Accelerating Numerical Software Libraries with Multi-Precision Algorithms

IDEAS Webinar

Wednesday, May 13, 2020

Hartwig Anzt and Piotr Luszczek

Who are we?

- **Hartwig Anzt**
  - Mathematical library developer;
  - Lead of the ECP Multiprecision effort;

Ginkgo

MAGMA

ECP
EXASCALE COMPUTING PROJECT

- **Piotr Luszczek**:
  - Performance engineer and developer for multiple numerical libraries and benchmarks;
  - Member of the xSDK4ECP project;

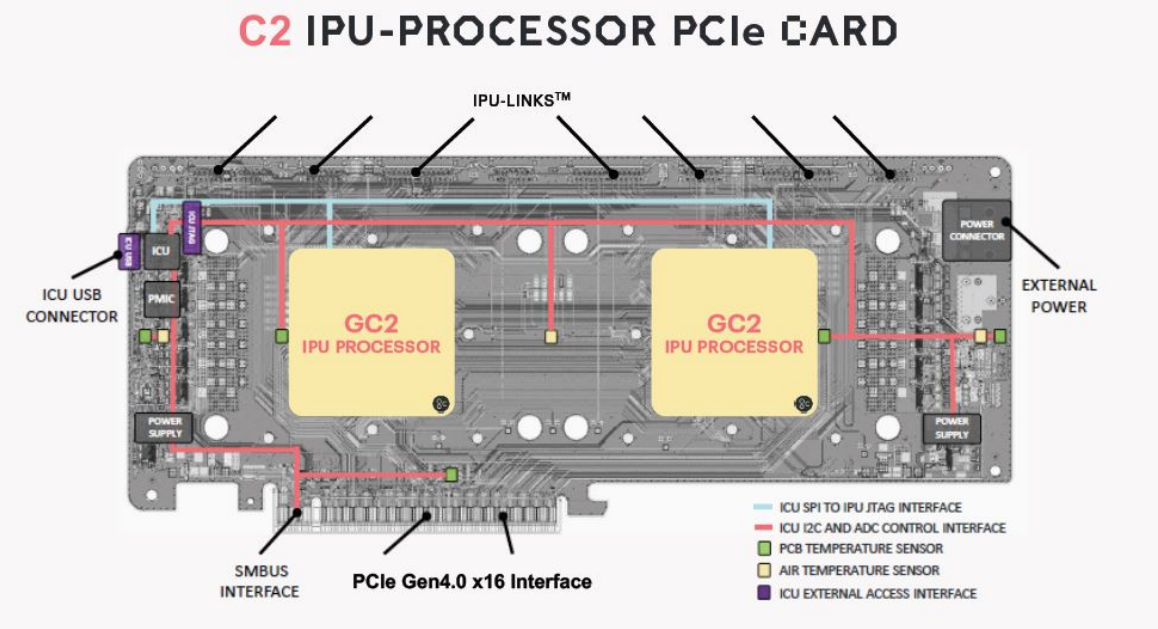
HPCG

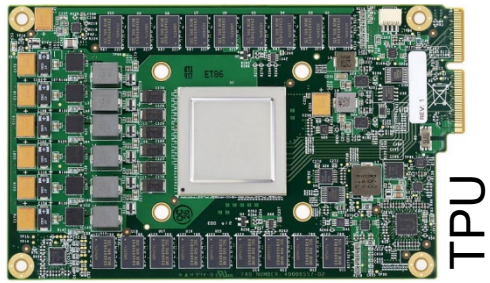
HPL

MAGMA

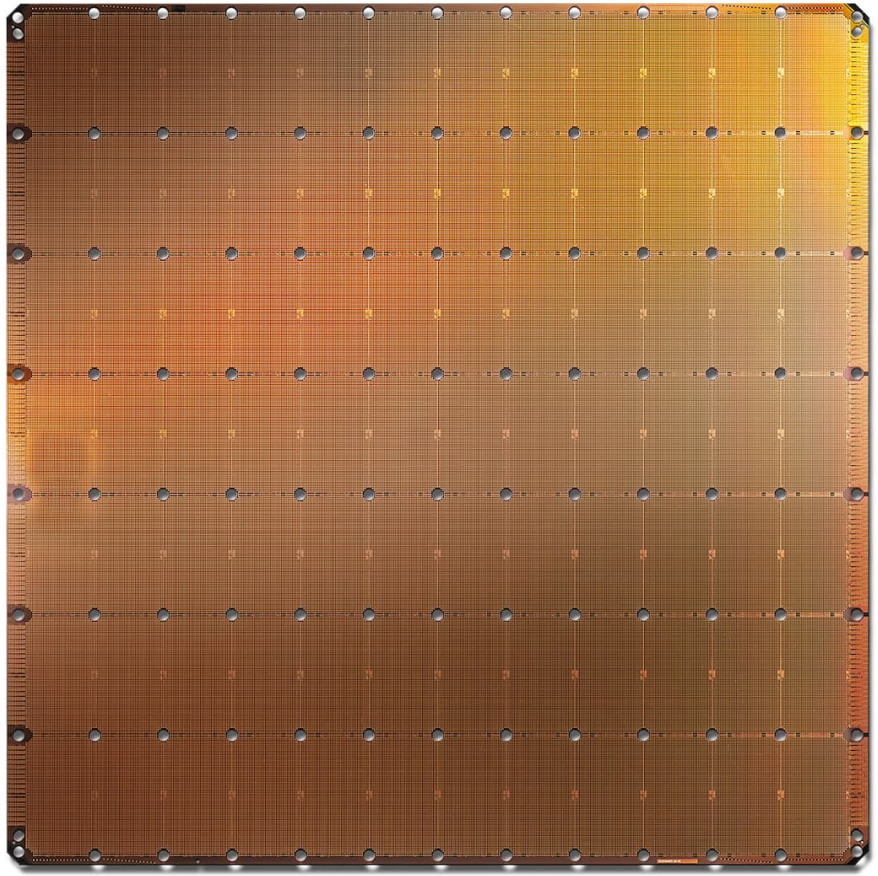
PLASMA

Modern Hardware: Beyond Traditional IEEE Floating point





TPU



Cerebras WSE

```
fmac [z]=[z], [w], a
      3D    3D   3D
```

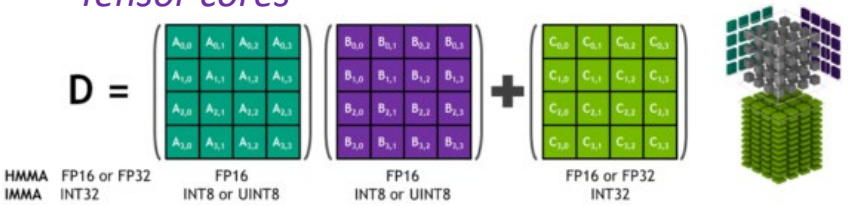
V100

Floating point formats and performance on GPUs

	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020
NVIDIA GPU generation			Tesla		Fermi			Kepler		Maxwell		Pascal		Volta	
Rel. compute performance			1 : 8		1 : 8			1 : 24		1 : 32		1 : 2 : 4		1 : 2 : 16*	
Rel. memory performance			1 : 2		1 : 2			1 : 2		1 : 2		1 : 2 : 4		1 : 2 : 4	

double : single : half

*Tensor cores



Floating point formats and performance on GPUs

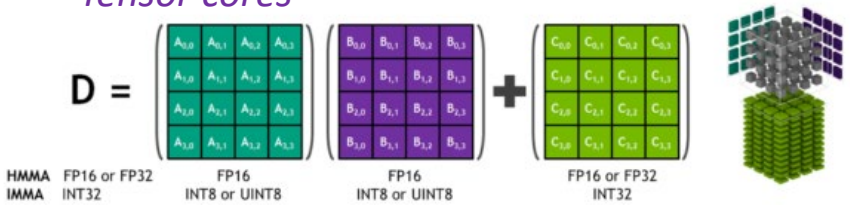
	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020
NVIDIA GPU generation			Tesla		Fermi			Kepler		Maxwell		Pascal		Volta	
Rel. compute performance			1 : 8		1 : 8			1 : 24		1 : 32		1 : 2 : 4		1 : 2 : 16*	
Rel. memory performance			1 : 2		1 : 2			1 : 2		1 : 2		1 : 2 : 4		1 : 2 : 4	

double : single : half

For **compute-bound applications**, the performance gains from using lower precision **depend on the architecture**.

Up to 16x for FP16 on Volta, up to 32x for FP32 on Maxwell.

**Tensor cores*

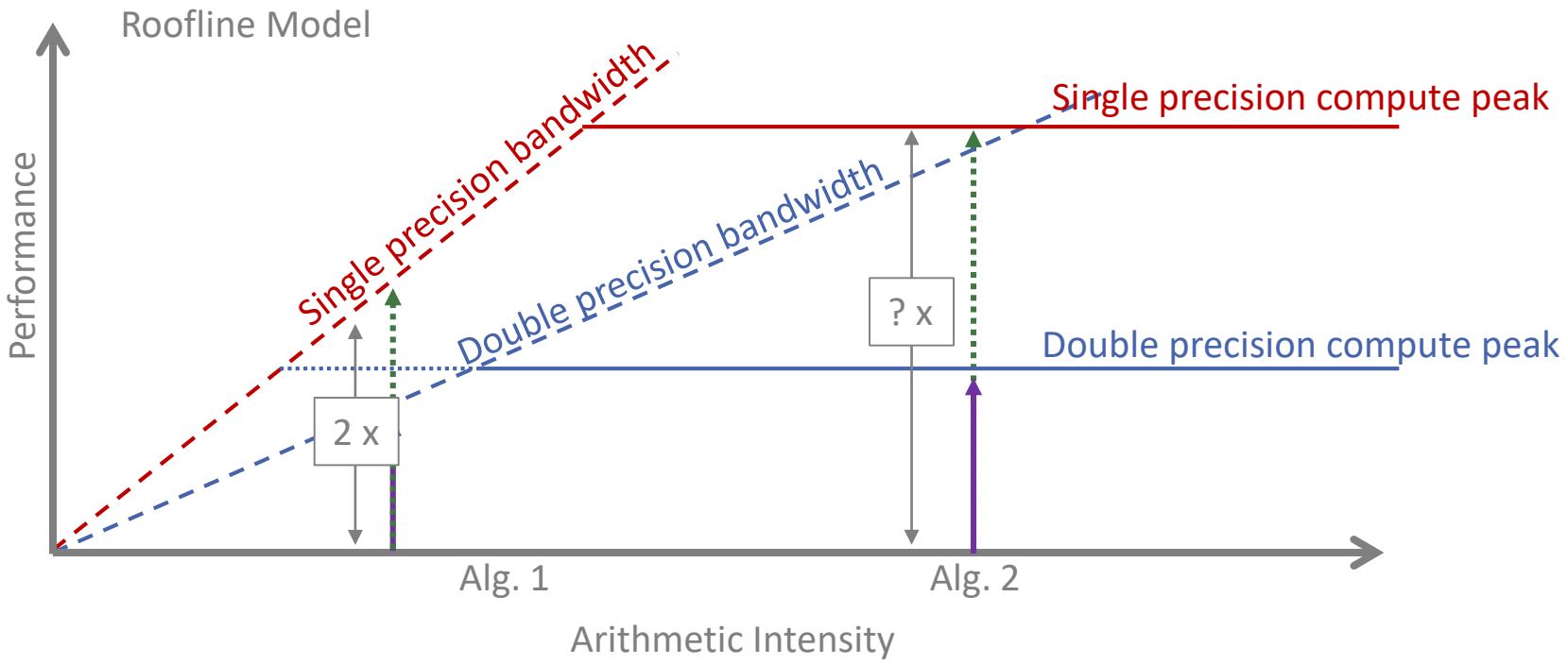


For **memory-bound applications**, the performance gains from using lower precision are **architecture-independent** and correspond to the floating point format complexity (#bits).

Generally, 2x for FP32, 4x for FP16.

Take-Away I

- Performance of **compute-bound** algorithms depends on **format support of hardware**.
- Performance of **memory-bound** algorithms scales **hardware-independent** with **inverse of format complexity**.



IEEE 754 Floating Point Formats

Broadly speaking….

- The length of the **exponent** determines the **range** of the values that can be represented;
- The length of the **significand** determines how **accurate** values can be represented;

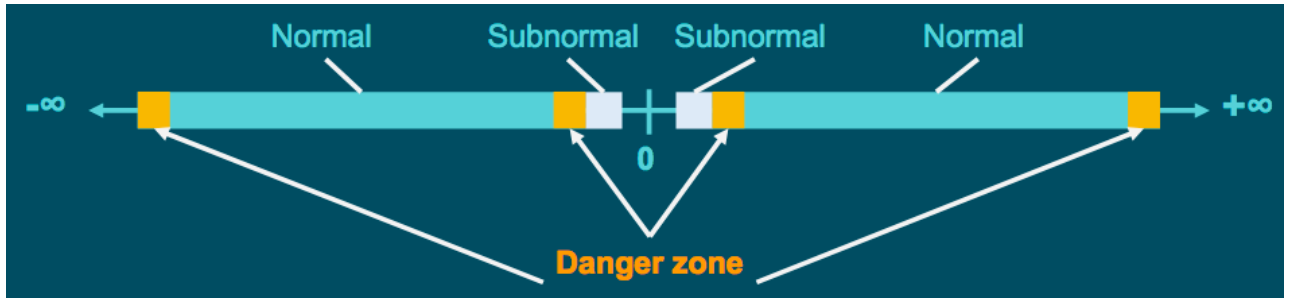
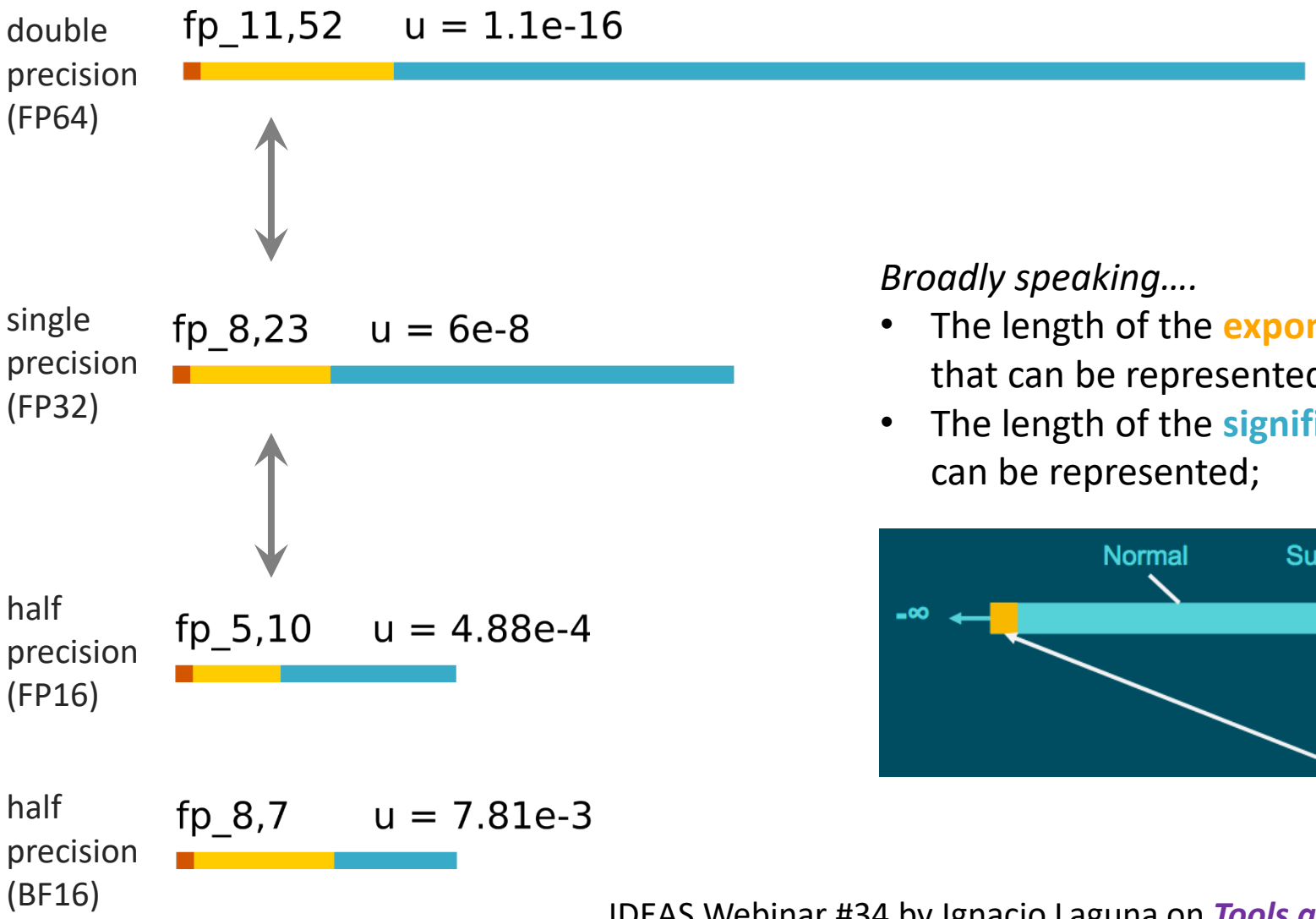


Figure courtesy of Ignacio Laguna, LLNL

IDEAS Webinar #34 by Ignacio Laguna on ***Tools and Techniques for Floating-Point Analysis***

IEEE 754 Floating Point Formats



Broadly speaking….

- The length of the **exponent** determines the **range** of the values that can be represented;
- The length of the **significand** determines how **accurate** values can be represented;

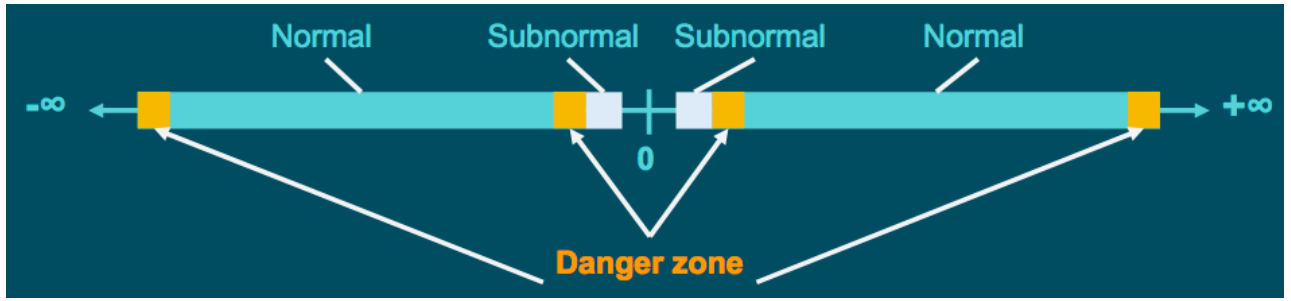


Figure courtesy of Ignacio Laguna, LLNL

IDEAS Webinar #34 by Ignacio Laguna on ***Tools and Techniques for Floating-Point Analysis***

Floating point formats and accuracy

- The length of the **exponent** determines the **range** of the values that can be represented;
- The length of the **significand** determines how **accurate** values can be represented;

- *Rounding effects accumulate over a sequence of computations;*

Worst case: $fl\left(\sum_{i=1}^{n} x_i\right) = \sum_{i=1}^{n} x_i + n \cdot u$ with u being the unit round off.

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

Floating point formats and accuracy

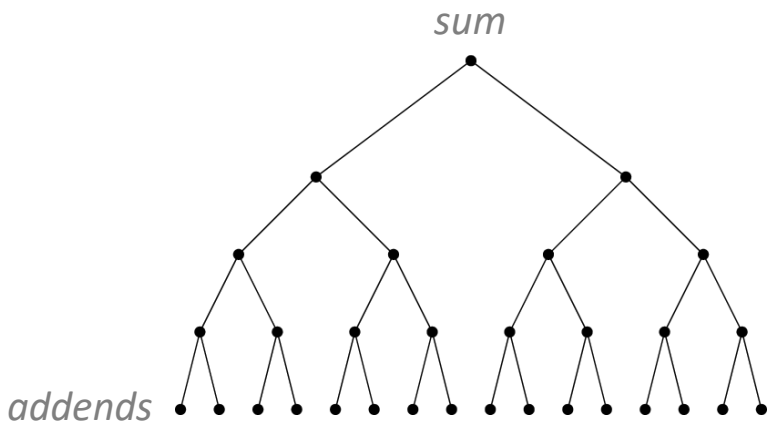
- The length of the **exponent** determines the **range** of the values that can be represented;
- The length of the **significand** determines how **accurate** values can be represented;

- *Rounding effects accumulate over a sequence of computations;*

Worst case: $fl\left(\sum_{i=1}^{n} x_i\right) = \sum_{i=1}^{n} x_i + n \cdot u$ with u being the unit round off.

In reality:

- *Stochastic effects reduce the impact of rounding;*
- *Parallel systems compute sums using blocking techniques (tree-based sum computation);*



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

Floating point formats and accuracy

- The length of the **exponent** determines the **range** of the values that can be represented;
- The length of the **significand** determines how **accurate** values can be represented;

- *Rounding effects accumulate over a sequence of computations;*

Let us focus on linear systems of the form $Ax = b$.

- The conditioning of a linear system reflects how sensitive the solution x is with regard to changes in the right-hand side b.

- Rounding in the arithmetic operations of a linear solver equivalent to perturbations of the right-hand-side.

- Rule of thumb:

relative residual accuracy = **(unit round-off)** * **(linear system's condition number)**

$10^{-6} = 10^{-16} * 10^{10}$

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

Floating point formats and accuracy

Linear System Ax=b with cond(A) $\approx 10^4$

Experiments based on the Ginkgo library https://ginkgo-project.github.io/ginkgo/examples/adaptiveprecision-blockjacobi/adaptiveprecision-blockjacobi.cpp

Floating point formats and accuracy

Linear System Ax=b with cond(A) ≈ 10^4

Double Precision

```
Initial residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
111.127
Final residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
5.0775e-10
CG iteration count:     1231
CG execution time [ms]: 140.038
```

Rel. Residual ~10^{-12}

relative residual accuracy = (unit round-off) * (linear system's condition number)

Experiments based on the Ginkgo library https://ginkgo-project.github.io/ginkgo/examples/adaptiveprecision-blockjacobi/adaptiveprecision-blockjacobi.cpp

Floating point formats and accuracy

Linear System Ax=b with cond(A) ≈ 10^4

Double Precision

```
Initial residual norm sqrt(r^T r):
%%MatrixMarket matrix array real g
1 1
111.127
Final residual norm sqrt(r^T r):
%%MatrixMarket matrix array real g
1 1
5.0775e-10
CG iteration count:     1231
CG execution time [ms]: 140.038
```

Rel. Residual ~10^{-12}

Exploring floating point formats in sparse iterative solvers:

Ginkgo

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

...

- ValueType = double;
+ ValueType = float;

...

relative residual accuracy = (unit round-off) * (linear system's condition number)

Experiments based on the Ginkgo library https://ginkgo-project.github.io/ginkgo/examples/adaptiveprecision-blockjacobi/adaptiveprecision-blockjacobi.cpp

Floating point formats and accuracy

Linear System Ax=b with cond(A) ≈ 10^4

Double Precision

```
Initial residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
111.127
Final residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
5.0775e-10
CG iteration count:     1231
CG execution time [ms]: 140.038
```

Rel. Residual ~10^{-12}

Single Precision

```
Initial residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
111.127
Final residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
0.179829
CG iteration count:     1234
CG execution time [ms]: 127.152
```

Rel. Residual ~10^{-4}

relative residual accuracy = **(unit round-off)** * **(linear system's condition number)**

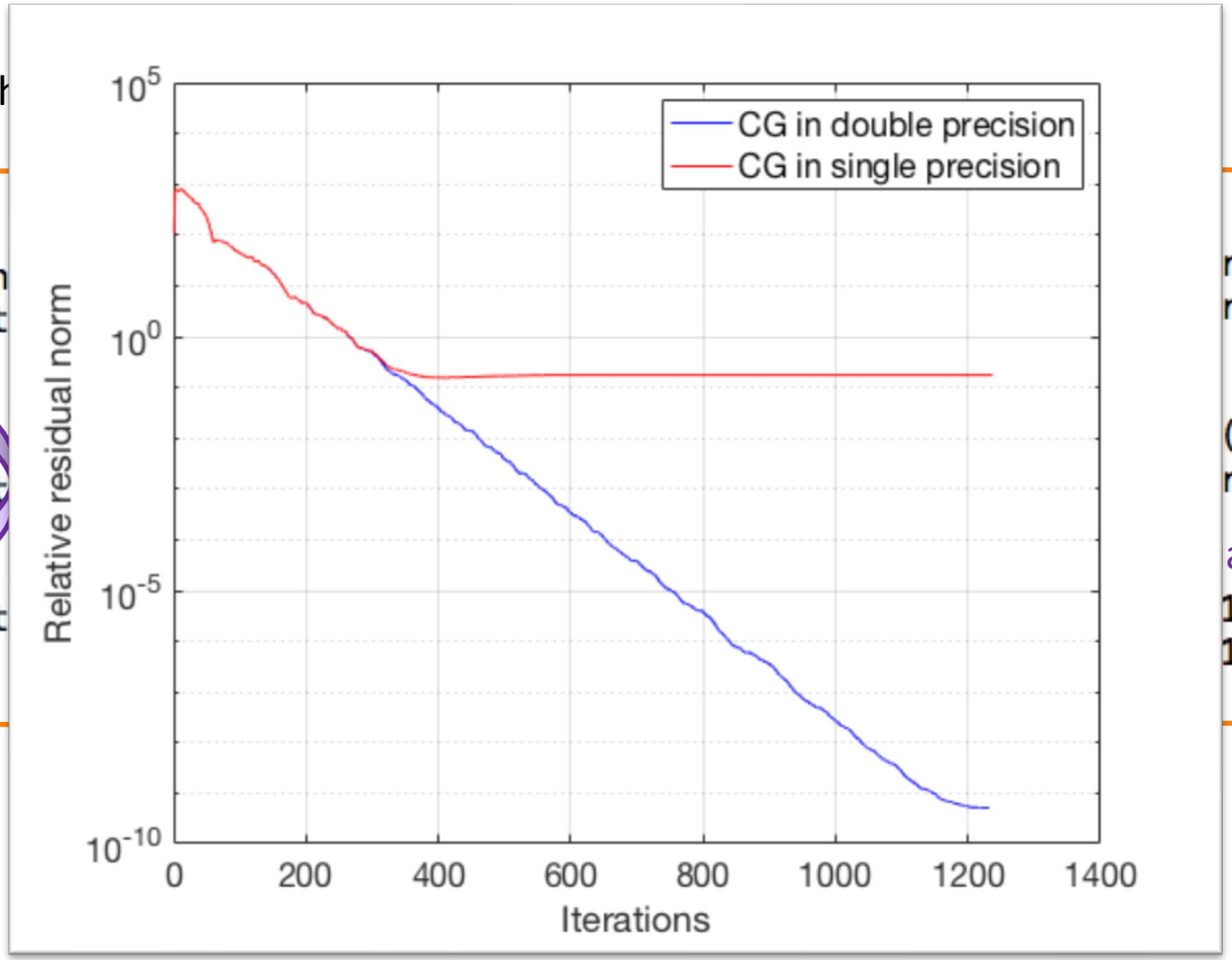
Experiments based on the Ginkgo library https://ginkgo-project.github.io/ginkgo/examples/adaptiveprecision-blockjacobi/adaptiveprecision-blockjacobi.cpp

Floating point formats and accuracy

Linear System Ax=b with

Double Precision

```
Initial residual n
%%MatrixMarket mat
1 1
111.127
Final residual no
%%MatrixMarket mat
1 1
5.0775e-10
CG iteration count
CG execution time
```




```
rt(r^T r):
ray real general


(r^T r):
ray real general
```

al ~10^{-4}

```
1234
127.152
```

Experiments based on the Ginkgo library https://ginkgo-project.github.io/ginkgo/examples/adaptiveprecision-blockjacobi/adaptiveprecision-blockjacobi.cpp

Floating point formats and accuracy

Linear System Ax=b with cond(A) $\approx 10^4$

Double Precision

```
Initial residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
111.127
Final residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
5.0775e-10
CG iteration count:      1231
CG execution time [ms]: 140.038
```

Single Precision

```
Initial residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
111.127
Final residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
0.179829
CG iteration count:      1234
CG execution time [ms]: 127.152
```

Single Precision is 10% faster!

Experiments based on the Ginkgo library https://ginkgo-project.github.io/ginkgo/examples/adaptiveprecision-blockjacobi/adaptiveprecision-blockjacobi.cpp

Floating point formats and accuracy

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

apache2 from SuiteSparse

Double Precision

```
Initial residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
1390.67
Final residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
4.53915e-06
CG iteration count:     6460
CG execution time [ms]: 2992.91
```

Rel. Residual ~10^{-9}

Single Precision

```
Initial residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
1390.67
Final residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
1588.77
CG iteration count:     8887
CG execution time [ms]: 2972.46
```

No improvement

Experiments based on the Ginkgo library https://ginkgo-project.github.io/ginkgo/examples/adaptiveprecision-blockjacobi/adaptiveprecision-blockjacobi.cpp

Take-Away II

- **Relative residual accuracy** = **(unit round-off)** * **(linear system's condition number)**
- For **ill-conditioned problems**, we need **high precision** to provide high accuracy results.
- Only if the problem is **well-conditioned**, and a **low-accuracy solution is acceptable**, we can use a **low precision format** throughout the complete solution process.
- **Templating** the precision format (i.e. ValueType) allows to **quickly switch between formats.**
  - *C++ very powerful in this respect*
  - *Use production-ready libraries templating the precision: Ginkgo, Kokkos Kernels, Trilinos, etc.*

Low precision for solving ill-conditioned problems

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

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

Low precision for solving ill-conditioned problems

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

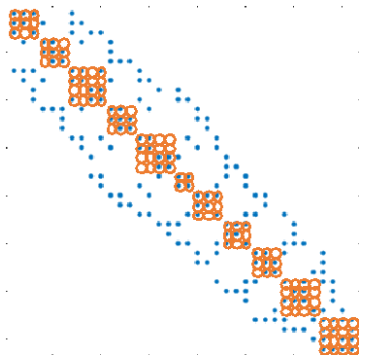
    and solve $Ax = b \quad \Leftrightarrow \quad P^{-1}Ax = P^{-1}b \quad \Leftrightarrow \quad \tilde{A}x = \tilde{b}\,.$
- **Why should we use a preconditioner P^{-1} in full** (high) **precision?**

Low precision for solving ill-conditioned problems

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

    and solve $Ax = b \quad \Leftrightarrow \quad P^{-1}Ax = P^{-1}b \quad \Leftrightarrow \quad \tilde{A}x = \tilde{b}\,.$
- **Why should we use a preconditioner P^{-1} in full (high) precision?**

- **Jacobi method** based on **diagonal scaling** $P = diag(A)$
- **Block-Jacobi** 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.



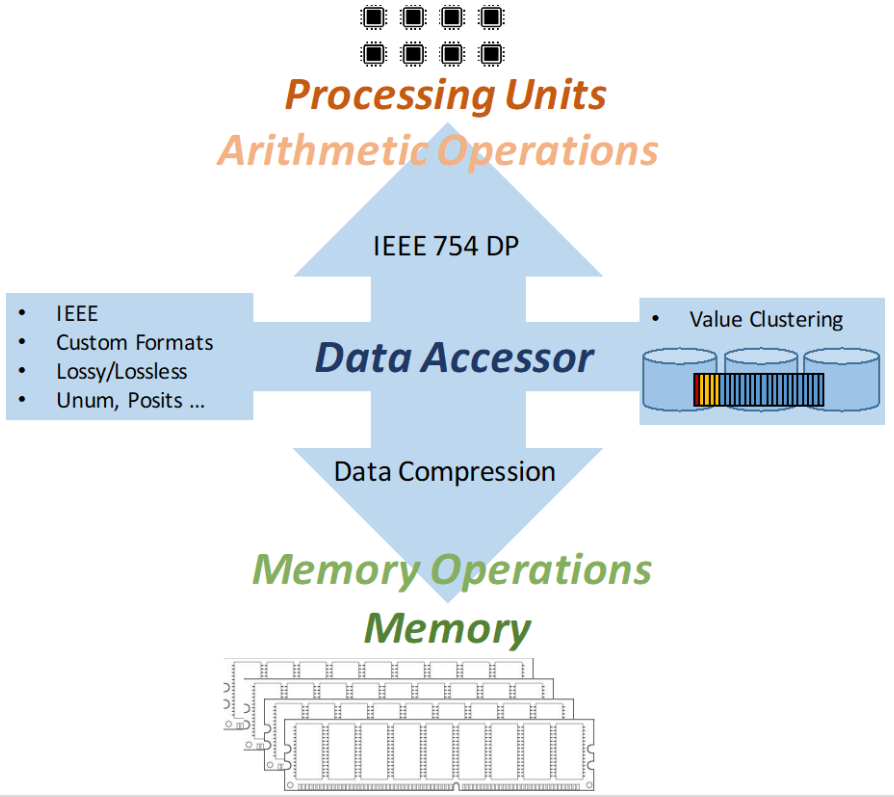
Low precision for solving ill-conditioned problems

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

    and solve $Ax = b \quad \Leftrightarrow \quad P^{-1}Ax = P^{-1}b \quad \Leftrightarrow \quad \tilde{A}x = \tilde{b}\,.$
- **Why should we use a preconditioner P^{-1} in full (high) precision?**

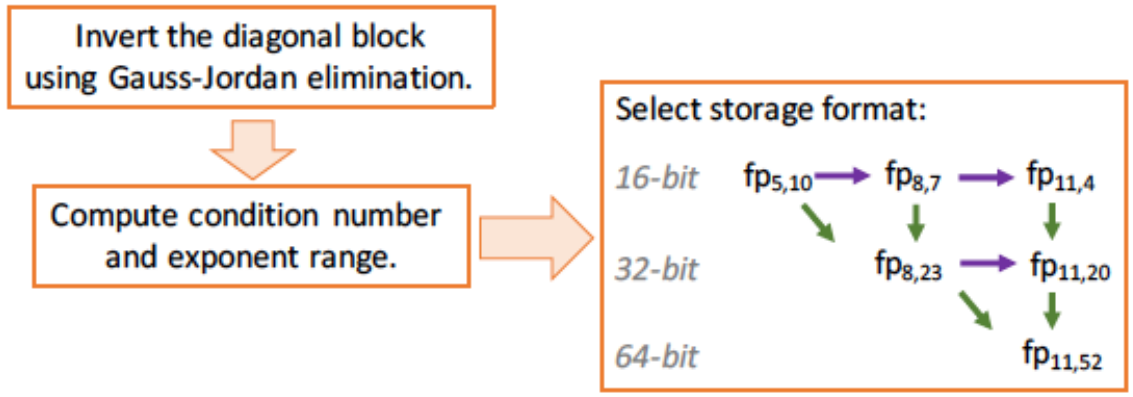
- **Jacobi method** based on **diagonal scaling** $P = diag(A)$
- **Block-Jacobi** is based on **block-diagonal scaling**: $P = diag_B(A)$
  - Each block corresponds to one (small) linear system.
    - *Larger* blocks typically improve convergence.
    - *Larger* blocks make block-Jacobi more expensive.

Idea: Store the inverted diagonal in low precision



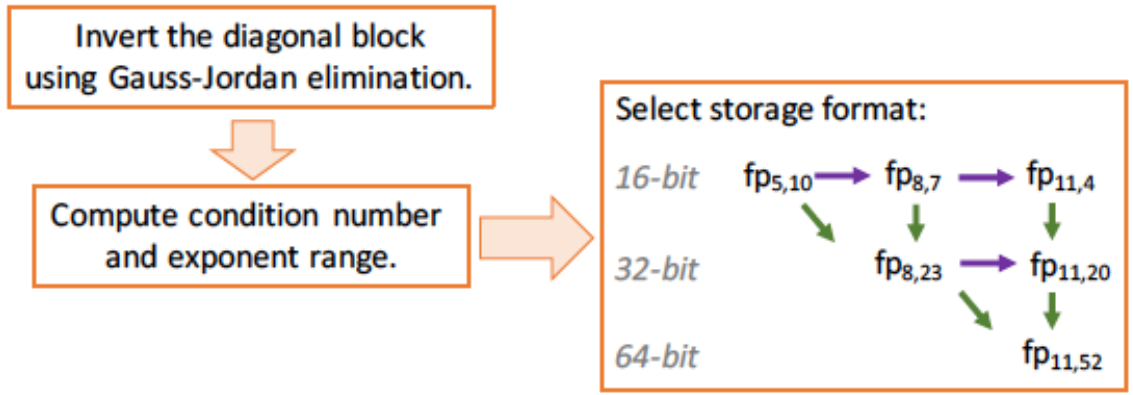
Adaptive Precision Preconditioning

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



Adaptive Precision Preconditioning

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



\+ **Regularity preserved;**

\+ Flexible in the accuracy preserved;

Preserving Preconditioner accuracy of 10^{-2}

\+ No flexible Krylov solver needed

(Preconditioner constant operator);

\+ Can handle non-spd problems (featuring pivoting);

\+ Can be used in any preconditionable solver;

\- **Overhead** of the **precision detection**

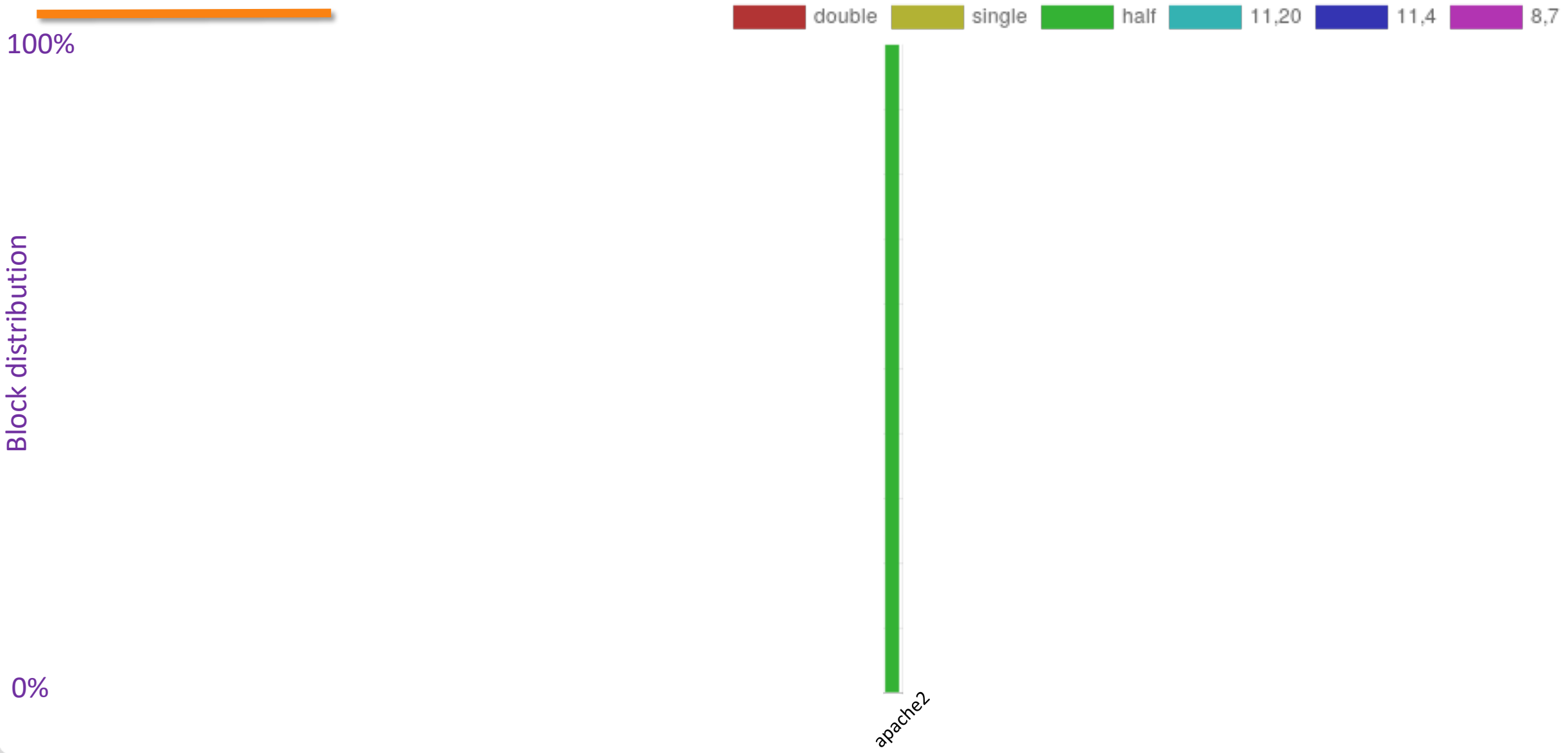
(condition number calculation);

\- **Overhead** from storing **precision information**

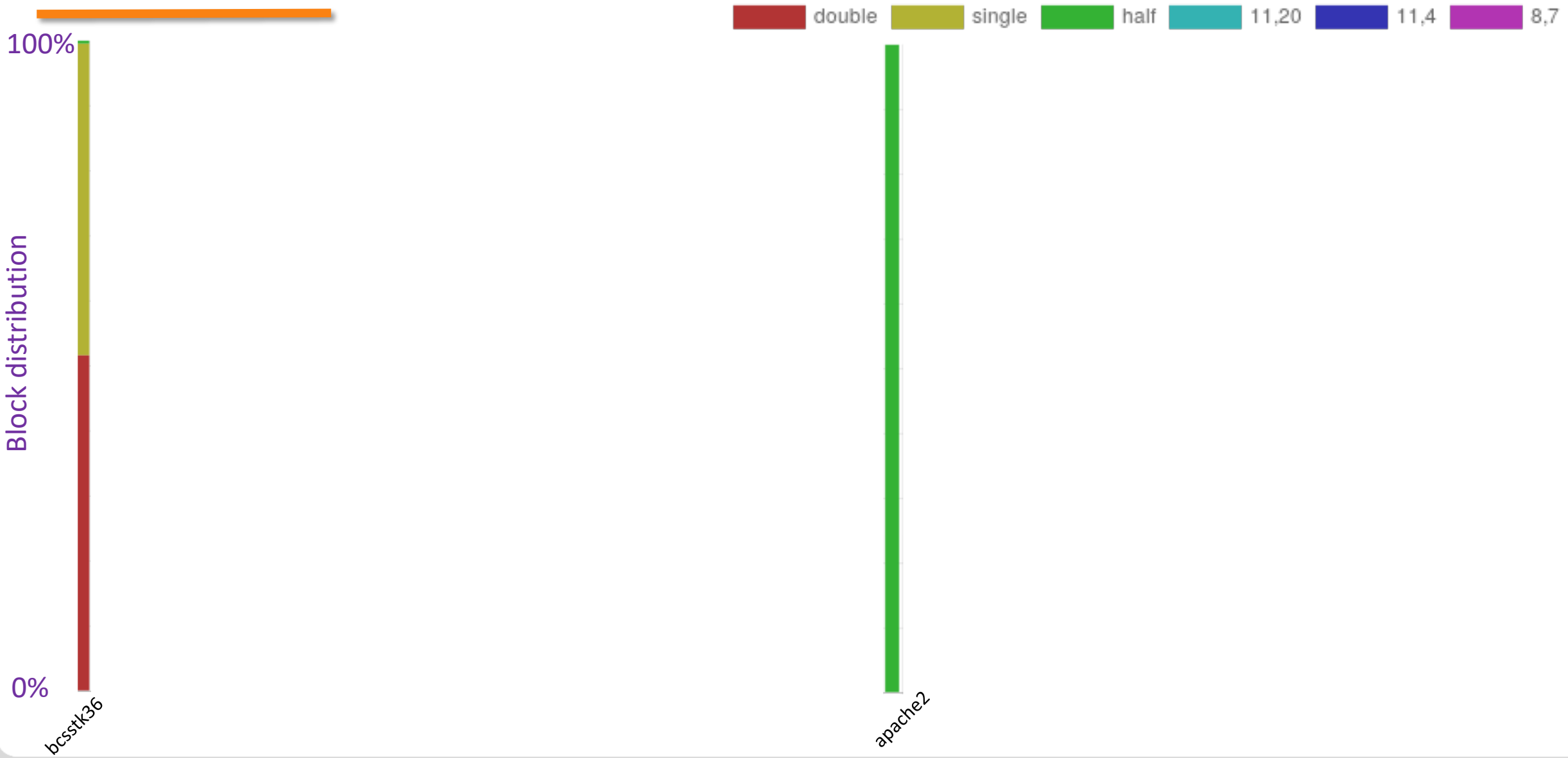
(need to additionally store/retrieve flag);

\- Speedups / preconditioner quality **problem-dependent**;

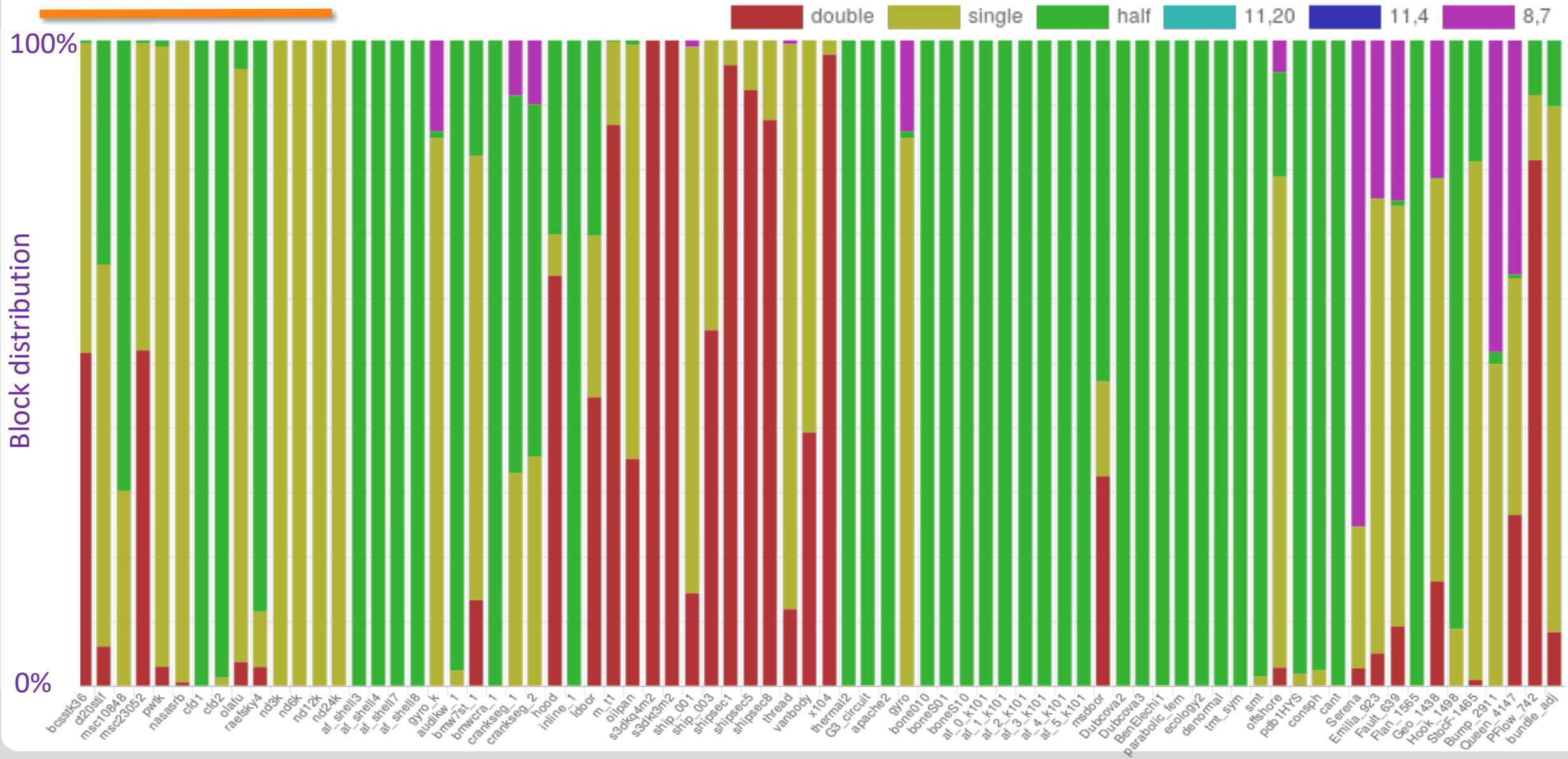
Adaptive Precision Preconditioning



Adaptive Precision Preconditioning



Adaptive Precision Preconditioning



Floating point formats and accuracy

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

apache2 from SuiteSparse

Double Precision + Double Preconditioner

```
Initial residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
1390.67
Final residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
3.97985e-06
CG iteration count:      4797
CG execution time [ms]: 2971.18
```

Rel. Residual ~10^{-9}

Double Precision + Mixed Precision Preconditioner

```
Initial residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
1390.67
Final residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
3.98574e-06
CG iteration count:      4794
CG execution time [ms]: 2568.1
```

Rel. Residual ~10^{-9}

16% runtime improvement

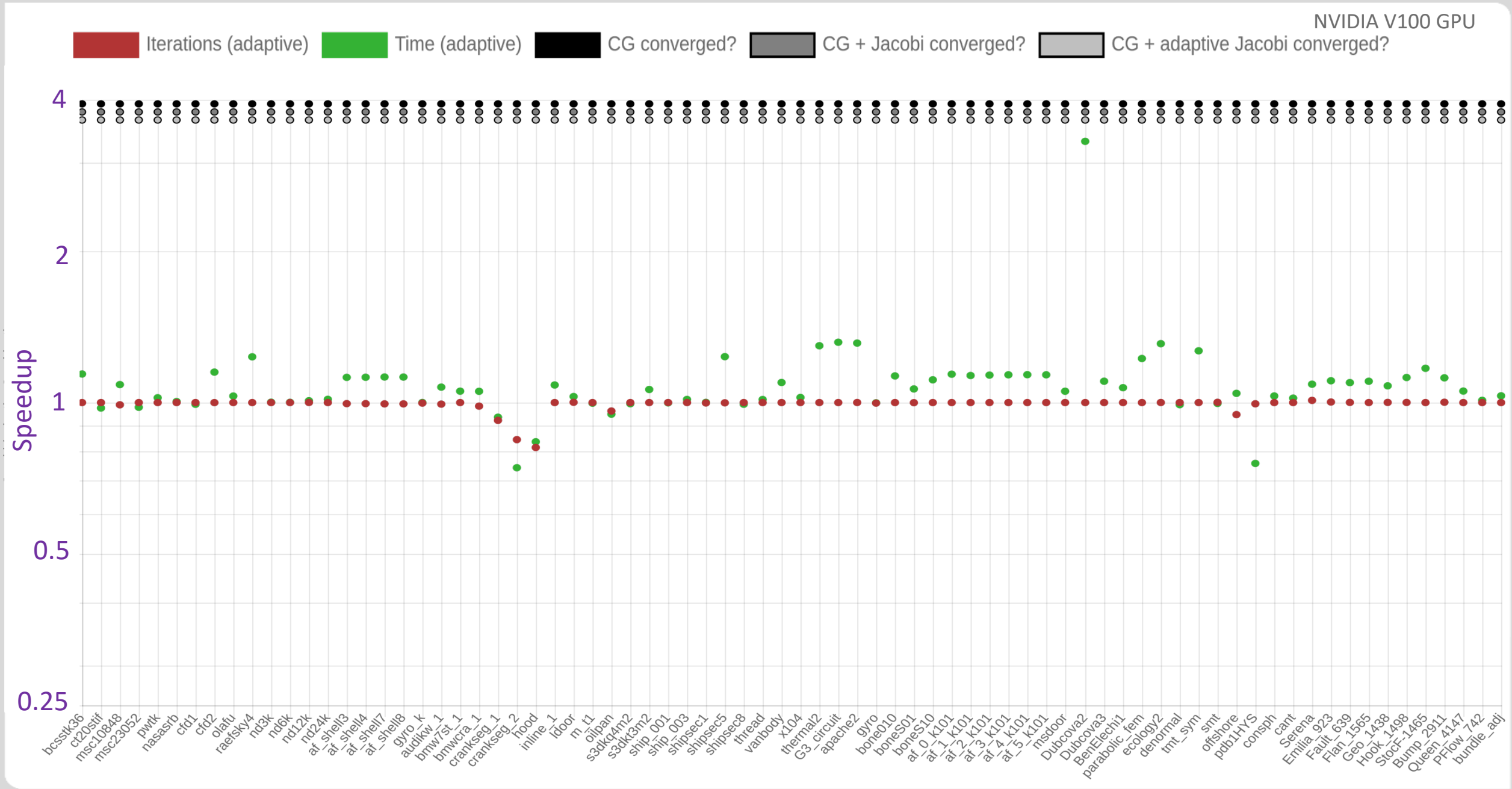
Experiments based on the Ginkgo library https://ginkgo-project.github.io/ginkgo/examples/adaptiveprecision-blockjacobi/adaptiveprecision-blockjacobi.cpp

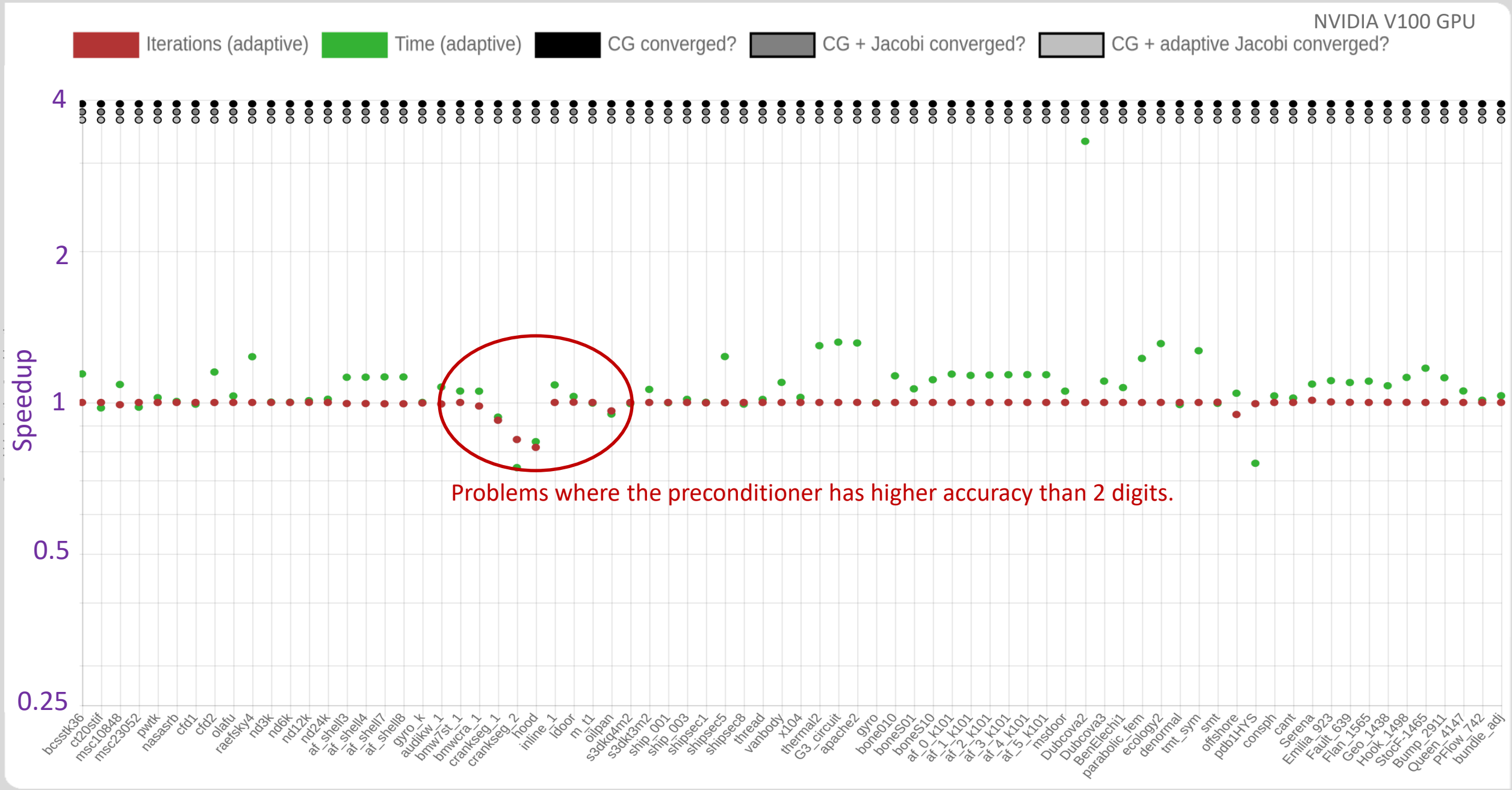
Floating point formats and accuracy

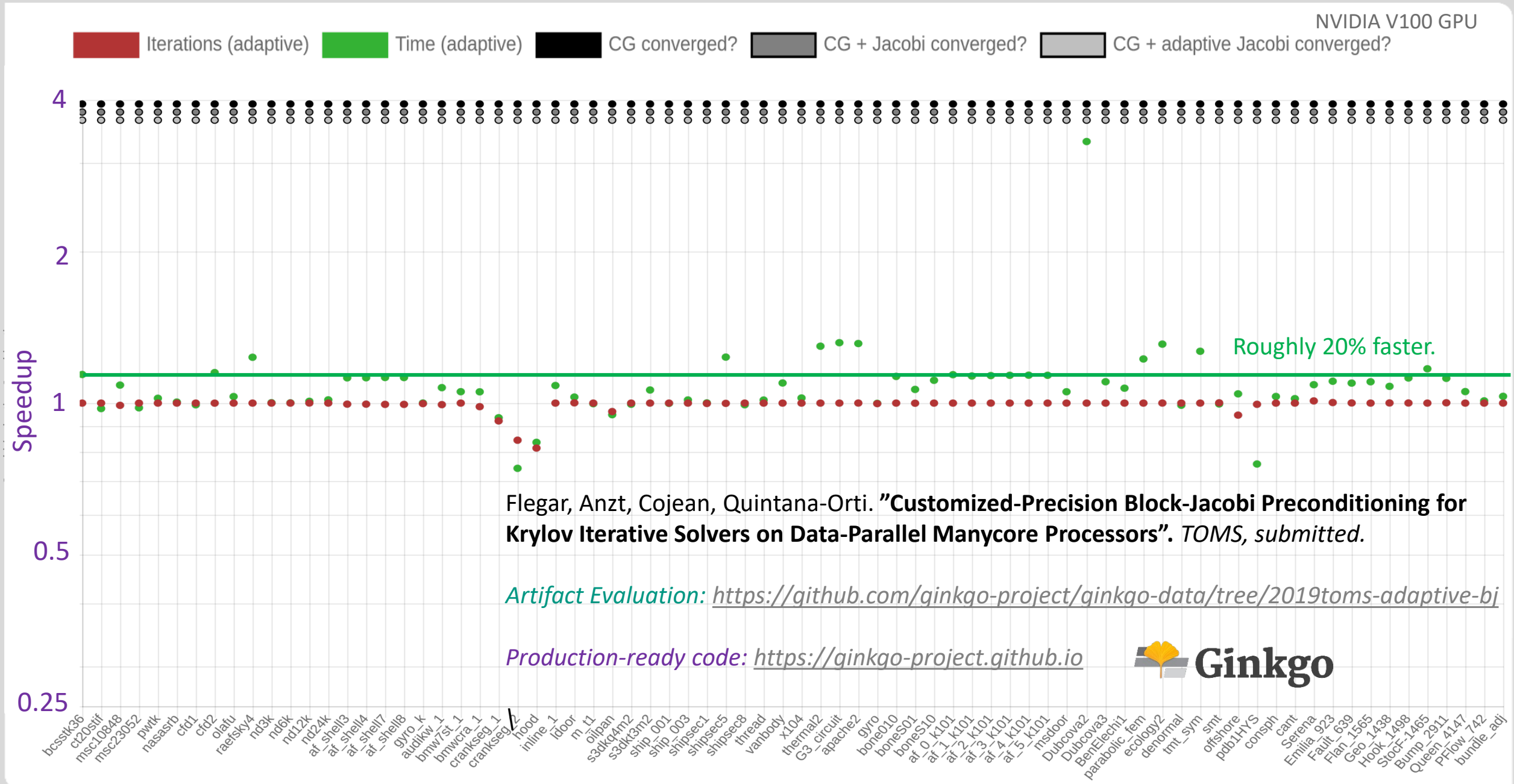
ginkgo/examples/adaptiveprecision-blockjacobi/adaptiveprecision-blockjacobi.cpp

```
…
        auto solver_gen =
            cg::build()
                .with_criteria(gko::share(iter_stop), gko::share(tol_stop))
                        .with_preconditioner(bj::build()
                    .with_max_block_size(16u)
                    .with_storage_optimization(
                        gko::precision_reduction::autodetect())
                    .on(exec))
                .on(exec);
…
```

Experiments based on the Ginkgo library https://ginkgo-project.github.io/ginkgo/examples/adaptiveprecision-blockjacobi/adaptiveprecision-blockjacobi.cpp


NVIDIA V100 GPU
Iterations (adaptive)
Time (adaptive)
CG converged?
CG + Jacobi converged?
CG + adaptive Jacobi converged?
Speedup
4
2
1
0.5
0.25
Problem
bcsstk36
ct20stif
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
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


NVIDIA V100 GPU
Iterations (adaptive)
Time (adaptive)
CG converged?
CG + Jacobi converged?
CG + adaptive Jacobi converged?
Speedup
4
2
1
0.5
0.25
Problems where the preconditioner has higher accuracy than 2 digits.
Problem
bcsstk36
ct20stif
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
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


NVIDIA V100 GPU
Iterations (adaptive)
Time (adaptive)
CG converged?
CG + Jacobi converged?
CG + adaptive Jacobi converged?
Speedup
4
2
1
0.5
0.25
Roughly 20% faster.
Flegar, Anzt, Cojean, Quintana-Orti. "Customized-Precision Block-Jacobi Preconditioning for Krylov Iterative Solvers on Data-Parallel Manycore Processors". TOMS, submitted.
Artifact Evaluation: https://github.com/ginkgo-project/ginkgo-data/tree/2019toms-adaptive-bj
Production-ready code: https://ginkgo-project.github.io
Ginkgo
Problem

Take-Away III

- **Low precision preconditioners** can be used to **accelerate iterative solvers**.
  - Preconditioners need to adapt their precision to numerical requirements.
  - The preconditioner precision determines how much accuracy is preserved.
- For memory-bound preconditioners, **decoupling the arithmetic precision from the memory precision** provides the **runtime savings** while preserving a **constant preconditioner**.
- **To increase the performance benefits, shift most of the work to the low precision preconditioner.**

Using a low precision solver as preconditioner

- **To increase the performance benefits, shift most of the work to the low precision preconditioner.**

- Use a **simple (cheap) iterative solver** in **high precision**
  and a **sophisticated (expensive) solver** in **low precision** as preconditioner.
  - Most of the work is done in low precision (fast).
  - The high precision outer solver ensures high quality of the solution.

- Popular example: Iterative Refinement

  For an approximate solution $x^{(k)}$, the residual computes as $r = b - Ax^{(k)}$.
  The exact solution for $Ax = b$ is $x = x^{(k)} + c$ where c is the solution of $Ac = r$.

Mixed Precision Iterative Refinement

Choose initial guess x	***high precision***
do {	
Compute r = b – Ax	***high precision***
Solve A * c = r	***low precision***
Update x = x + c	***high precision***
} while (||r||>tol)	***high precision***

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

Sri Pranesh's mixed precision Matlab suite:
https://github.com/SrikaraPranesh/Multi_precision_NLA_kernels

Mixed Precision Iterative Refinement using sparse iterative solvers

Linear System Ax=b with cond(A) ≈ 10^4

Double Precision Iterative Refinement

```
Initial residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
111.127
Final residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
7.16102e-11
MPIR iteration count:     18
MPIR execution time [ms]: 213.491
```

Rel. Residual ~10^{-14}

Mixed Precision Iterative Refinement

```
Initial residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
111.127
Final residual norm sqrt(r^T r):
%%MatrixMarket matrix array real general
1 1
7.41333e-11
MPIR iteration count:     18
MPIR execution time [ms]: 183.296
```

Rel. Residual ~10^{-14}

16% runtime improvement

Experiments based on the Ginkgo library https://ginkgo-project.github.io/ginkgo/examples/mixed-precision-ir/mixed-precision-ir.cpp

Mixed Precision Iterative Refinement using sparse iterative solvers

Some references:

Strzodka et al. **Pipelined Mixed Precision Algorithms on FPGAs for Fast and Accurate PDE Solvers from Low Precision Components**, IEEE Symposium on Field-Programmable Custom Computing Machines, 2006.

Goedekke et al. **Performance and accuracy of hardware-oriented native-, emulated- and mixed-precision solvers in FEM simulations**, International Journal of Parallel, Emergent and Distributed Systems, 2007.

Buratti et al. **Using Mixed Precision for Sparse Matrix Computations to Enhance the Performance while Achieving 64-bit Accuracy**, ACM TOMS 2008.

Baboulin et al. **Accelerating scientific computations with mixed precision algorithms**, CPC, 2009.

Anzt et al. **Mixed precision iterative refinement methods for linear systems: Convergence analysis based on Krylov subspace methods**, PARA 2010.

…

For sparse iterative methods, the benefits relate to the bandwidth savings.

Mixed Precision Iterative Refinement using sparse iterative solvers

Some references:

Strzodka et al. **Pipelined Mixed Precision Algorithms on FPGAs for Fast and Accurate PDE Solvers from Low Precision Components**, IEEE Symposium on Field-Programmable Custom Computing Machines, 2006.

Goedekke et al. **Performance and accuracy of hardware-oriented native-, emulated- and mixed-precision solvers in FEM simulations**, International Journal of Parallel, Emergent and Distributed Systems, 2007.

Buratti et al. **Using Mixed Precision for Sparse Matrix Computations to Enhance the Performance while Achieving 64-bit Accuracy**, ACM TOMS 2008.

Baboulin et al. **Accelerating scientific computations with mixed precision algorithms**, CPC, 2009.

Anzt et al. **Mixed precision iterative refinement methods for linear systems: Convergence analysis based on Krylov subspace methods**, PARA 2010.

…

For sparse iterative methods, the benefits relate to the bandwidth savings.

Some software packages allowing for mixed precision sparse linear system solves:

Ginkgo hypre MAGMA PETSc TRILINOS

Refinement symmetric eigenvalues and eigenvectors

Low precision inputs (must be accurate in lower precision)

1: **Input:** $A = A^T \in \mathbb{R}^{n\times n}$, $\widehat{X} \in \mathbb{R}^{n\times \ell}$, $1 \le \ell \le n$

2: **Output:** $X' \in \mathbb{R}^{n\times \ell}$, $\widetilde{D} = \mathrm{diag}\left(\widetilde{\lambda}_i\right) \in \mathbb{R}^{\ell\times\ell}$, $\widetilde{E} \in \mathbb{R}^{\ell\times\ell}$, $\omega \in \mathbb{R}$

3: **function** $\left[X', \widetilde{D}, \widetilde{E}, \omega\right] \leftarrow$ RefSyEv$(A, \widehat{X})$

4: $\quad R \leftarrow \mathbb{I}_n - \widehat{X}^T\widehat{X}$

5: $\quad S \leftarrow \widehat{X}^T A \widehat{X}$

6: $\quad \widehat{\lambda}_i \leftarrow s_{ii}/(1 - r_{ii})$ for $i = 1, \ldots, \ell$ ▷ Compute approximate eigenvalues.

7: $\quad \widetilde{D} \leftarrow \mathrm{diag}\left(\widetilde{\lambda}_i\right)$

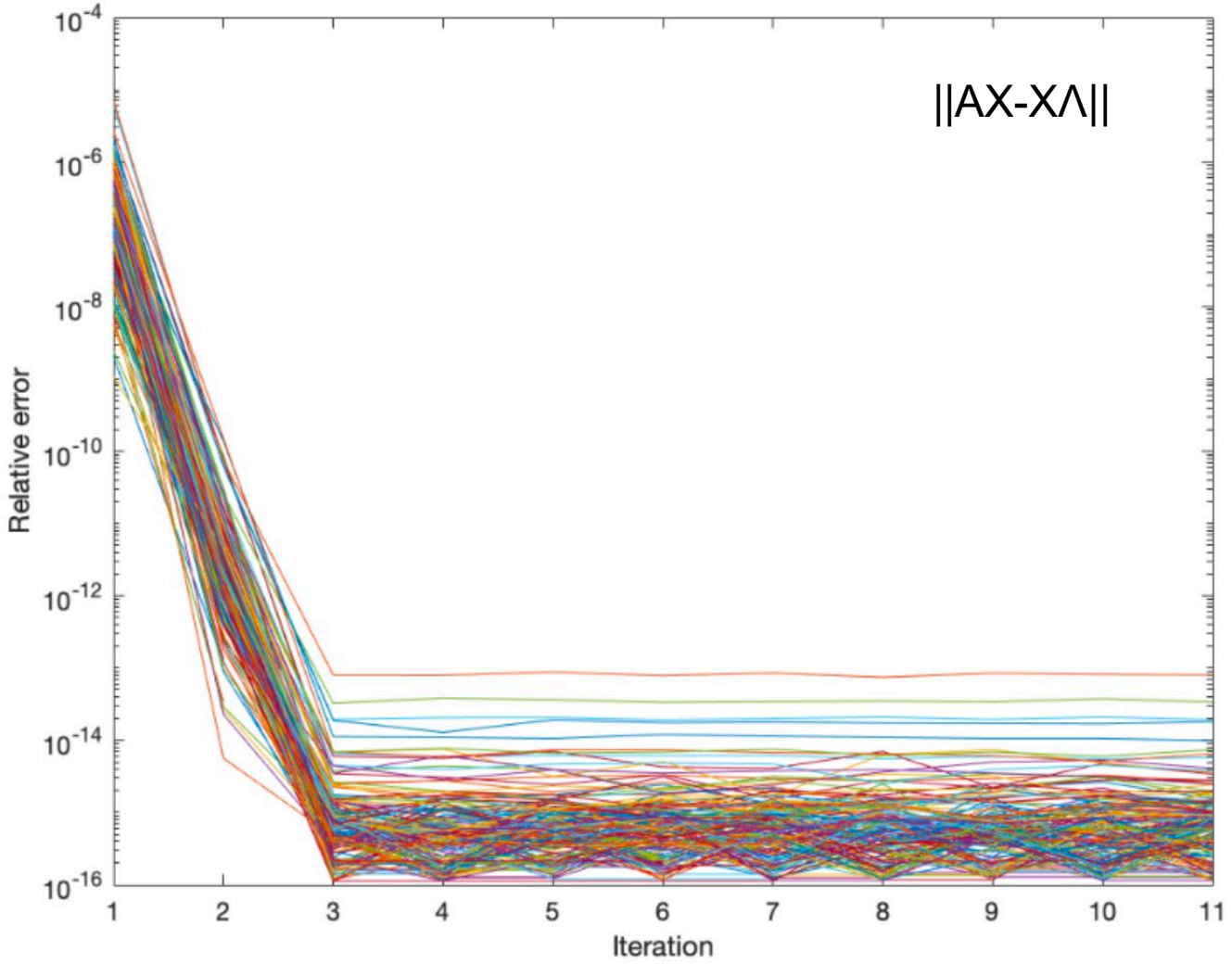
8: $\quad \omega \leftarrow 2\left(\left\|S - \widetilde{D}\right\|_2 + \|A\|_2\|R\|_2\right)$

9: $\quad e_{ij} \leftarrow \begin{cases} \frac{s_{ij} + \widetilde{\lambda}_j r_{ij}}{\widetilde{\lambda}_j - \widetilde{\lambda}_i} & \text{if } \left|\widetilde{\lambda}_i - \widetilde{\lambda}_j\right| > \omega \\ r_{ij}/2 & \text{otherwise} \end{cases}$ for $1 \le i, j \le \ell$ ▷ Compute the entries of the refinement matrix $\widetilde{E}$.

10: $\quad X' \leftarrow \widehat{X} + \widehat{X}\widetilde{E}$ ▷ Update $\widehat{X}$ by $\widehat{X}(\mathbb{I}_n + \widetilde{E})$

11: **end function**

Convergence for each of 200 eigenvalues (backward error)



Complexity Analysis

Input: $A = A^T \in \mathbb{R}^{n\times n}$, $\widehat{X} \in \mathbb{R}^{n\times \ell}$, $1 \le \ell \le n$

Output: $X' \in \mathbb{R}^{n\times \ell}$, $\widetilde{D} = \mathrm{diag}\left(\widetilde{\lambda}_i\right) \in \mathbb{R}^{\ell\times\ell}$, $\widetilde{E} \in \mathbb{R}^{\ell\times\ell}$, $\omega \in \mathbb{R}$

function $\left[X', \widetilde{D}, \widetilde{E}, \omega\right] \leftarrow$ RefSyEv$(A, \widehat{X})$

$R \leftarrow \mathbb{I}_n - \widehat{X}^T\widehat{X}$

$S \leftarrow \widehat{X}^T A\widehat{X}$

$\widehat{\lambda}_i \leftarrow s_{ii}/(1 - r_{ii})$ for $i = 1, \ldots, \ell$ ▷ Compute approximate eigenvalues.

$\widetilde{D} \leftarrow \mathrm{diag}\left(\widetilde{\lambda}_i\right)$

$\omega \leftarrow 2\left(\left\|S - \widetilde{D}\right\|_2 + \|A\|_2\|R\|_2\right)$

$$e_{ij} \leftarrow \begin{cases} \frac{s_{ij} + \widetilde{\lambda}_j r_{ij}}{\widetilde{\lambda}_j - \widetilde{\lambda}_i} & \text{if } \left|\widetilde{\lambda}_i - \widetilde{\lambda}_j\right| > \omega \\ r_{ij}/2 & \text{otherwise} \end{cases} \quad \text{for } 1 \le i, j \le \ell$$ ▷ Compute the entries of the refinement matrix $\widetilde{E}$.

$X' \leftarrow \widehat{X} + \widehat{X}\widetilde{E}$ ▷ Update $\widehat{X}$ by $\widehat{X}(\mathbb{I}_n + \widetilde{E})$

end function

4 matrix multiplies in higher precision $7n^3$ per iteration

Take-Away IV

- **Mixed precision iterative refinement** is a powerful strategy to accelerate **linear solves;**
  - Iterative inner solver e.g. for sparse systems;
  - Direct inner solver e.g. for dense systems;

- Mixed precision iterative refinement can also be used for **eigenvalue problems;**
  - Low precision eigenvector approximations as input;
  - Convergence in high precision after 3-4 IR steps;

- The performance benefits **depend on the problem and hardware capabilities;**

References and further reading

Higham. **Accuracy and stability of numerical algorithms**, SIAM, 2002.

Anzt et al. **Adaptive precision in block-Jacobi preconditioning for iterative sparse linear system solvers**, Concurrency and Computation: Practice and Experience, 2019.

Anzt et al. **Toward a modular precision ecosystem for high-performance computing**, IJHPCA, 2019.

Strzodka et al. **Pipelined Mixed Precision Algorithms on FPGAs for Fast and Accurate PDE Solvers from Low Precision Components**, IEEE Symposium on Field-Programmable Custom Computing Machines, 2006.

Goedekke et al. **Performance and accuracy of hardware-oriented native-, emulated- and mixed-precision solvers in FEM simulations**, International Journal of Parallel, Emergent and Distributed Systems, 2007.

Buratti et al. **Using Mixed Precision for Sparse Matrix Computations to Enhance the Performance while Achieving 64-bit Accuracy**, ACM TOMS 2008.

Baboulin et al. **Accelerating scientific computations with mixed precision algorithms**, CPC, 2009.

Prikopa et al. **On Mixed Precision Iterative Refinement for Eigenvalue Problems, Procedia Computer Science, 2013.**

Ogita et al. **Iterative refinement for symmetric eigenvalue decomposition II: clustered eigenvalues**, Japan Journal of Industrial and Applied Mathematics, 2018.