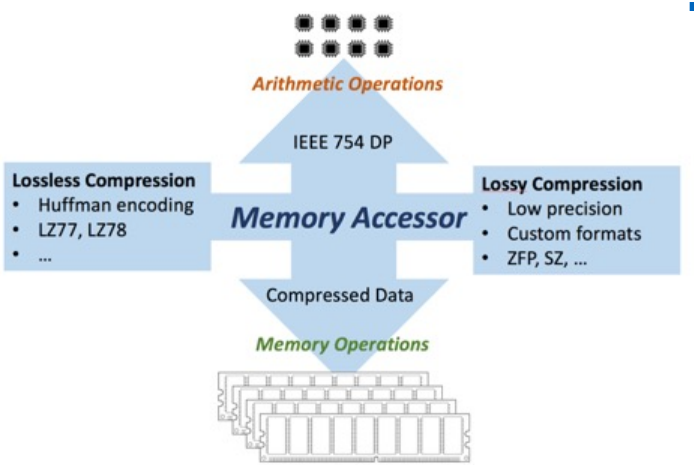

Memory-bound operations
(sparse linear algebra)
data access in low precision
faster access
recover high precision algorithmically
We leave a lot compute power unused
We may convert data to high precision and
compute in high precision without performance loss
Compute-bound operations
(dense linear algebra)
run arithmetic in low precision
faster because of higher FLOP/s
recover high precision algorithmically
fp64
fp32
Peak fp64 performance
Peak fp32 performance
Compute Performance [GFLOP / s]
10^4
10^3
10^0
10^1
10^2
10^3
Arithmetic Intensity [FLOP / Value]

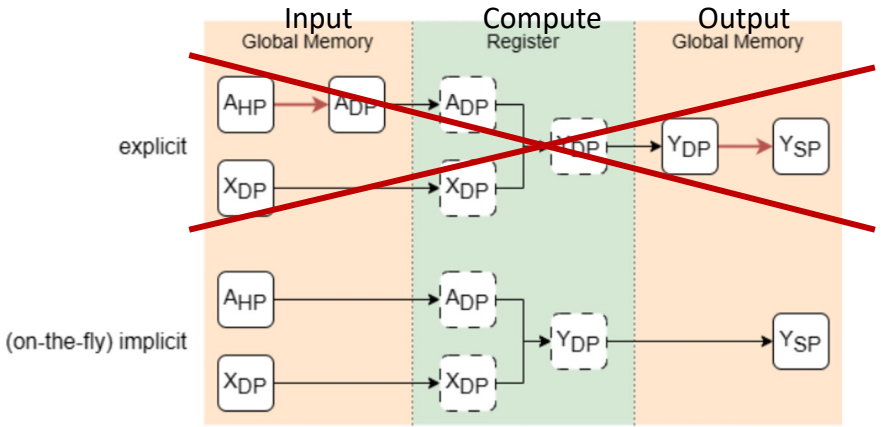
- Traditionally, we use a strong coupling between the precision formats used for arithmetic operations and storing data.
- *Maybe this is not the right thing?*
- *We should compute in fp64*
- *Maybe we should use the free compute cycles (vector/tensor cores) to compress the data*
- *Compression / Conversion needs to happen on-the-fly*

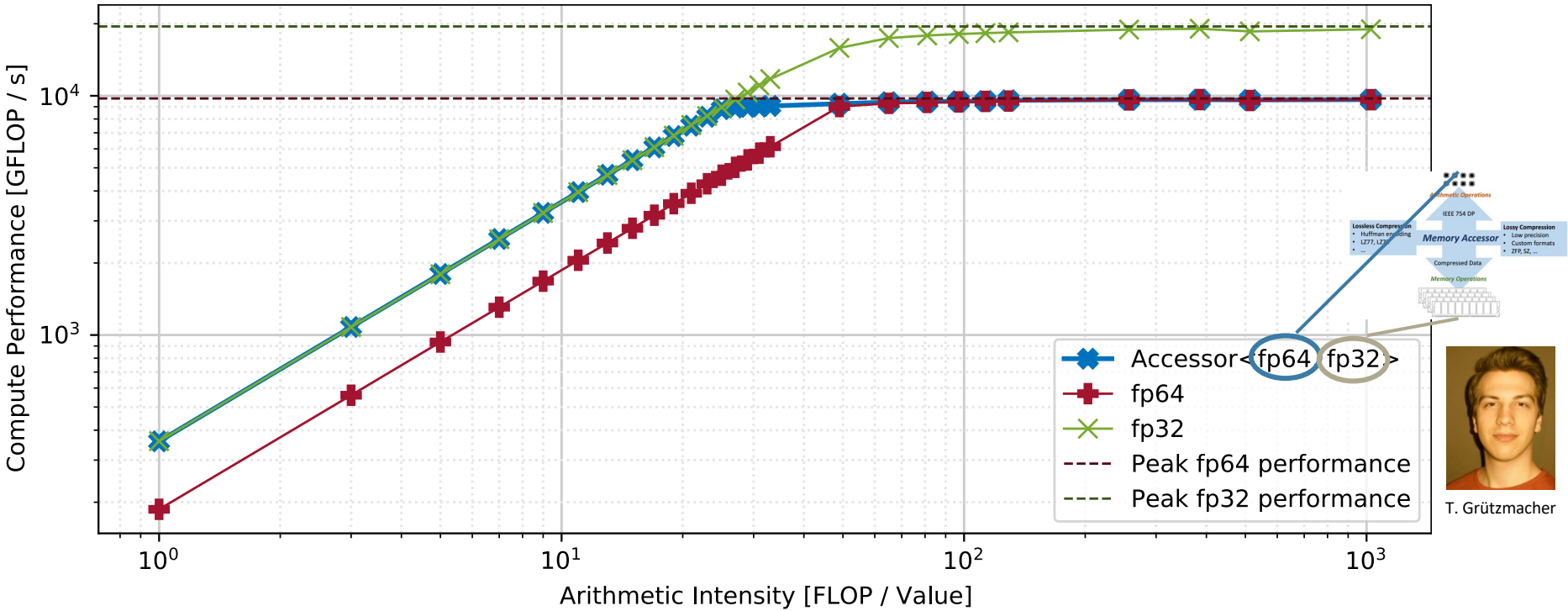




- Traditionally, we use a strong coupling between the precision formats used for arithmetic operations and storing data.
- *Maybe this is not the right thing?*
- *We should compute in fp64*
- *Maybe we should use the free compute cycles (vector/tensor cores) to compress the data*
- *Compression / Conversion needs to happen on-the-fly*







T. Grützmacher

Linear System Ax=b with cond(A) ≈ 10^7

(apache2 from SuiteSparse) **NVIDIA V100 GPU**

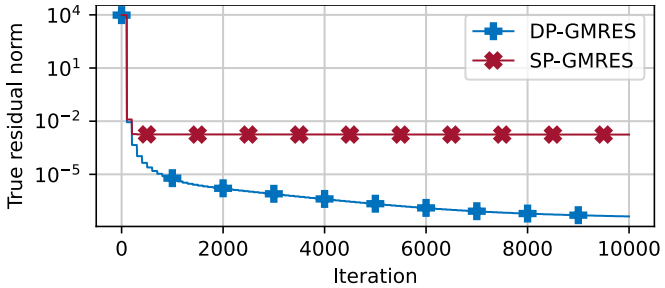
```
Double precision GMRES
Initial residual norm
sqrt(r^t r): 9670.36
Final residual norm
sqrt(r^T r): 9.6639e-09
GMRES iteration count: 23271
GMRES execution time: 43801 ms
```

Relative residual ~10^{-12}

```
Single precision GMRES
Initial residual norm
sqrt(r^t r): 9670.36
Final residual norm
sqrt(r^T r): 0.00175464
GMRES iteration count: 25000
GMRES execution time: 27376 ms
```

Relative residual ~10^{-7}

~2x faster!

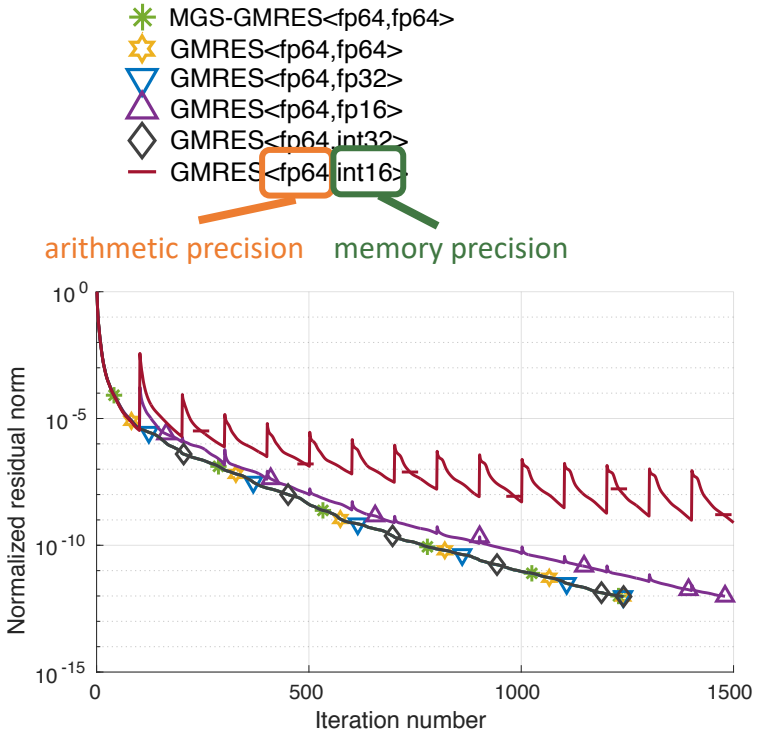


forward error ≈ (unit round-off) * (linear system's condition number)

N. Higham: Accuracy and stability of numerical algorithms. SIAM, 2002.

Compressed Basis (CB-) GMRES

- Use double precision in all arithmetic operations;
- Store Krylov basis vectors in lower precision;
  - Search directions are no longer DP-orthogonal;
  - Hessenberg system maps solution to “perturbed” Krylov subspace;
  - Additional iterations may be needed;
  - As long as the loss-of-orthogonality is moderate, we should see moderate convergence degradation;



Linear System Ax=b with cond(A) ≈ 10^7

(apache2 from SuiteSparse) **NVIDIA V100 GPU**

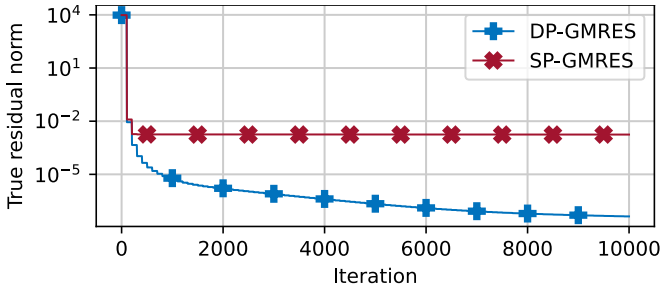
```
Double precision GMRES
Initial residual norm
sqrt(r^t r): 9670.36
Final residual norm
sqrt(r^T r): 9.6639e-09
GMRES iteration count: 23271
GMRES execution time: 43801 ms
```

Relative residual ~10^{-12}

```
Single precision GMRES
Initial residual norm
sqrt(r^t r): 9670.36
Final residual norm
sqrt(r^T r): 0.00175464
GMRES iteration count: 25000
GMRES execution time: 27376 ms
```

Relative residual ~10^{-7}

~2x faster!

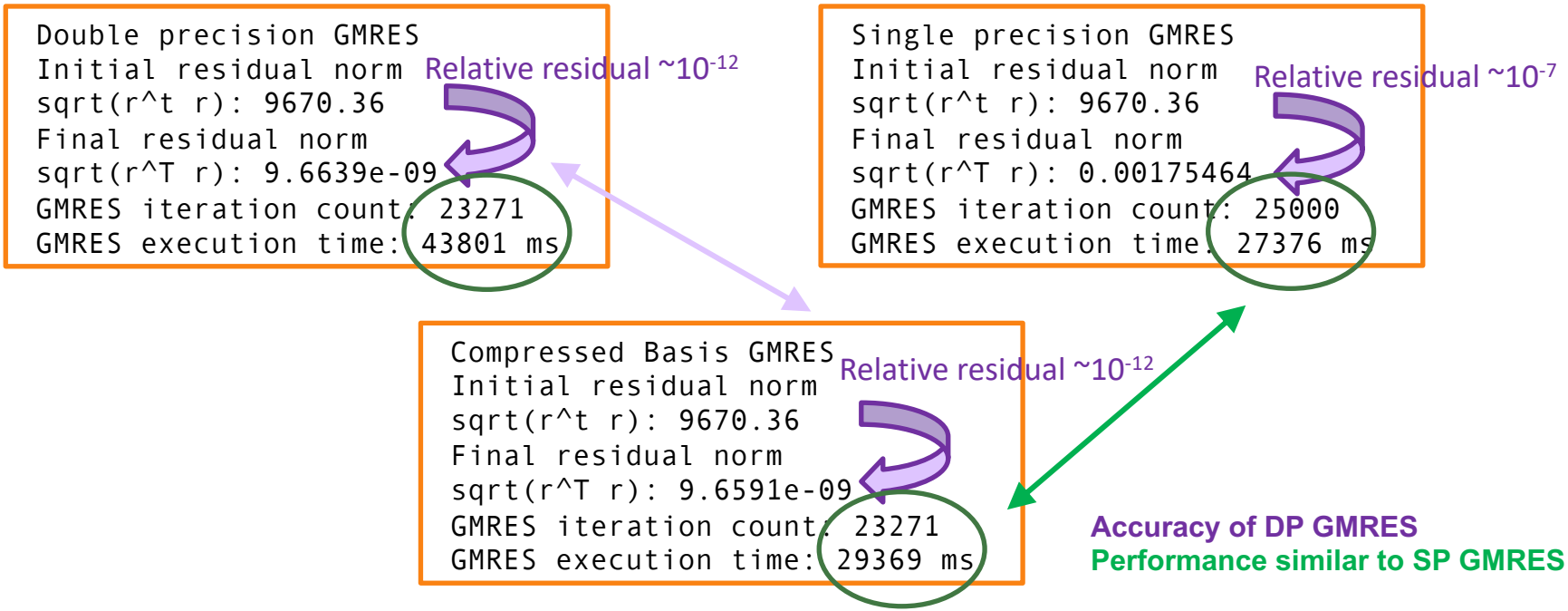


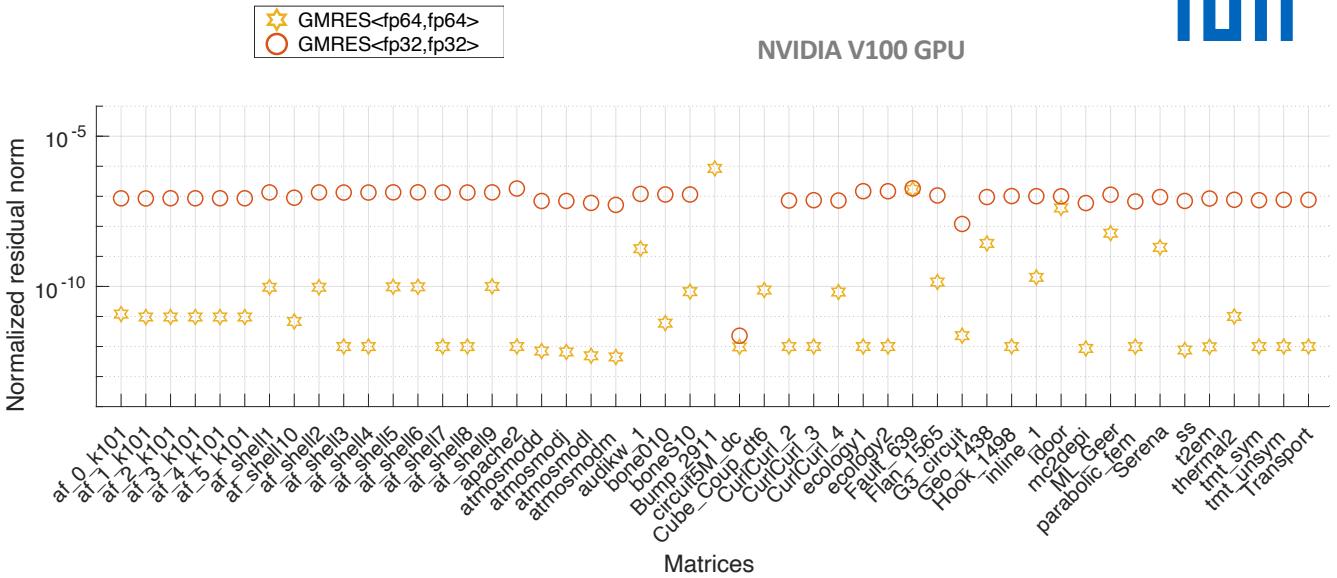
forward error ≈ (unit round-off) * (linear system's condition number)

N. Higham: Accuracy and stability of numerical algorithms. SIAM, 2002.

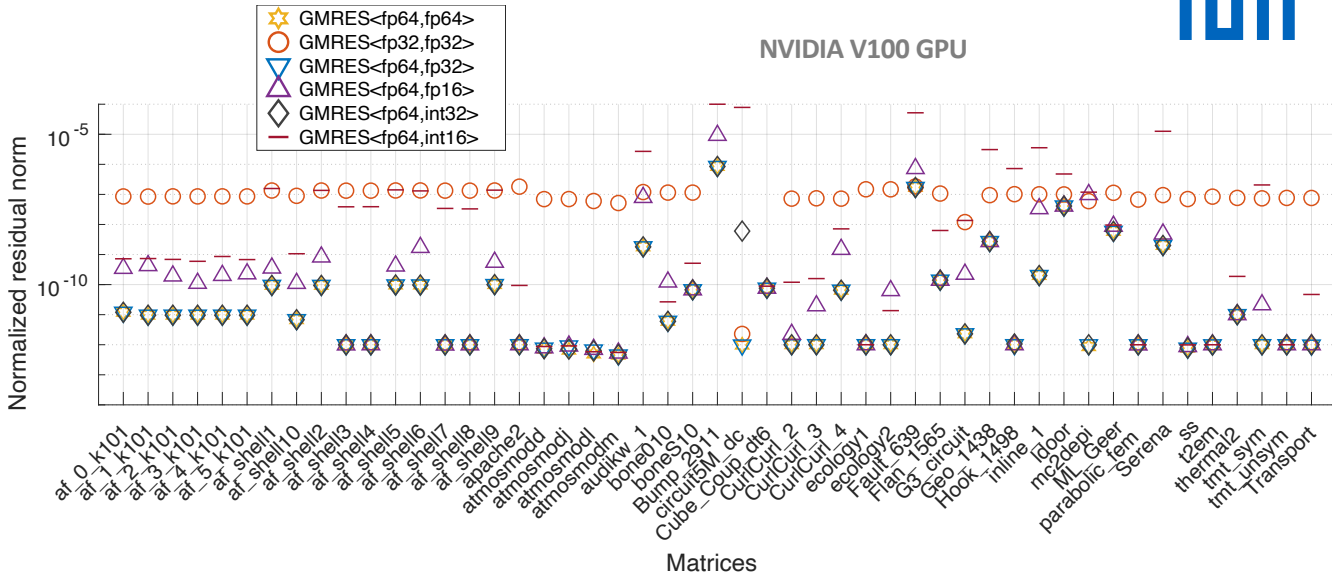
Linear System Ax=b with cond(A) ≈ 10^7

(apache2 from SuiteSparse) **NVIDIA V100 GPU**



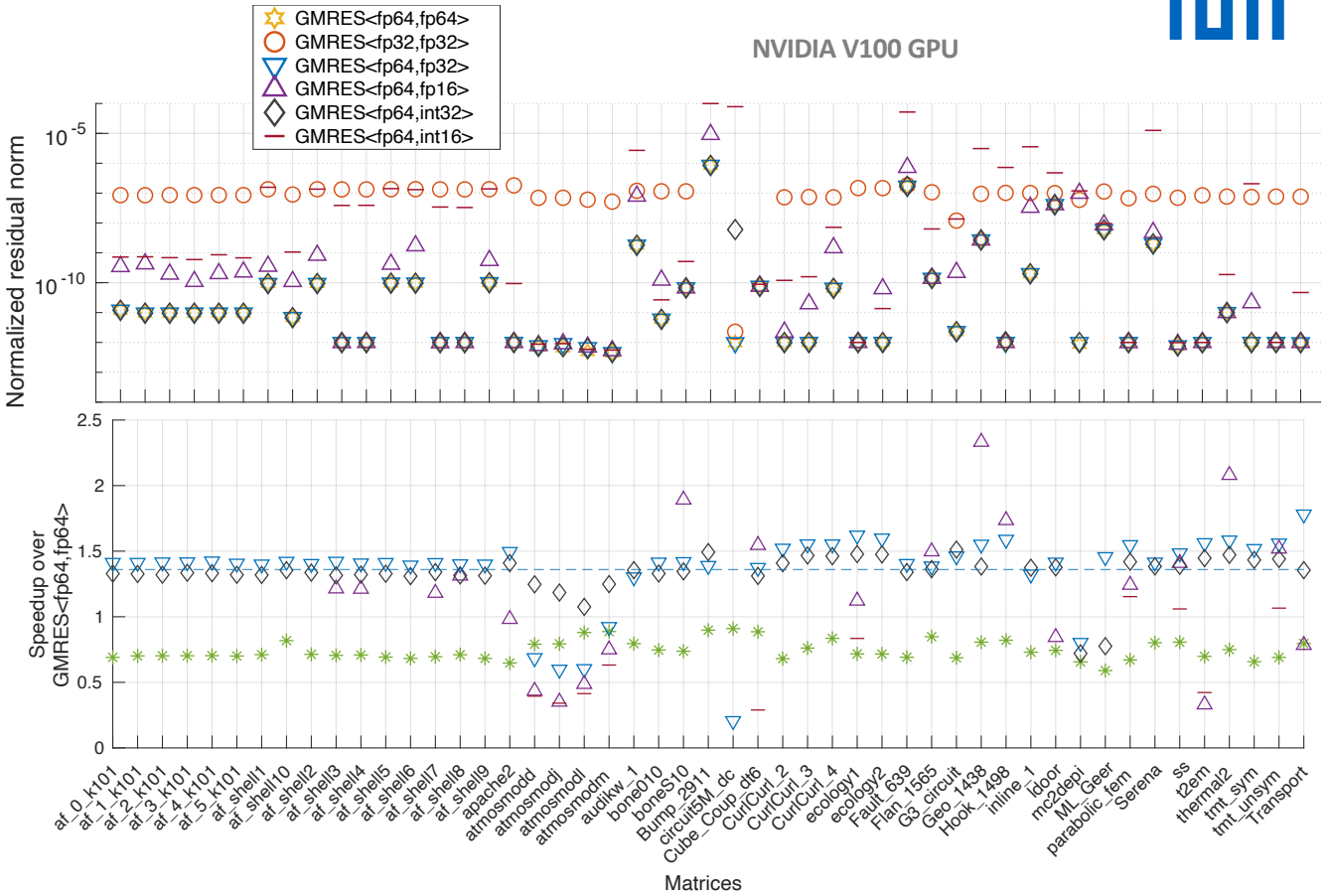

GMRES<fp64,fp64>
GMRES<fp32,fp32>
NVIDIA V100 GPU
Normalized residual norm
10^-5
10^-10
af_0_k101
af_1_k101
af_2_k101
af_3_k101
af_4_k101
af_5_k101
af_shell1
af_shell10
af_shell2
af_shell3
af_shell4
af_shell5
af_shell6
af_shell7
af_shell8
af_shell9
apache2
atmosmodd
atmosmodj
atmosmodl
atmosmodm
audikw_1
bone010
boneS10
Bump_2911
circuit5M_dc
Cube_Coup_dt6
CurlCurl_2
CurlCurl_3
CurlCurl_4
ecology1
ecology2
Fault_639
Flan_1565
G3_circuit
Geo_1438
Hook_1498
inline_1
ldoor
mc2depi
ML_Geer
parabolic_fem
Serena
ss
t2em
thermal2
tmt_sym
tmt_unsym
Transport
Matrices

- CB-GMRES using 32-bit storage preserves DP accuracy (SP-GMRES does not)



- CB-GMRES using 32-bit storage preserves DP accuracy (SP-GMRES does not)

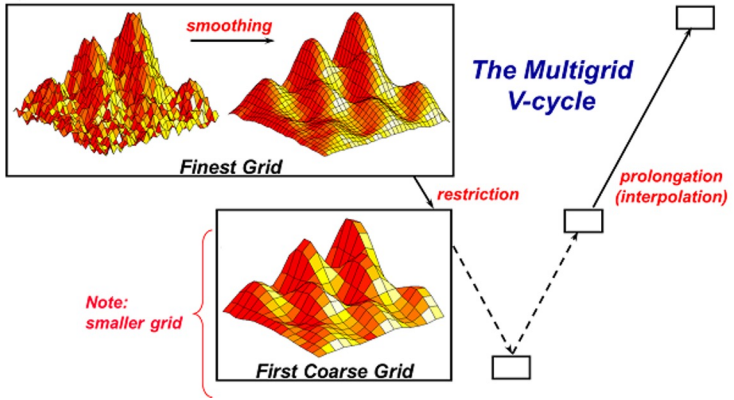
- Speedups problem-dependent
- Speedup Ø1.4x (for restart 100)
- 16-bit storage mostly inefficient

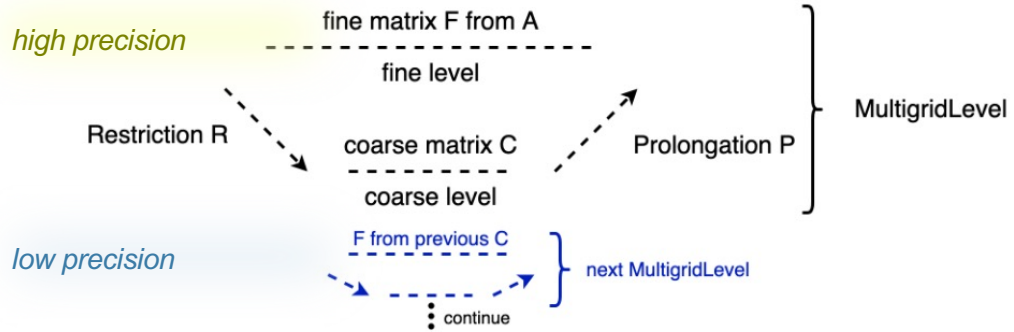


Ginkgo

Aliaga JI, Anzt H, Grützmacher T, Quintana-Ortí ES, Tomás AE. Compressed basis GMRES on high-performance graphics processing units. *The International Journal of High Performance Computing Applications*. 2022;0(0). doi:10.1177/10943420221115140

Mixed Precision Multigrid



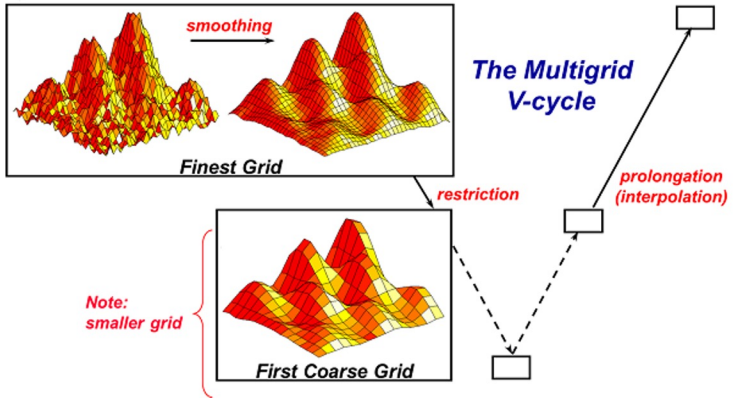


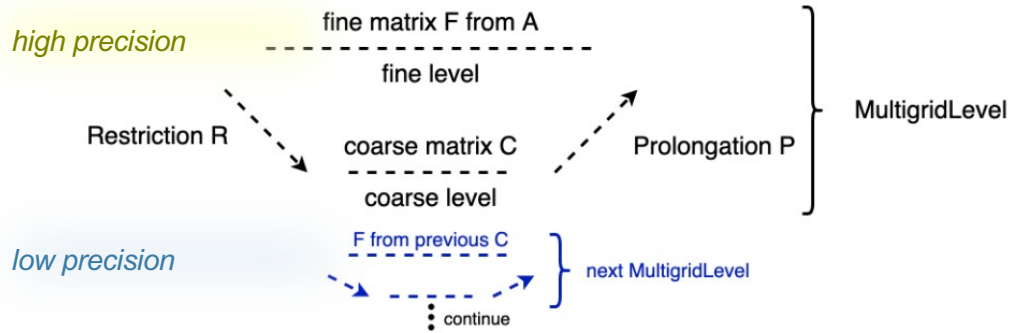
Mike Tsai

MS3

Linear Solvers in Large Distributed Applications - Part I of III

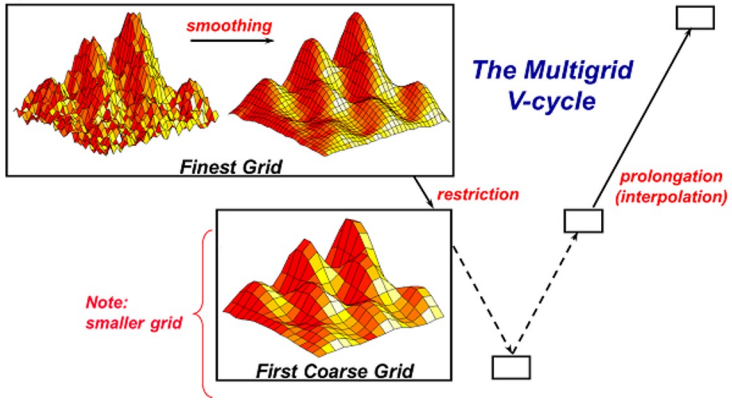
Stephen F. McCormick, Joseph Benzaken, Rasmus Tamstorf: Algebraic error analysis for mixed-precision multigrid solvers, https://arxiv.org/abs/2007.06614

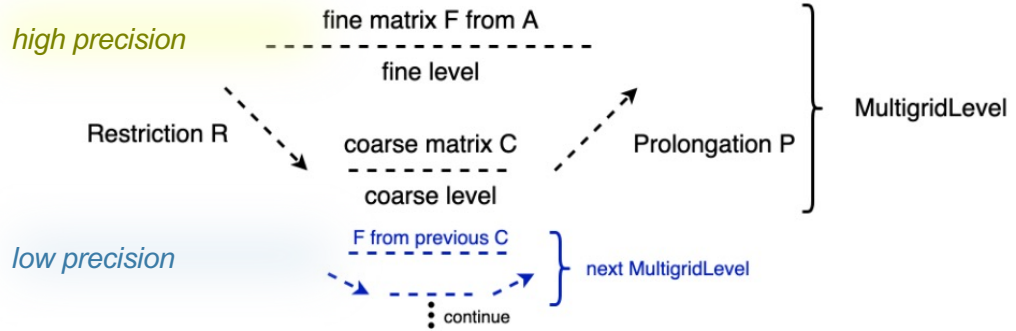




Ginkgo

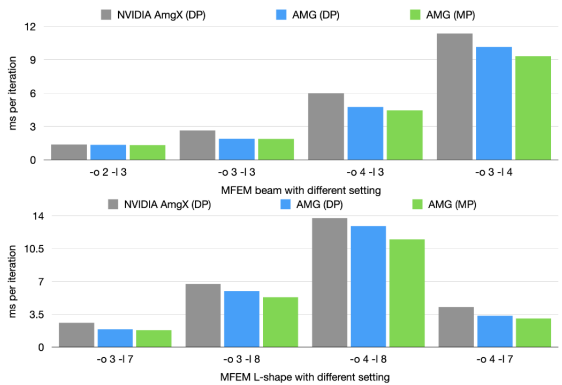
```
multigrid::build()
    .with_max_levels(10u) // equal to NVIDIA/AMGX 11 max levels
    .with_min_coarse_row(64u)
    .with_pre_smoother(sm, sm_f)
    .with_mg_level(pgm, pgm_f)
    .with_level_selector(
        [](const size_type level, const LinOp*) -> size_type {
            // Only the first level is generated by MultigridLevel(double).
            // The subsequent levels are generated by MultigridLevel(float)
            return level >= 1 ? 1 : 0;
        })
    .with_coarest_solver(coarest_solver_f)
```

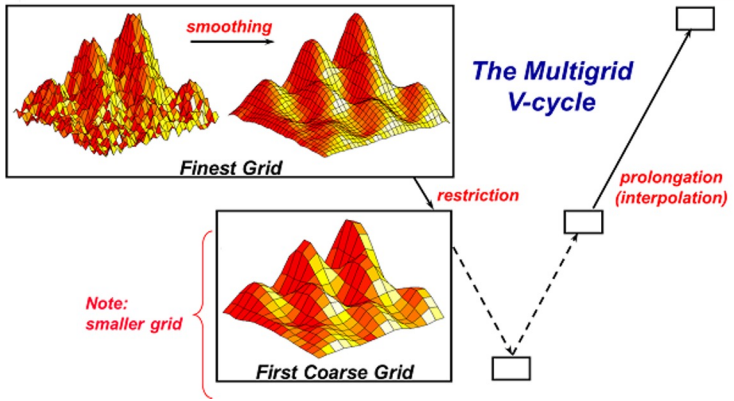


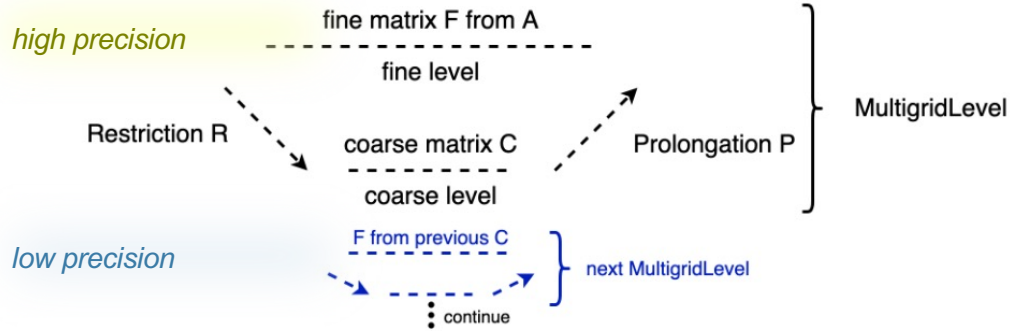


Ginkgo

```
multigrid::build()
    .with_max_levels(10u) // equal to NVIDIA/AMGX 11 max levels
    .with_min_coarse_row(64u)
    .with_pre_smoother(sm, sm_f)
    .with_mg_level(pgm, pgm_f)
    .with_level_selector(
        [](const size_type level, const LinOp*) -> size_type {
            // Only the first level is generated by MultigridLevel(double).
            // The subsequent levels are generated by MultigridLevel(float)
            return level >= 1 ? 1 : 0;
        })
    .with_coarest_solver(coarest_solver_f)
```

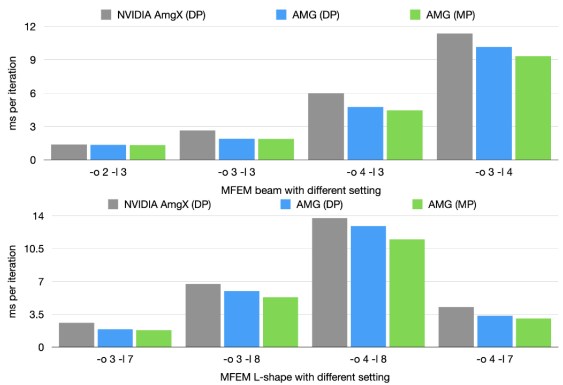






Ginkgo

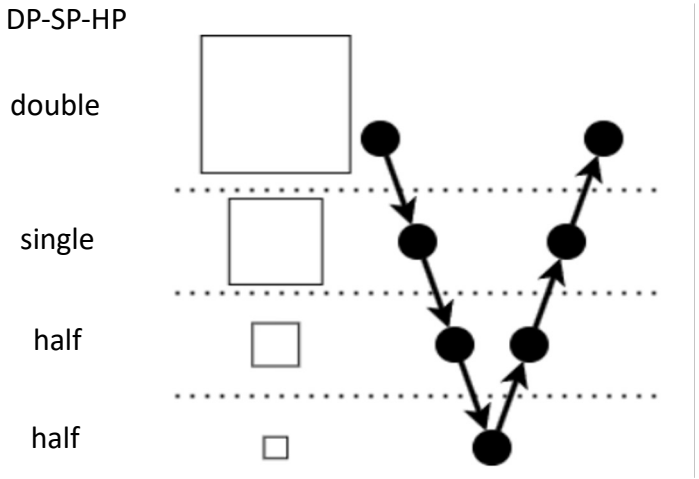
```
multigrid::build()
    .with_max_levels(10u) // equal to NVIDIA/AMGX 11 max levels
    .with_min_coarse_row(64u)
    .with_pre_smoother(sm, sm_f)
    .with_mg_level(pgm, pgm_f)
    .with_level_selector(
        [](const size_type level, const LinOp*) -> size_type {
            // Only the first level is generated by MultigridLevel(double).
            // The subsequent levels are generated by MultigridLevel(float)
            return level >= 1 ? 1 : 0;
        })
    .with_coarest_solver(coarest_solver_f)
```

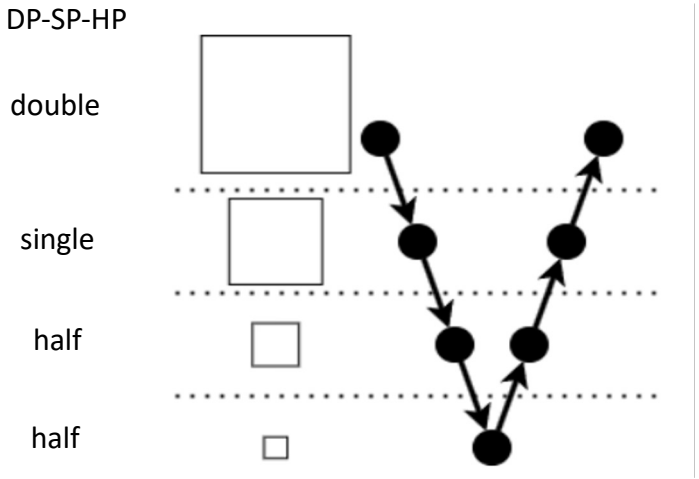



Mike Tsai

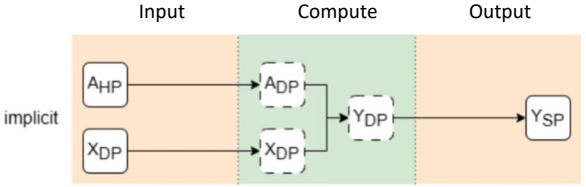

Natalie Beams

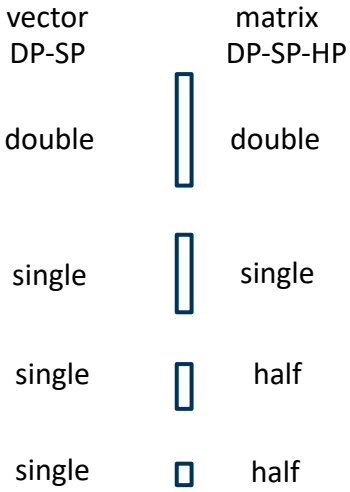
YHM Tsai, N Beams, H Anzt
Mixed Precision Algebraic Multigrid on GPUs
International Conference on Parallel Processing and Applied Mathematics, 113-125, 2022

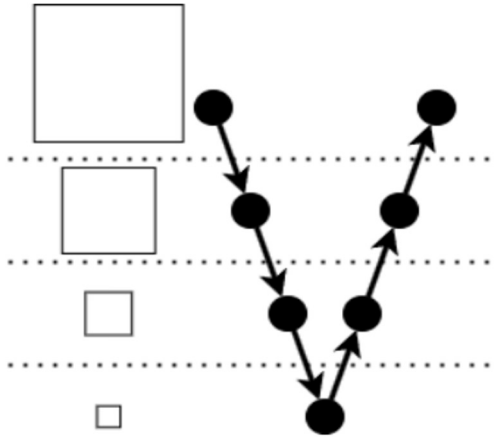

DP-SP-HP
double
single
half
half



problem	GINKGO's AMG (DP) res. norm	#iter	time[ms]	GINKGO's AMG (DP-SP) res. norm	#iter	time[ms]	GINKGO's AMG (DP-SP-HP) res. norm	#iter	time[ms]	GINKGO's AMG (DP-HP) res. norm	#iter	time[ms]
2cubes	6.56e-09	20	14.62	6.56e-09	20	12.91	NaN	800	493.44	NaN	800	485.06
cage13	3.68e-10	11	15.49	3.68e-10	11	14.09	4.24e-10	11	14.18	6.15e-10	15	17.79
cage14	3.41e-10	10	32.98	3.41e-10	10	29.80	3.73e-10	10	29.04	7.02e-10	13	35.15
offshore	1203.64	800	770.56	1533.85	800	688.64	NaN	800	706.05	NaN	800	671.65
thermal2	2.18e-06	349	489.08	2.38e-06	425	536.09	NaN	800	952.22	NaN	800	919.08
tmt_sym	6.95e-05	359	371.70	7.41e-05	401	379.21	NaN	800	703.40	NaN	800	684.14
beam(o3l3)	3.05e-15	44	43.65	3.05e-15	44	40.89	5.53e-15	86	80.72	7.27e-15	127	117.62
l-shape(o3l7)	4.64e-14	160	195.31	4.78e-14	171	191.62	3.74e-11	800	869.77	2.19e-07	800	868.83







(a) fp32: Single-precision IEEE Floating Point Format

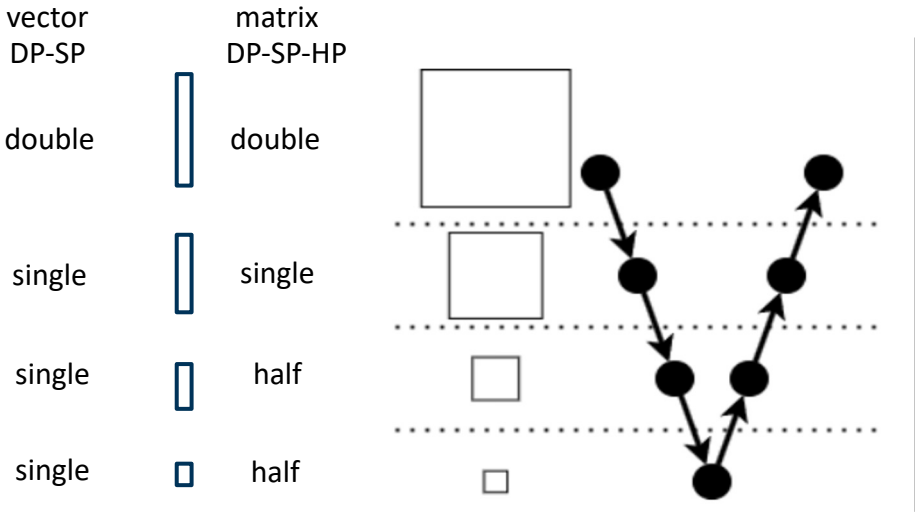
Range: ~1e^{-38} to ~3e^{38}

(b) fp16: Half-precision IEEE Floating Point Format

Range: ~5.96e^{-8} to 65504

(c) bfloat16: Brain Floating Point Format

Range: ~1e^{-38} to ~3e^{38}



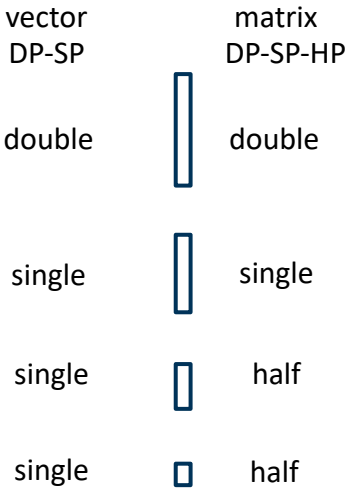
problem	Ginkgo's AMG (DP-SP-HP) res. norm	#iter	time[ms]	Ginkgo's AMG (DP-HP) res. norm	#iter	time[ms]	Ginkgo's AMG (DP-SP-BF) res. norm	#iter	time[ms]	Ginkgo's AMG (DP-BF) res. norm	#iter	time[ms]
2cubes	NaN	800	493.44	NaN	800	485.06	6.70e-09	20	13.53	1.15e-08	20	13.31
cage13	4.24e-10	11	14.18	6.15e-10	15	17.79	3.67e-10	11	13.88	3.71e-10	11	13.76
cage14	3.73e-10	10	29.04	7.02e-10	13	35.15	3.40e-10	10	29.46	3.42e-10	10	27.60
offshore	NaN	800	706.05	NaN	800	671.65	1416.76	800	713.00	2045.25	800	692.75
thermal2	NaN	800	952.22	NaN	800	919.08	2.80e-06	597	730.73	2.92e-06	635	723.24
tmt_sym	NaN	800	703.4	NaN	800	684.14	9.59e-05	649	593.51	9.87e-05	754	665.21
beam(o3l3)	5.53e-15	86	80.72	7.27e-15	127	117.62	3.15e-15	44	42.90	3.29e-15	44	42.99
l-shape(o3l7)	3.74e-11	800	869.77	2.19e-07	800	868.83	5.01e-14	186	203.06	5.10e-14	192	208.69

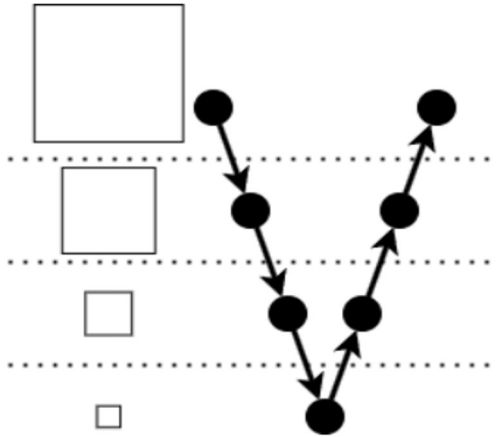
Mike Tsai

Natalie Beams

Ginkgo

YHM Tsai, N Beams, H Anzt
Three-precision algebraic multigrid on GPUs Future Generation Computer Systems 149, 280-293, 2023.





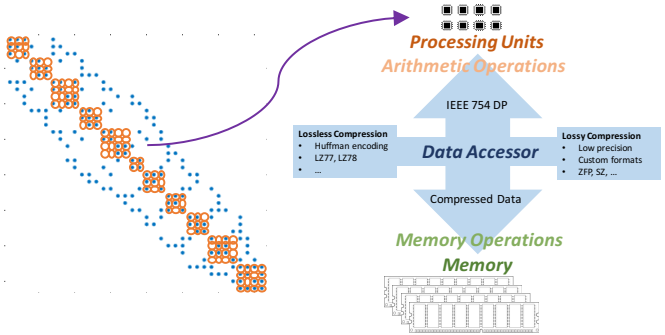
problem	Ginkgo's AMG (DP-SP-HP) res. norm	#iter	time[ms]	Ginkgo's AMG (DP-HP) res. norm	#iter	time[ms]	Ginkgo's AMG (DP-SP-BF) res. norm	#iter	time[ms]	Ginkgo's AMG (DP-BF) res. norm	#iter	time[ms]
2cubes	NaN	800	493.44	NaN	800	485.06	6.70e-09	20	13.53	1.15e-08	20	13.31
cage13	4.24e-10	11	14.18	6.15e-10	15	17.79	3.67e-10	11	13.88	3.71e-10	11	13.76
cage14	3.73e-10	10	29.04	7.02e-10	13	35.15	3.40e-10	10	29.46	3.42e-10	10	27.60
offshore	NaN	800	706.05	NaN	800	671.65	1416.76	800	713.00	2045.25	800	692.75
thermal2	NaN	800	952.22	NaN	800	919.08	2.80e-06	597	730.73	2.92e-06	635	723.24
tmt_sym	NaN	800	703.4	NaN	800	684.14	9.59e-05	649	593.51	9.87e-05	754	665.21
beam(o3l3)	5.53e-15	86	80.72	7.27e-15	127	117.62	3.15e-15	44	42.90	3.29e-15	44	42.99
l-shape(o3l7)	3.74e-11	800	869.77	2.19e-07	800	868.83	5.01e-14	186	203.06	5.10e-14	192	208.69

Mixed Precision Preconditioning

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

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

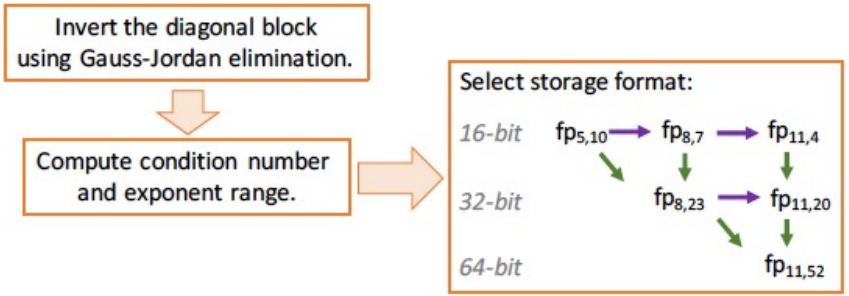
- **Block-Jacobi preconditioner** is based on **block-diagonal scaling**: $P = diag_B(A)$
  - Each block corresponds to one (small) linear system.
    - *Larger* blocks typically improve convergence.
    - *Larger* blocks make block-Jacobi more expensive.



- *Why should we store the preconditioner matrix P^{-1} in full (high) precision?*

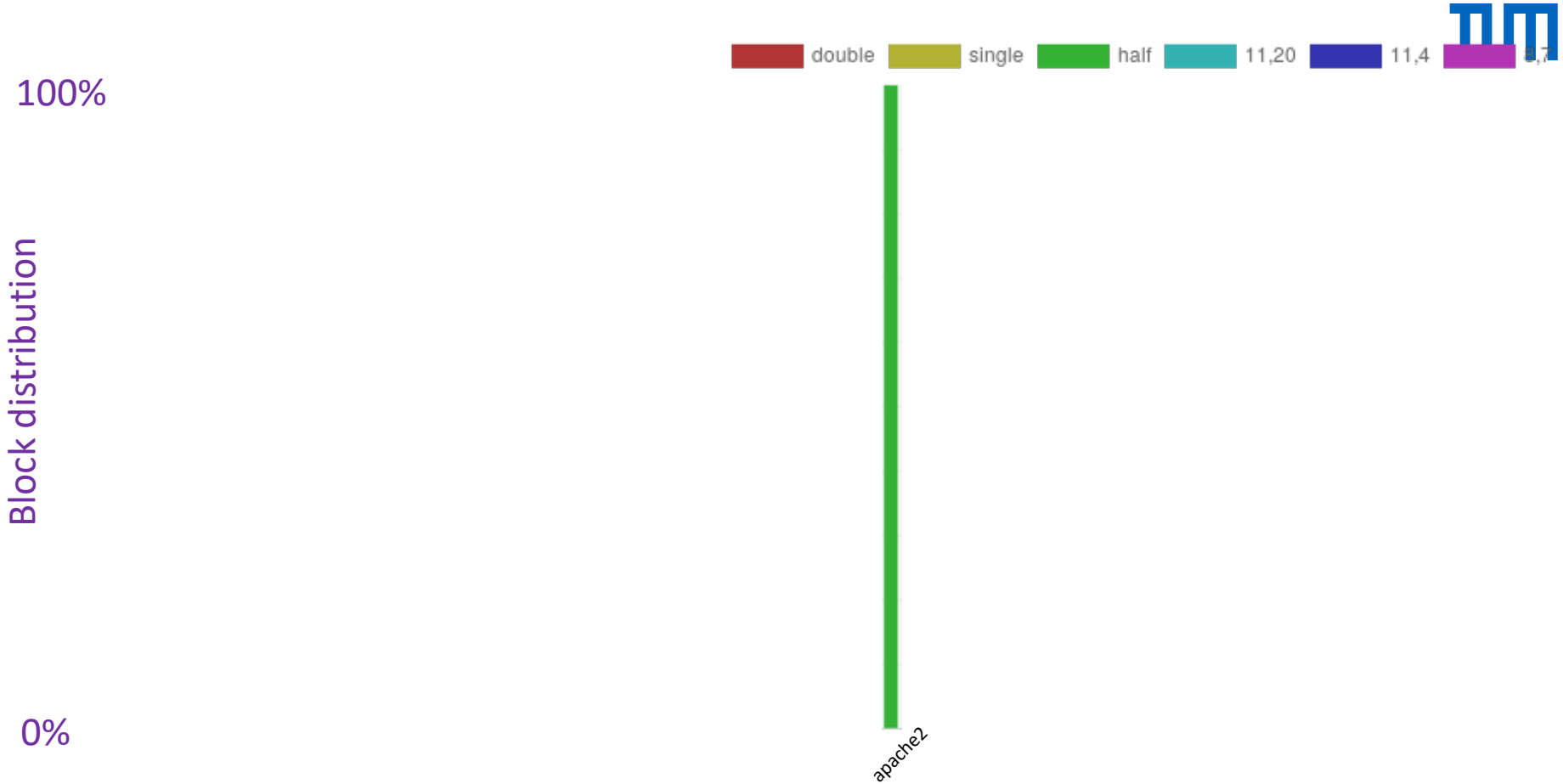
- Use the accessor to store the inverted diagonal blocks in lower precision.
  - *Be careful to preserve the regularity of each inverted diagonal block!*

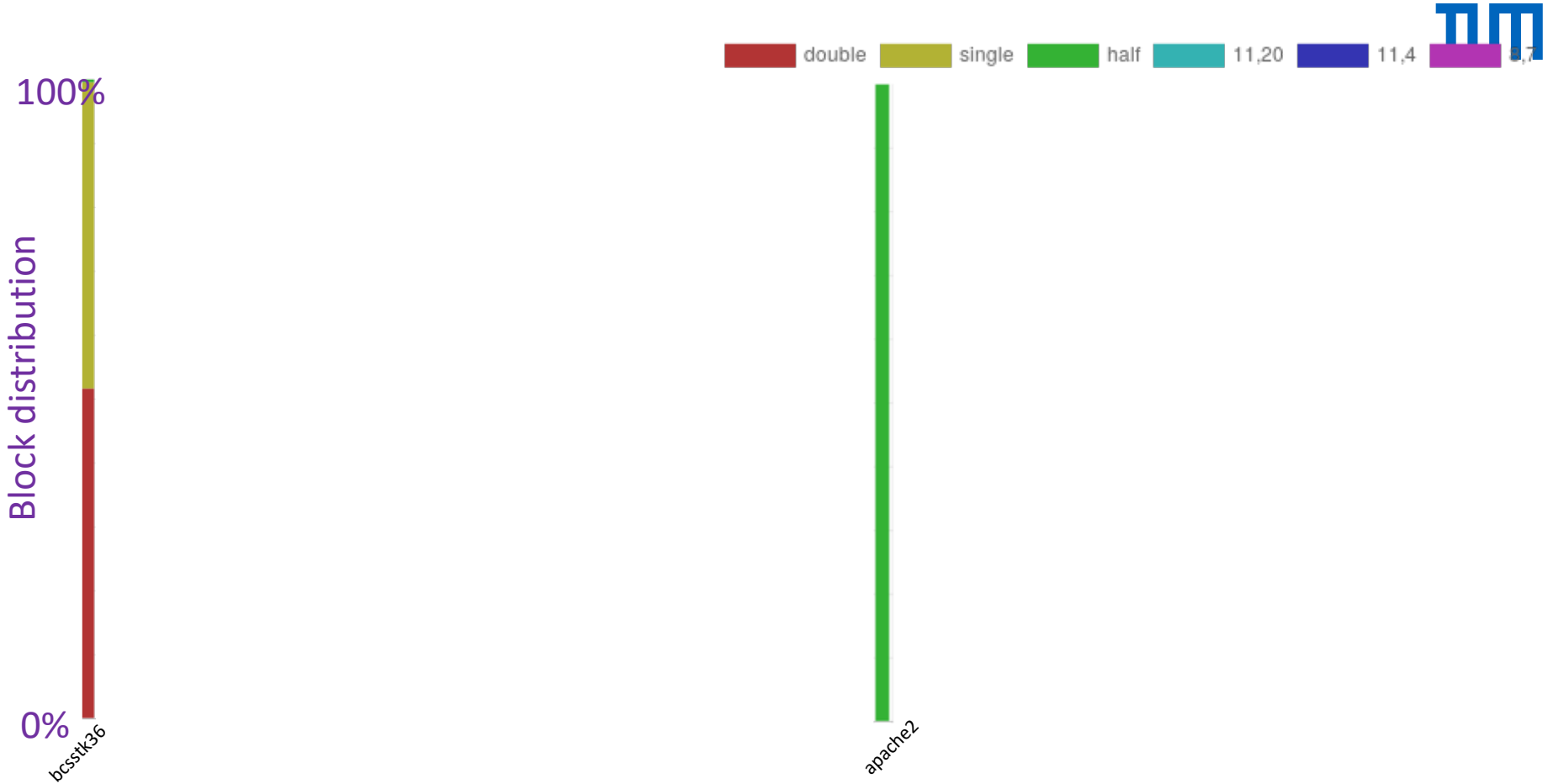
- Choose how much accuracy of the preconditioner should be preserved in the selection of the storage format.
- All computations use double precision, but store blocks in lower precision.

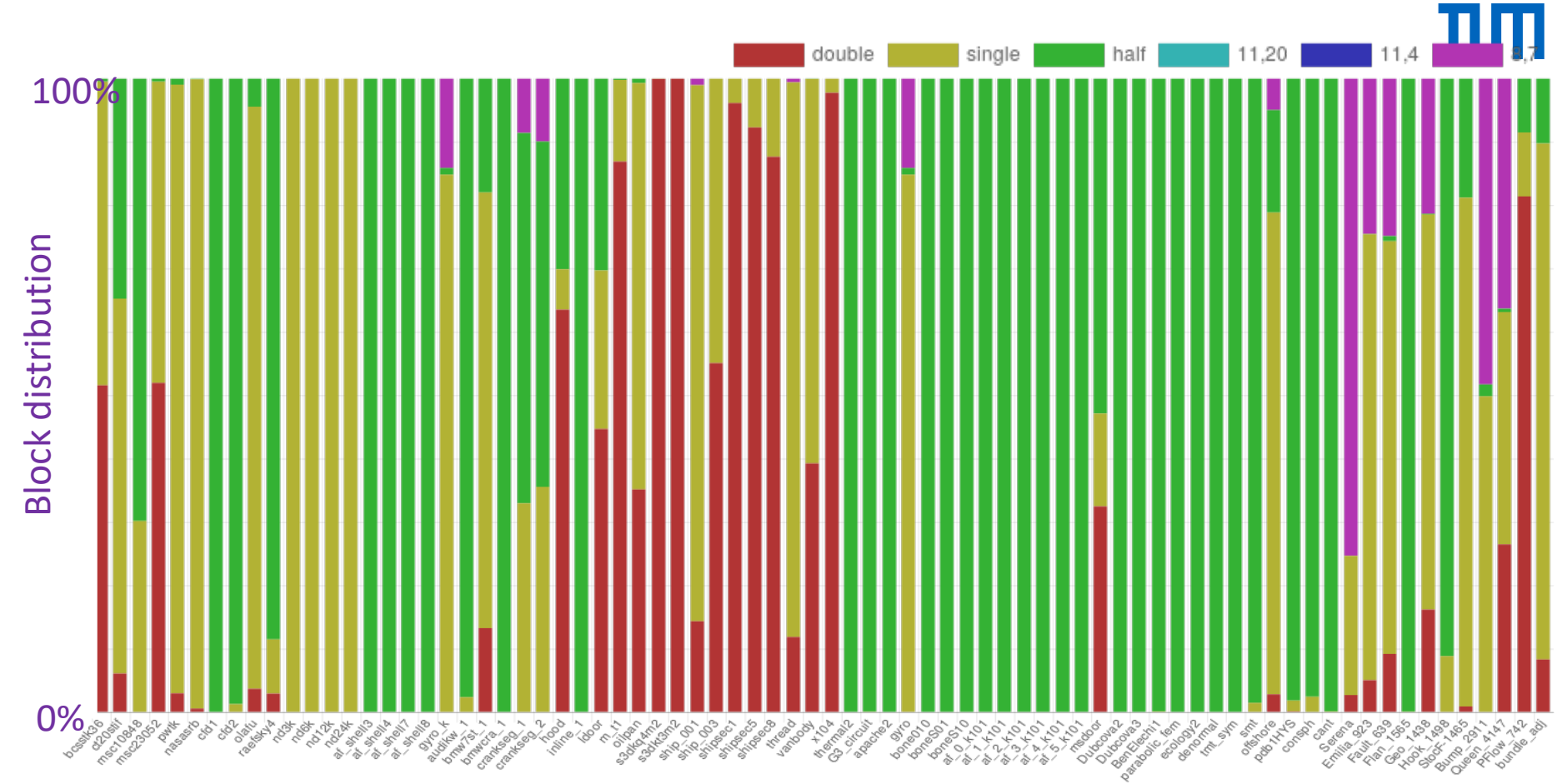


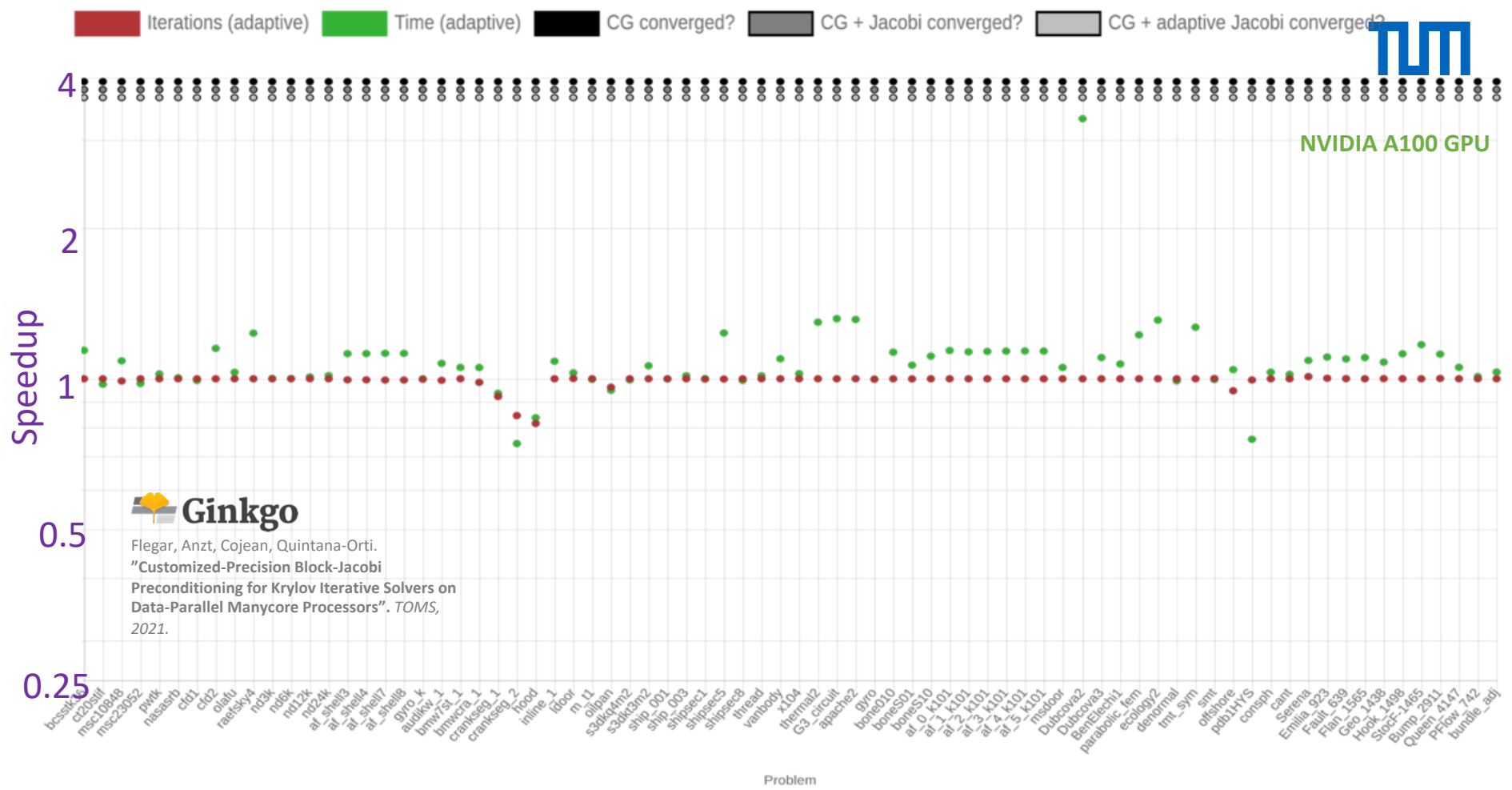
+ **Regularity preserved;**
+ Flexibility in the accuracy;
+ ”Not a low precision preconditioner”
  + Preconditioner is a constant operator;
  + No flexible Krylov solver needed ;

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


double
single
half
11,20
11,4
6,7
100%
Block distribution
0%
apache2


double
single
half
11,20
11,4
8,7
100%
Block distribution
0%
bcsstk36
apache2

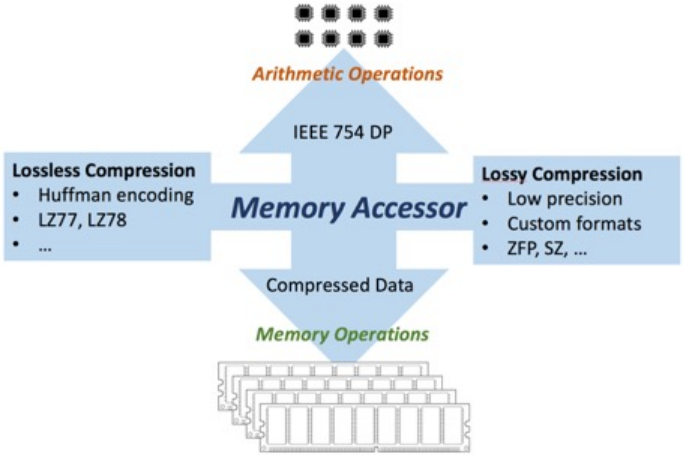

double
single
half
11,20
11,4
8,7
100%
0%
Block distribution
bcsstk36
d20stif
msc10848
msc23052
pwtk
nasasrb
cfd1
cfd2
olafu
raefsky4
nd3k
nd6k
nd12k
nd24k
af_shell3
af_shell4
af_shell7
af_shell8
gyro_k
audikw_1
bmw7st_1
bmwcra_1
crankseg_1
crankseg_2
hood
inline_1
ldoor
m_t1
oilpan
s3dkq4m2
s3dkt3m2
ship_001
ship_003
shipsec1
shipsec5
shipsec8
thread
vanbody
x104
thermal2
G3_circuit
apache2
gyro
bone010
boneS01
boneS10
af_0_k101
af_1_k101
af_2_k101
af_3_k101
af_4_k101
af_5_k101
msdoor
Dubcova2
Dubcova3
BenElechi1
parabolic_fem
ecology2
denormal
tmt_sym
smt
offshore
pdb1HYS
consph
cant
Serena
Emilia_923
Fault_639
Flan_1565
Geo_1438
Hook_1498
StocF-1465
Bump_2911
Queen_4147
PFlow_742
bundle_adj

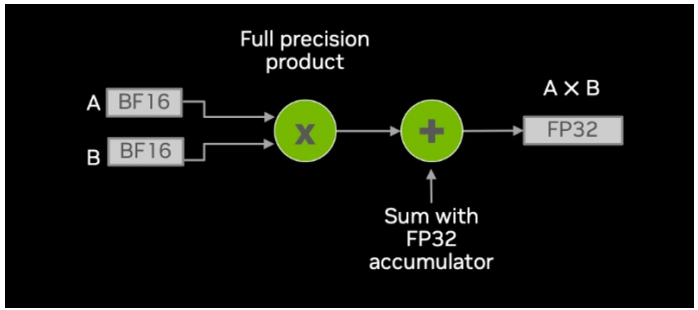


Ginkgo

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

Can mixed precision accelerate Sparse Solvers?
– Yes, but we have to work harder…





NVIDIA Tensor Core operation.

- The performance is limited by the bandwidth
- Hardware increasingly features powerful matrix engines
- We should use these ”free” FLOP/s
- One strategy is to use lossy/lossless data compression for memory operations and communication
- We need efficient need on-chip data compression

We have no standard for sparse operations

cuSPARSE, rocSPARSE, oneAPI, PETSc, Trilinos…
all have different interfaces & functionality

We try to define an interface that allows for horizontal and vertical compatibility:

- Useful as building blocks for high-level algorithms
- Vendors can wrap their current interface
- Horizontal: bindings for Fortran, C, …

SparseBLAS workshop

https://icl.utk.edu/workshops/sparseblas2023/index.html

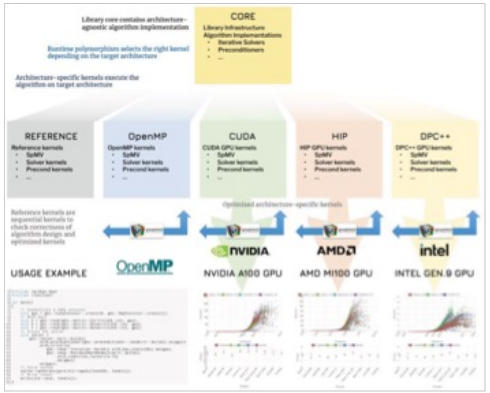
Next week:
https://icl.utk.edu/workshops/sparseblas2024/index.html

Want to participate in the discussion
– reach out hanzt@icl.utk.edu

Ginkgo

DESIGN

Ginkgo is a C++ framework for sparse numerical linear algebra. Using a universal linear operator abstraction, Ginkgo provides basic building blocks such as the sparse matrix vector product for a variety of matrix formats, iterative solvers, and preconditioners. Ginkgo targets multi- and many-core systems, and currently features back-ends for AMD GPUs, Intel CPU/GPUs, NVIDIA GPUs, and OpenMP-supporting architectures. Core functionality is separated from hardware-specific kernels for easy extension to other architectures, with runtime polymorphism selecting the correct kernels.



SUSTAINABILITY

Ginkgo is part of the Extreme-scale Scientific Software Stack (E4S) and the extreme-scale Software Development Kit (xSDK), and adopts the xSDK community policies for sustainable software development and high software quality. The source code of the Ginkgo library can be accessed in a public git repository on GitHub. Code development in Ginkgo is realized in a Continuous Integration / Continuous Benchmarking framework. GitLab runners are used on private servers and HPC clusters where Docker/Enroot is used to provide different compilation and execution environments. To test the correct execution, each functionality is complemented by unit tests. The unit testing is realized using the Google Test framework.

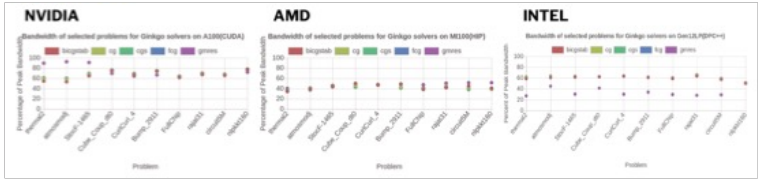
This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration, the Horizon2020 EuroHPC Project MICROCARD, and the Helmholtz Impuls und Vernetzungsfond VH-NG-1241.

SPONSORED BY

HARDWARE PARTNERS

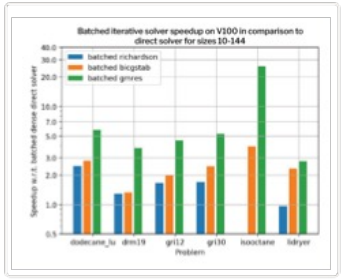
INTEGRATION

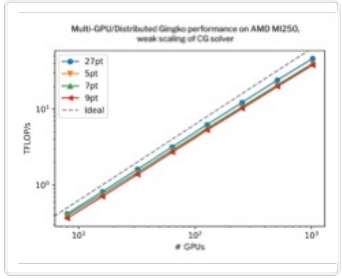
PERFORMANCE



FUNCTIONALITY

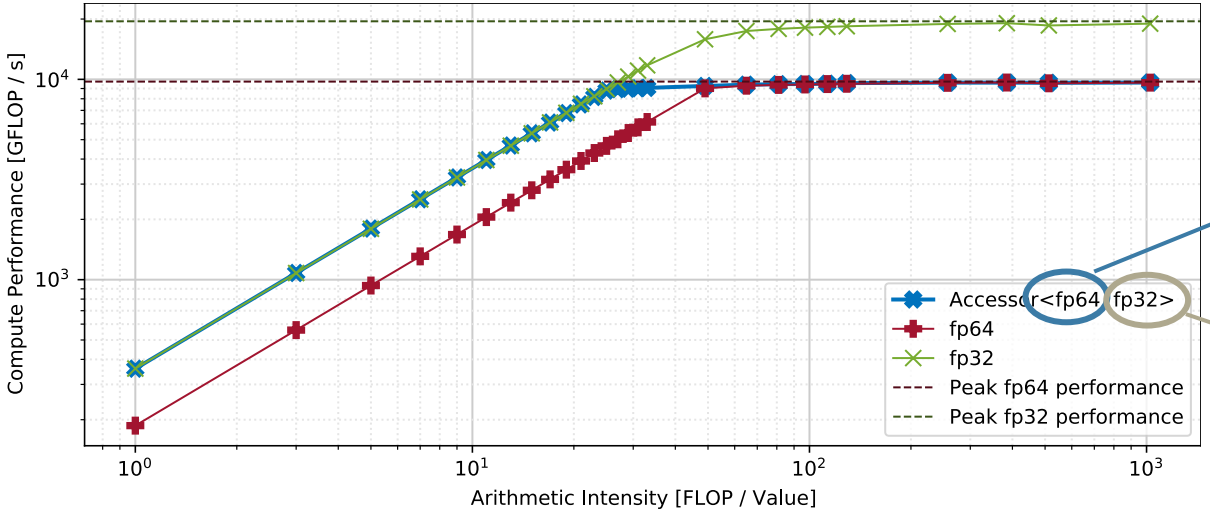
	Functionality	OMP	CUDA	HIP	DPC++
Basic	SpMV	✓	✓	✓	✓
	SpMM	✓	✓	✓	✓
	SpGeMM	✓	✓	✓	✓
Krylov solvers	BiCG	✓	✓	✓	✓
	BiCGSTAB	✓	✓	✓	✓
	CG	✓	✓	✓	✓
	CGS	✓	✓	✓	✓
	GMRES	✓	✓	✓	✓
	IDR	✓	✓	✓	✓
Preconditioners	(Block-) Jacobi	✓	✓	✓	✓
	ILU/IC		✓	✓	✓
	Parallel ILU/IC	✓	✓	✓	✓
	Parallel ILUT/ICT	✓	✓	✓	✓
	Sparse Approximate Inverse	✓	✓	✓	✓
Batched	Batched BiCGSTAB	✓	✓	✓	
	Batched CG	✓	✓	✓	
	Batched GMRES	✓	✓	✓	
	Batched ILU	✓	✓	✓	✓
	Batched ISAI	✓	✓	✓	✓
	Batched Jacobi	✓	✓	✓	✓
AMG	AMG preconditioner	✓	✓	✓	✓
	AMG solver	✓	✓	✓	✓
	Parallel Graph Match	✓	✓	✓	✓
Sparse direct	Symbolic Cholesky	✓	✓	✓	✓
	Numeric Cholesky	UNDER DEVELOPMENT			
	Symbolic LU	UNDER DEVELOPMENT			
	Numeric LU	✓	✓	✓	
	Sparse TRSV	✓	✓	✓	
Utilities	On-Device Matrix Assembly	✓	✓	✓	✓
	MC64/RCM reordering	✓			
	Wrapping user data	✓			
	Logging	✓			
	PAPI counters	✓			

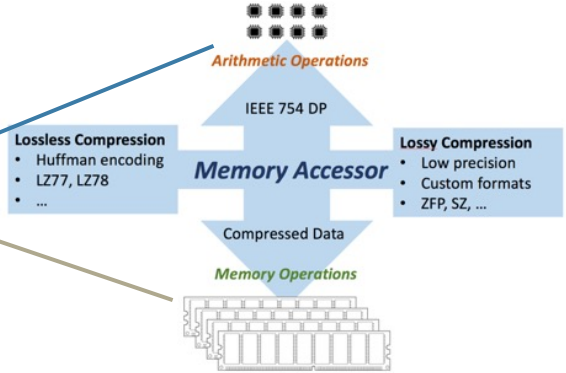


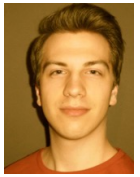


ICL INNOVATIVE COMPUTING LABORATORY https://github.com/ginkgo-project/ginkgo

Memory Accessor for NVIDIA A100 GPU







T. Grützmacher