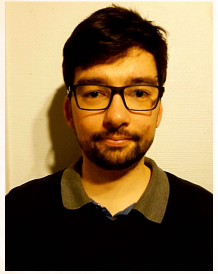
Adaptive Precision Preconditioning

9th JLESC Workshop

April 15th-17th, 2019 | Knoxville, TN

Hartwig Anzt, Terry Cojean, Goran Flegar, Thomas Grűtzmacher, Pratik Nayak, Enrique S. Quintana-Orti
Steinbuch Centre for Computing (SCC)



Terry Cojean

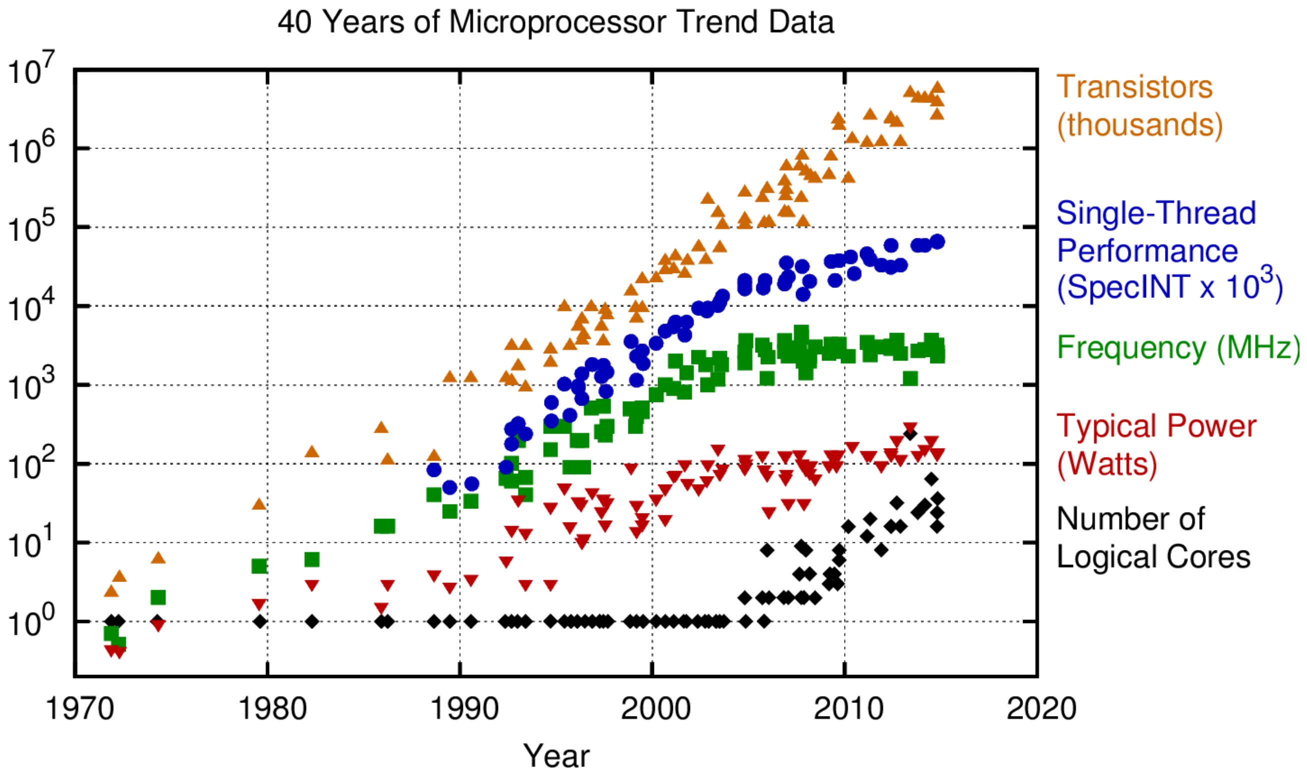
Goran Flegar

Thomas Grützmacher

Pratik Nayak

Tobias Ribizel

Where do we stand?

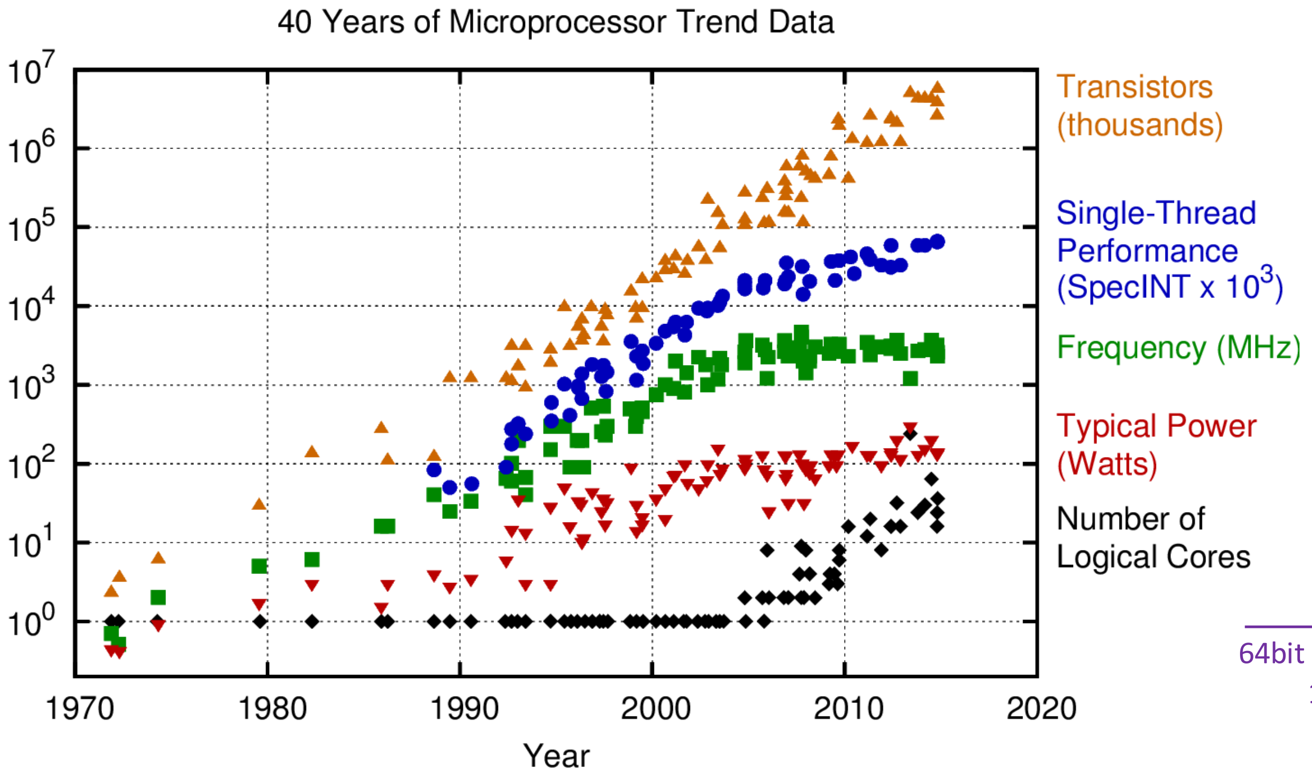


Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

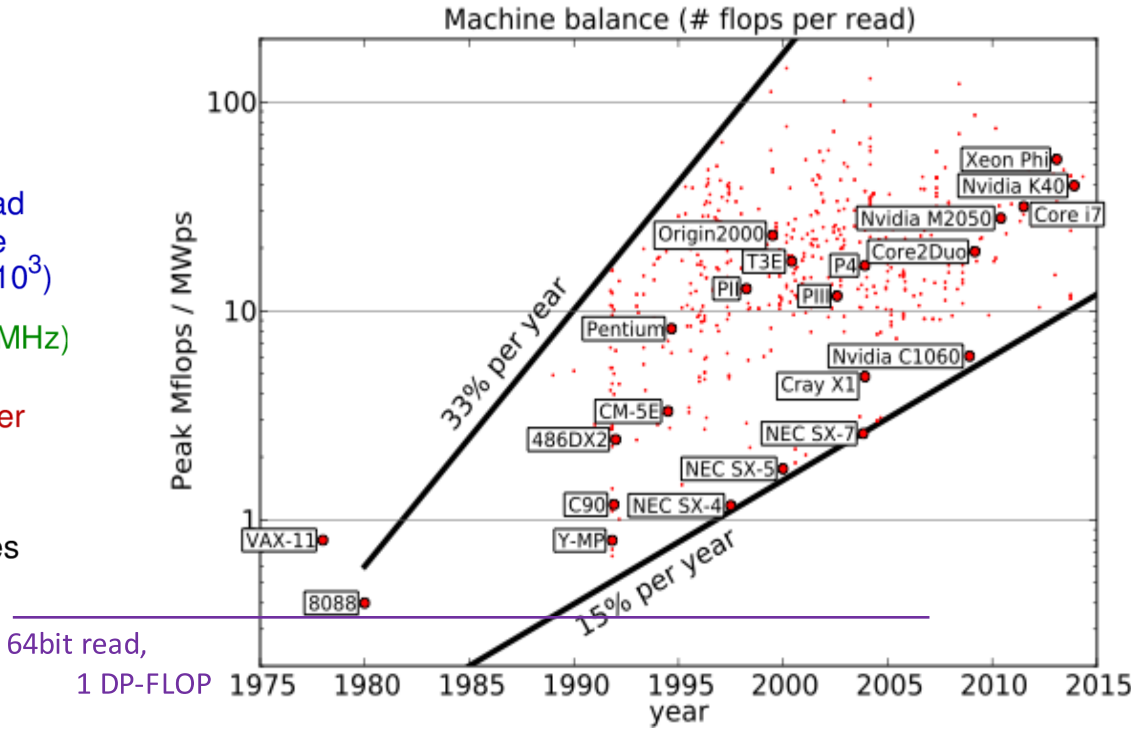
- Explosion in core count.

"Parallelism needed – synchronization kills performance."

Where do we stand?



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp



John D. McCalpin (TACC)

- Explosion in core count.

"Parallelism needed – synchronization kills performance."

- Compute power (#FLOPs) grows much faster than bandwidth.

"Operations are free, memory access is what counts."

Where do we stand?

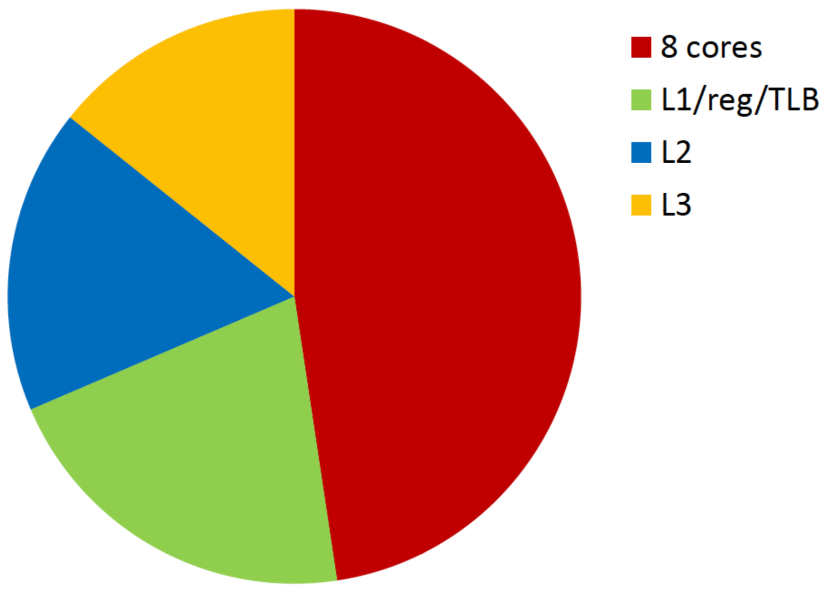
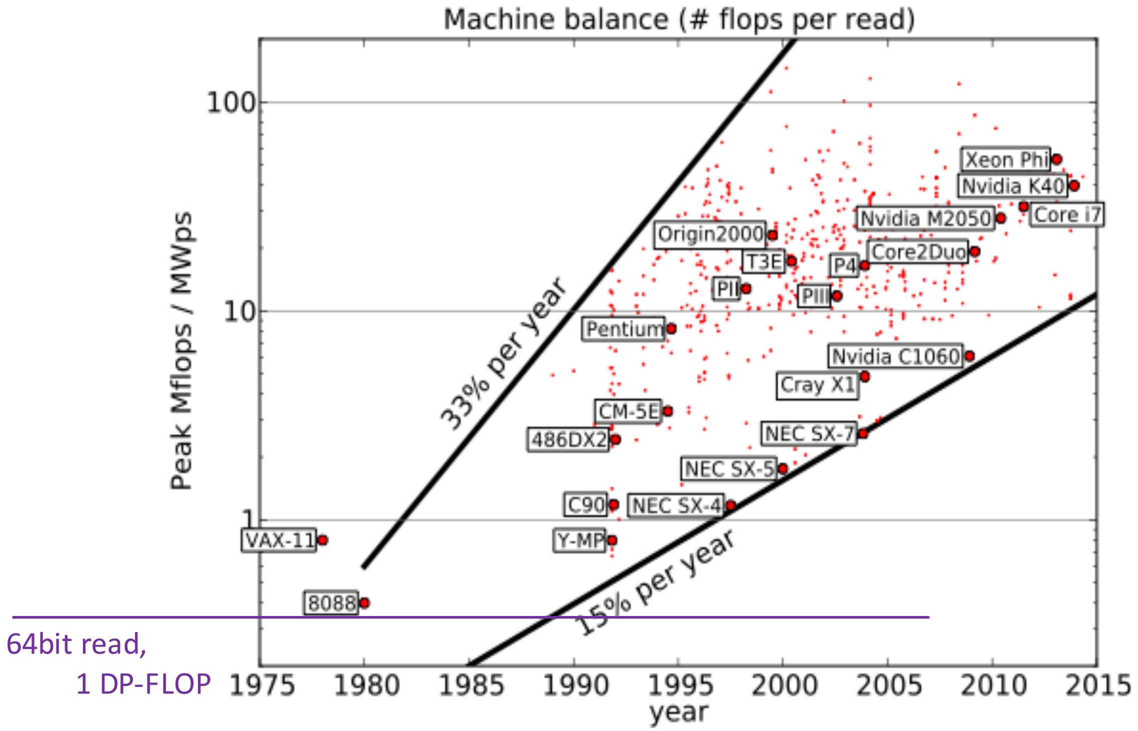


Figure 1.1.7: Power breakdown of an 8 core server chip.

Mark Horowitz (2014): **Computing's energy problem (and what we can do about it)**



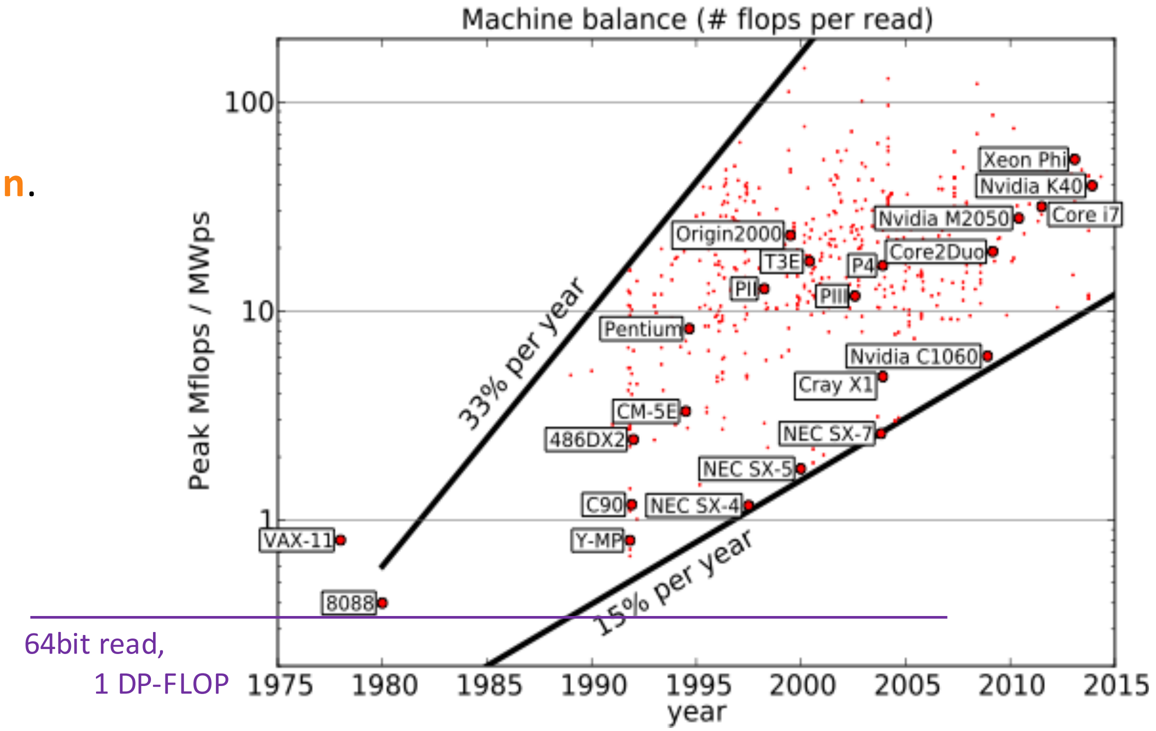
John D. McCalpin (TACC)

- Compute power (#FLOPs) grows much faster than bandwidth.

"Operations are free, memory access is what counts."

Algorithms reflecting hardware evolution

- The arithmetic operations should use high precision formats natively supported by hardware.
- **Data access** should be as cheap as possible, **reduced precision**.



John D. McCalpin (TACC)

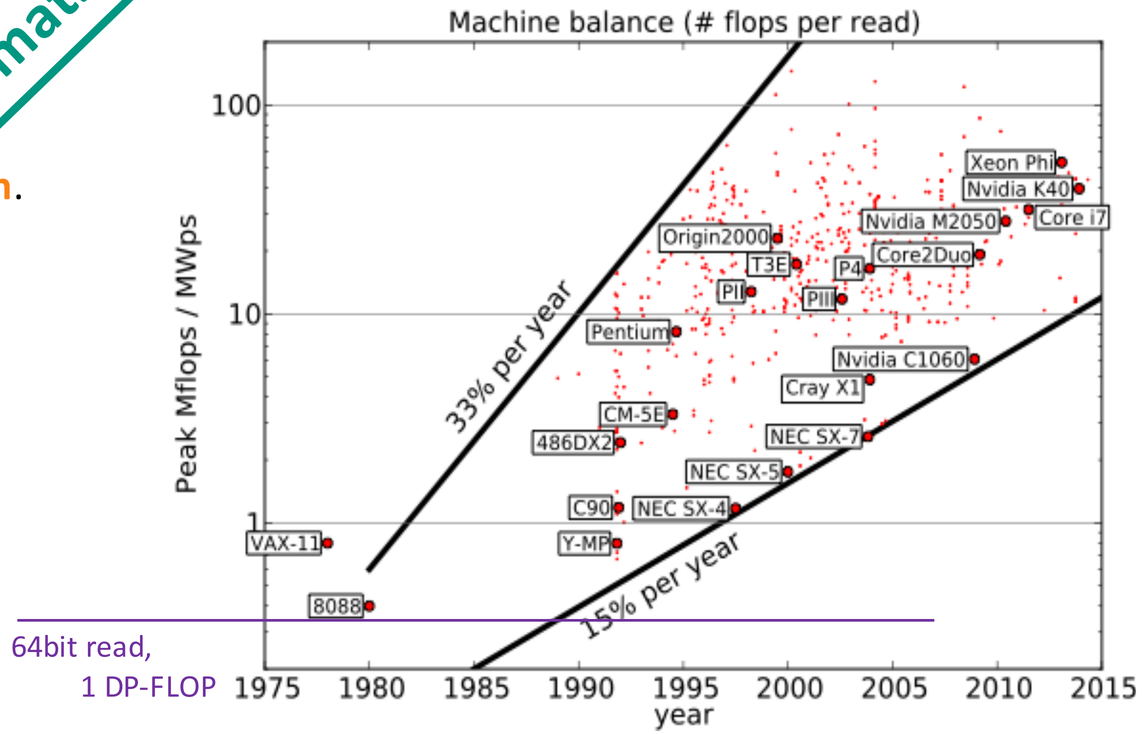
- Compute power (#FLOPs) grows much faster than bandwidth.

"Operations are free, memory access is what counts."

Algorithms reflecting hardware evolution

- The arithmetic operations should use high precision forma[illegible] natively supported by hardware.
- **Data access** should be as cheap as possible, **reduc[illegible]sion**.

Radically decouple storage format from arithmetic format.



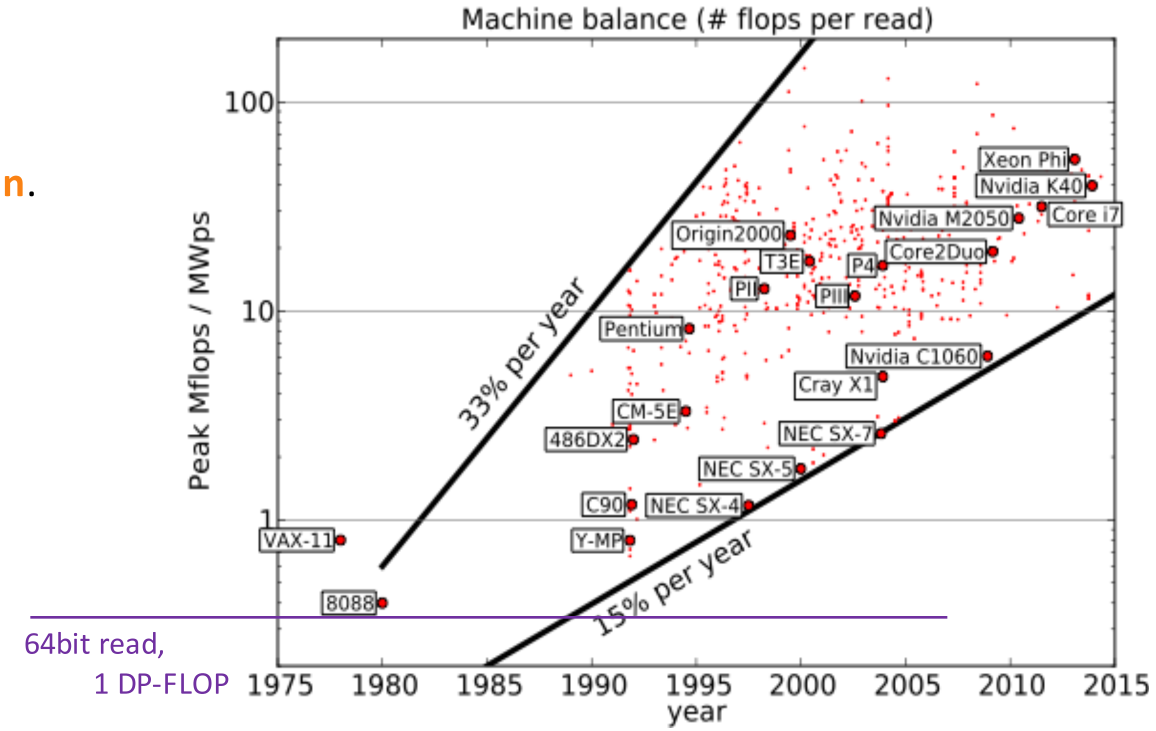
John D. McCalpin (TACC)

- Compute power (#FLOPs) grows much faster than bandwidth.

"Operations are free, memory access is what counts."

Algorithms reflecting hardware evolution

- The arithmetic operations should use high precision formats natively supported by hardware.
- **Data access** should be as cheap as possible, **reduced precision**.
- **Radically decouple storage format from arithmetic format.**
    - *Double precision in all arithmetic operations.*
    - *Algorithm-aware precision when accessing data.*



John D. McCalpin (TACC)

- Compute power (#FLOPs) grows much faster than bandwidth.

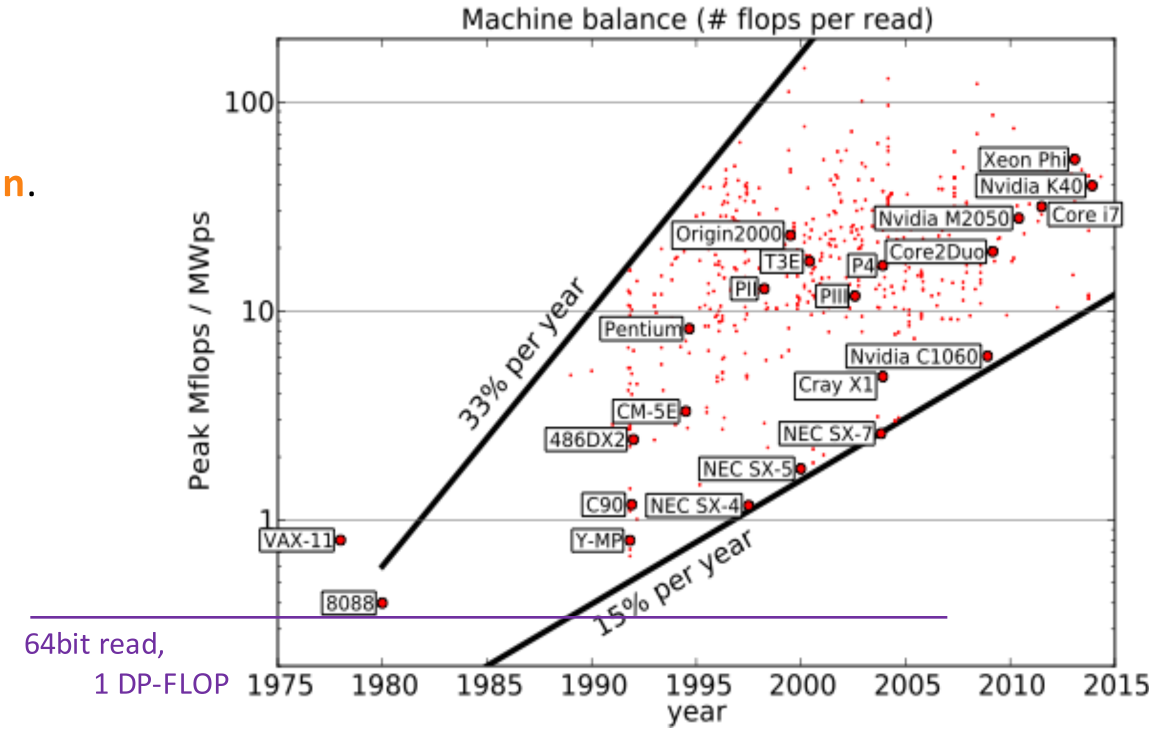
  "Operations are free, memory access is what counts."

Algorithms reflecting hardware evolution

- The arithmetic operations should use high precision formats natively supported by hardware.
- **Data access** should be as cheap as possible, **reduced precision**.
- **Radically decouple storage format from arithmetic format.**
  - *Double precision in all arithmetic operations.*
  - *Algorithm-aware precision when accessing data.*

Challenges when using double precision
+ IEEE single/half precision:

- Need explicit conversion.
- Data range reduction: protect against under- / overflow.
- Need to duplicate data in memory (half/single/double).



John D. McCalpin (TACC)

- Compute power (#FLOPs) grows much faster than bandwidth.

"Operations are free, memory access is what counts."

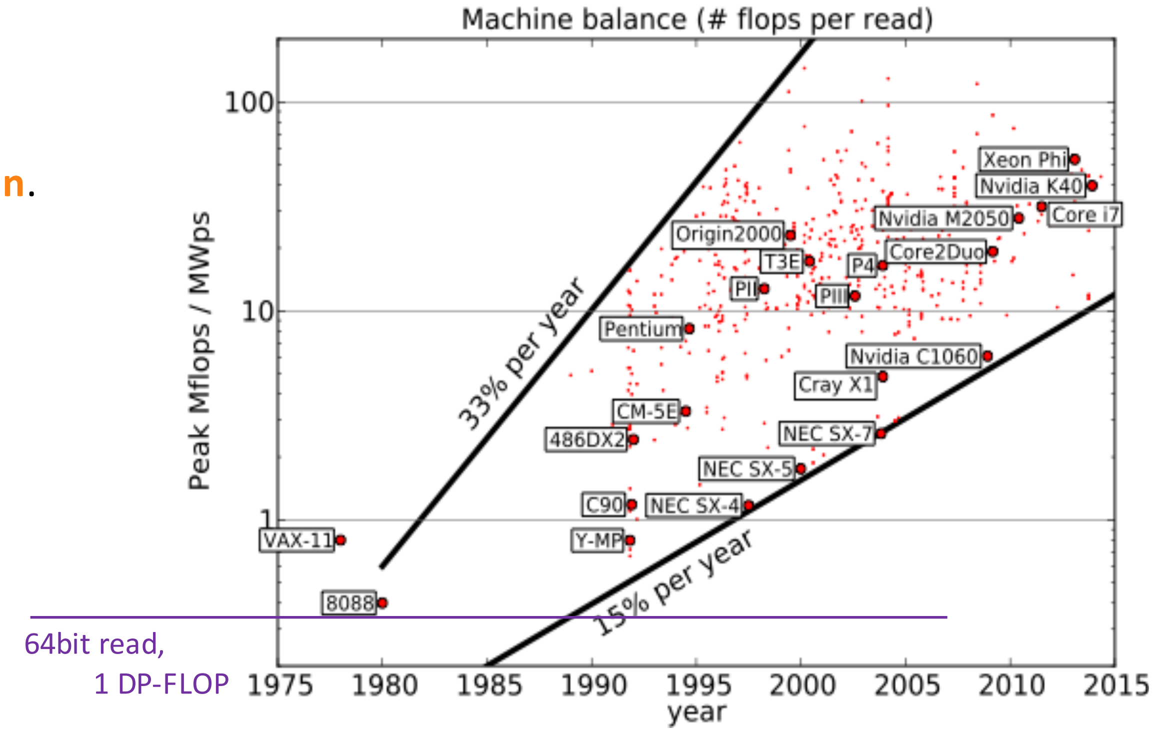
Algorithms reflecting hardware evolution

- The arithmetic operations should use high precision formats natively supported by hardware.
- **Data access** should be as cheap as possible, **reduced precision**.
- **Radically decouple storage format from arithmetic format.**
  - *Double precision in all arithmetic operations.*
  - *Algorithm-aware precision when accessing data.*

Challenges when using double precision
+ IEEE single/half precision:

- Need explicit conversion.
- Data range reduction: protect against under- / overflow.
- Need to duplicate data in memory (half/single/double).

- Better: **Customized Precision Format**



John D. McCalpin (TACC)

- Compute power (#FLOPs) grows much faster than bandwidth.

"Operations are free, memory access is what counts."

Use reduced precision for “approximate” Objects

- Preconditioners for iterative solvers.

Use reduced precision for "approximate" Objects

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

$$Ax = b \Leftrightarrow \tilde{A}x = \tilde{b}$$

Use reduced precision for "approximate" Objects

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

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

Use reduced precision for “approximate” Objects

- **Preconditioners for iterative solvers.**
  - Idea: Approximate inverse of system matrix to make the system “easier to solve”: $P^{-1} \approx A^{-1}$
    $\tilde{A} = P^{-1}A\,, \quad \tilde{b} = P^{-1}b$, and we solve $Ax = b \Leftrightarrow \tilde{A}x = \tilde{b}$.
- **Why should we store the preconditioner matrix P^{-1} in full (high) precision?**

Use reduced precision for “approximate” Objects

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

      $\tilde{A} = P^{-1}A$, $\tilde{b} = P^{-1}b$, and we solve $Ax = b \Leftrightarrow \tilde{A}x = \tilde{b}$.
- **Why should we store the preconditioner matrix P^{-1} in full (high) precision?**
    - **We have to ensure regularity!**

Use reduced precision for "approximate" Objects

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

    $\tilde{A} = P^{-1}A\,,\quad \tilde{b} = P^{-1}b$, and we solve $Ax = b \Leftrightarrow \tilde{A}x = \tilde{b}$.
- **Why should we store the preconditioner matrix P^{-1} in full (high) precision?**
  - **We have to ensure regularity!**

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

Use reduced precision for “approximate” Objects

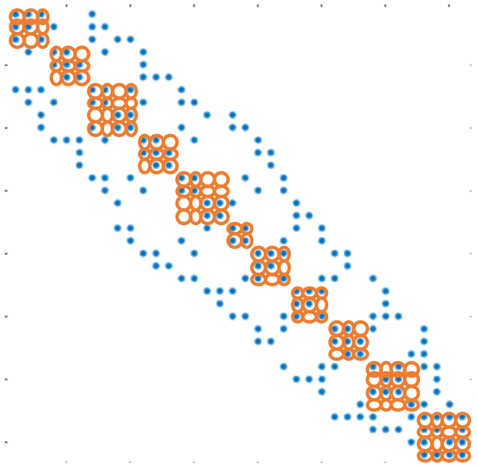
- **Preconditioners for iterative solvers.**
  - Idea: Approximate inverse of system matrix to make the system “easier to solve”: $P^{-1} \approx A^{-1}$

    $\tilde{A} = P^{-1}A, \quad \tilde{b} = P^{-1}b$, and we solve $Ax = b \Leftrightarrow \tilde{A}x = \tilde{b}$.
- **Why should we store the preconditioner matrix P^{-1} in full (high) precision?**
  - **We have to ensure regularity!**

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

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

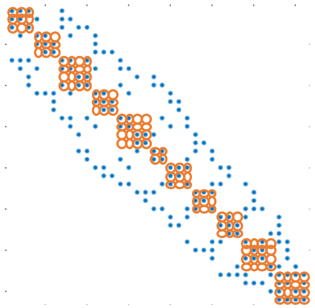
      Extreme case: one block of matrix size.

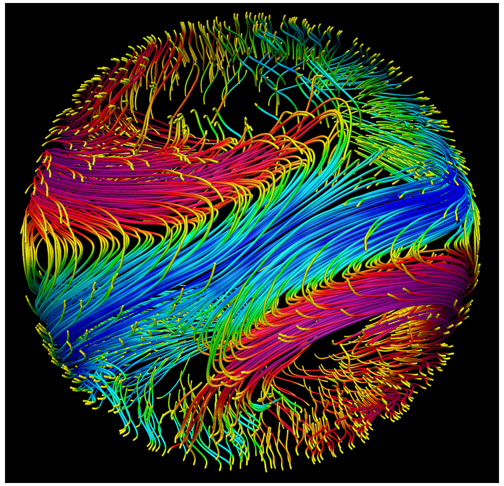


Block-Jacobi Preconditioning

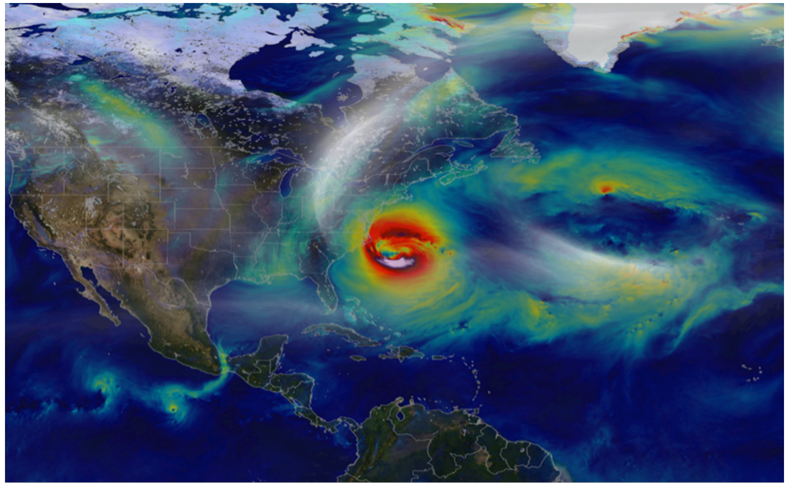
- **Block-Jacobi** method typically used as **preconditioner** inside linear- / Eigenvalue solvers.
- Target: large, sparse linear systems.

  discretizations often carry a **block-structure** (multiple variables per node).
- “Natural blocks” of small size (8, 12..).
- System matrix often stored in **sparse data structure** (CSR).

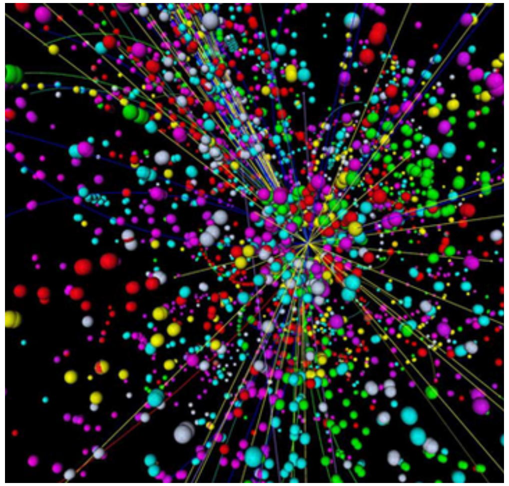




http://www.nas.nasa.gov/SC14/



https://science.nasa.gov/earth-science/focus-areas/earth-weather



http://i.livescience.com

Block-Jacobi Preconditioning

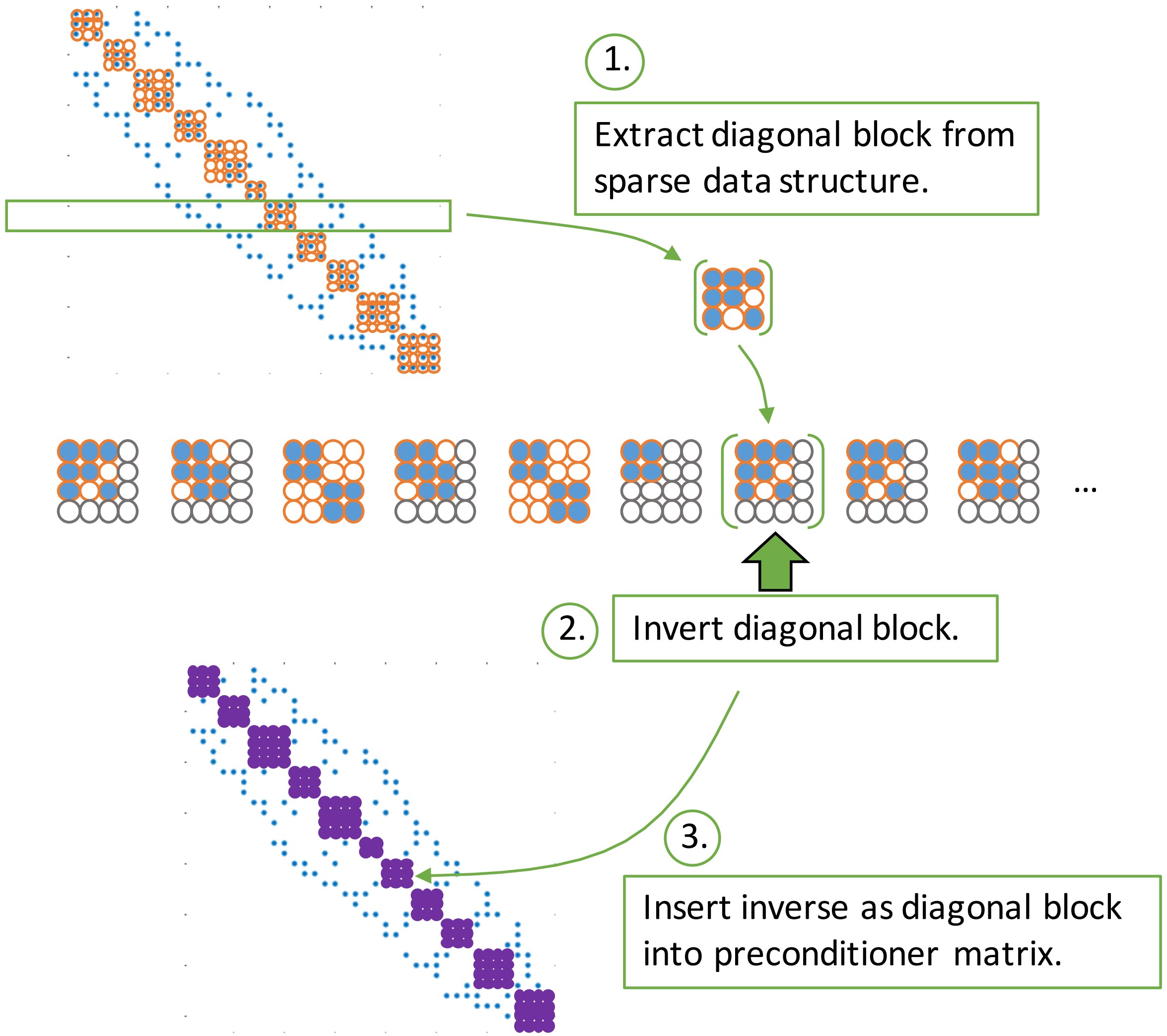
Preconditioner Setup:

- Identify the diagonal blocks $P = diag_B(A)$
- Form the block-Inverse $P^{-1} \approx A^{-1}$

Preconditioner Application:

- Apply the preconditioner in every solver iteration via:

$$y := P^{-1}x$$



Block-Jacobi Preconditioning

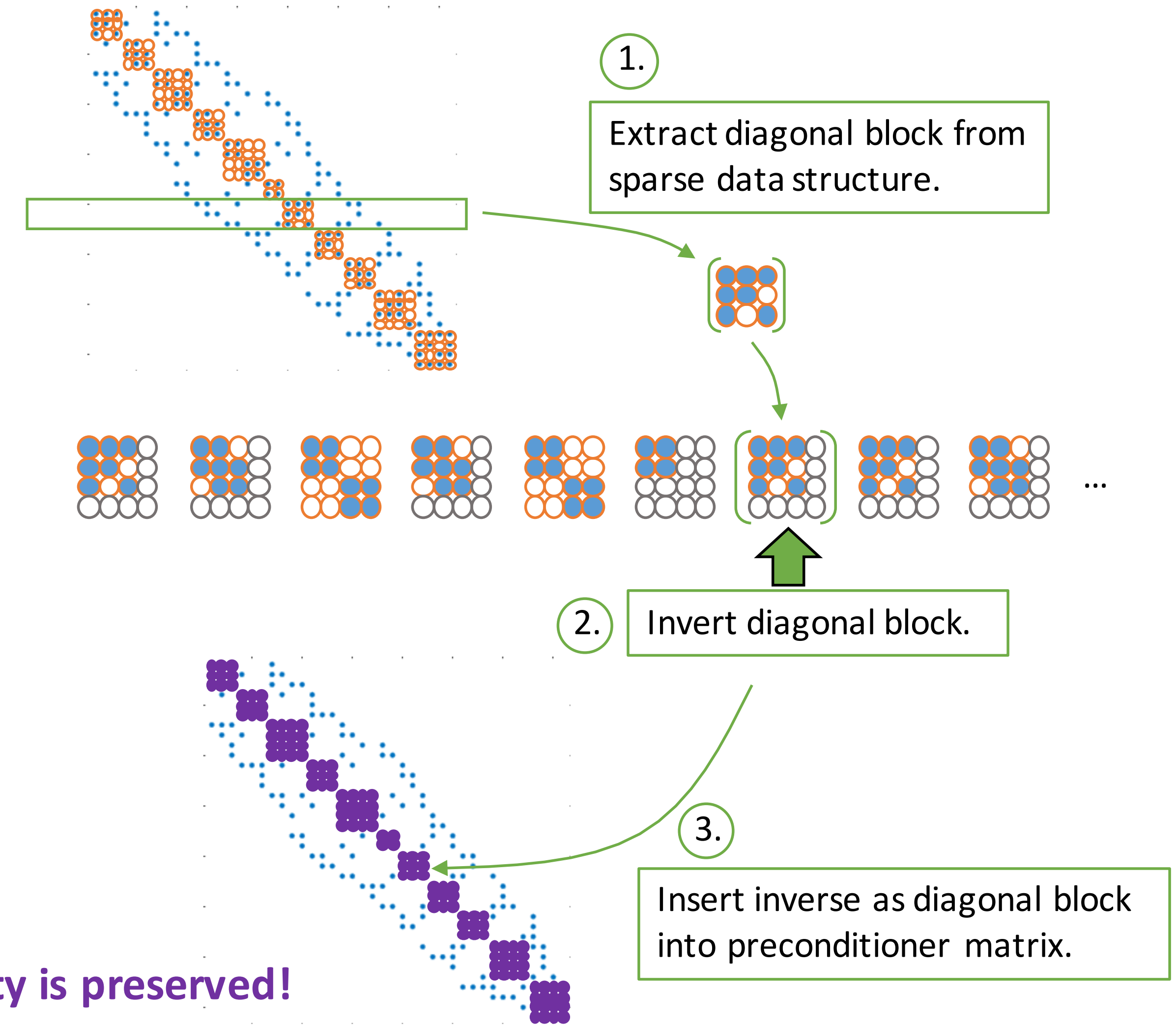
Preconditioner Setup:

- Identify the diagonal blocks $P = diag_B(A)$
- Form the block-Inverse $P^{-1} \approx A^{-1}$

Preconditioner Application:

- Apply the preconditioner in every solver iteration via:

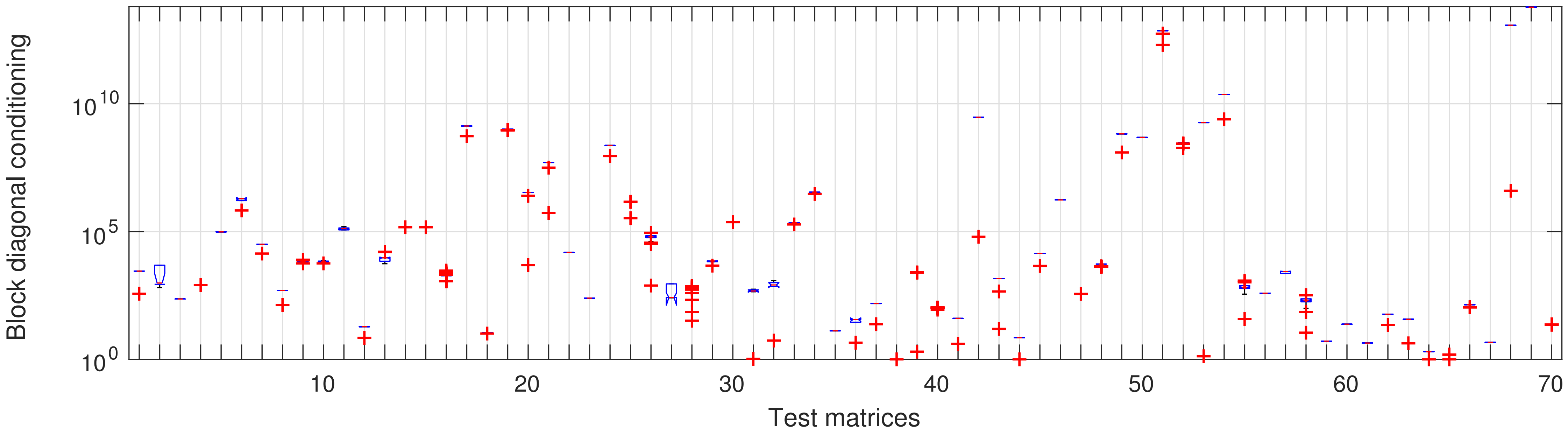
$$y := P^{-1}x$$



We can store diagonal blocks in lower precision, if regularity is preserved!

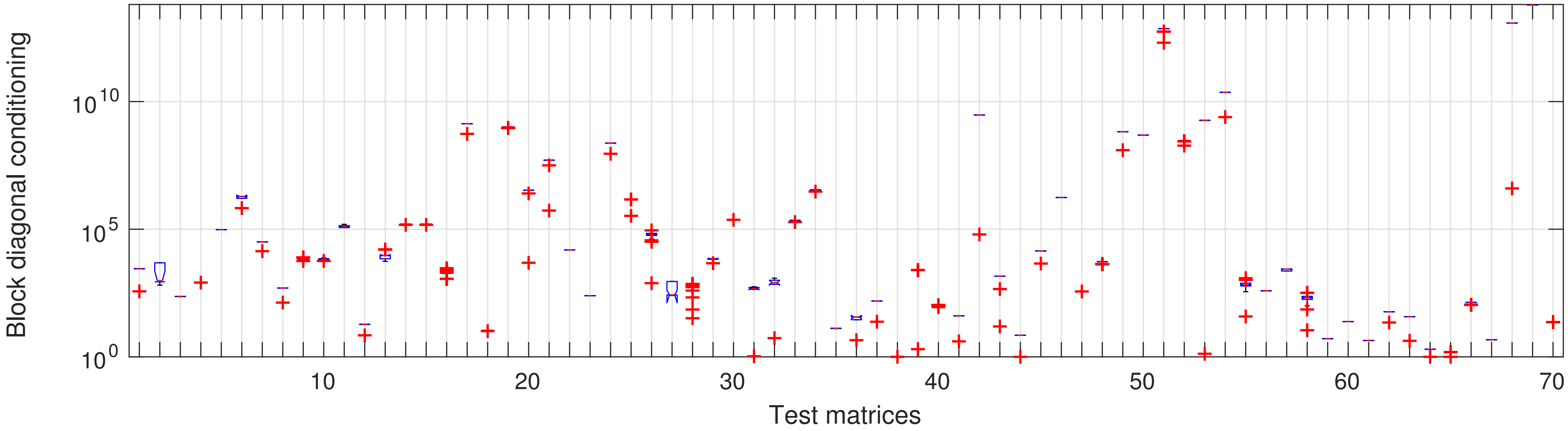
Customized Precision Block-Jacobi Preconditioning

- 70 matrices from the SuiteSparse Matrix Collection
- Use block-size 24 with Super-Variable agglomeration (24 is upper bound for size of blocks)
- Report conditioning of all arising diagonal blocks

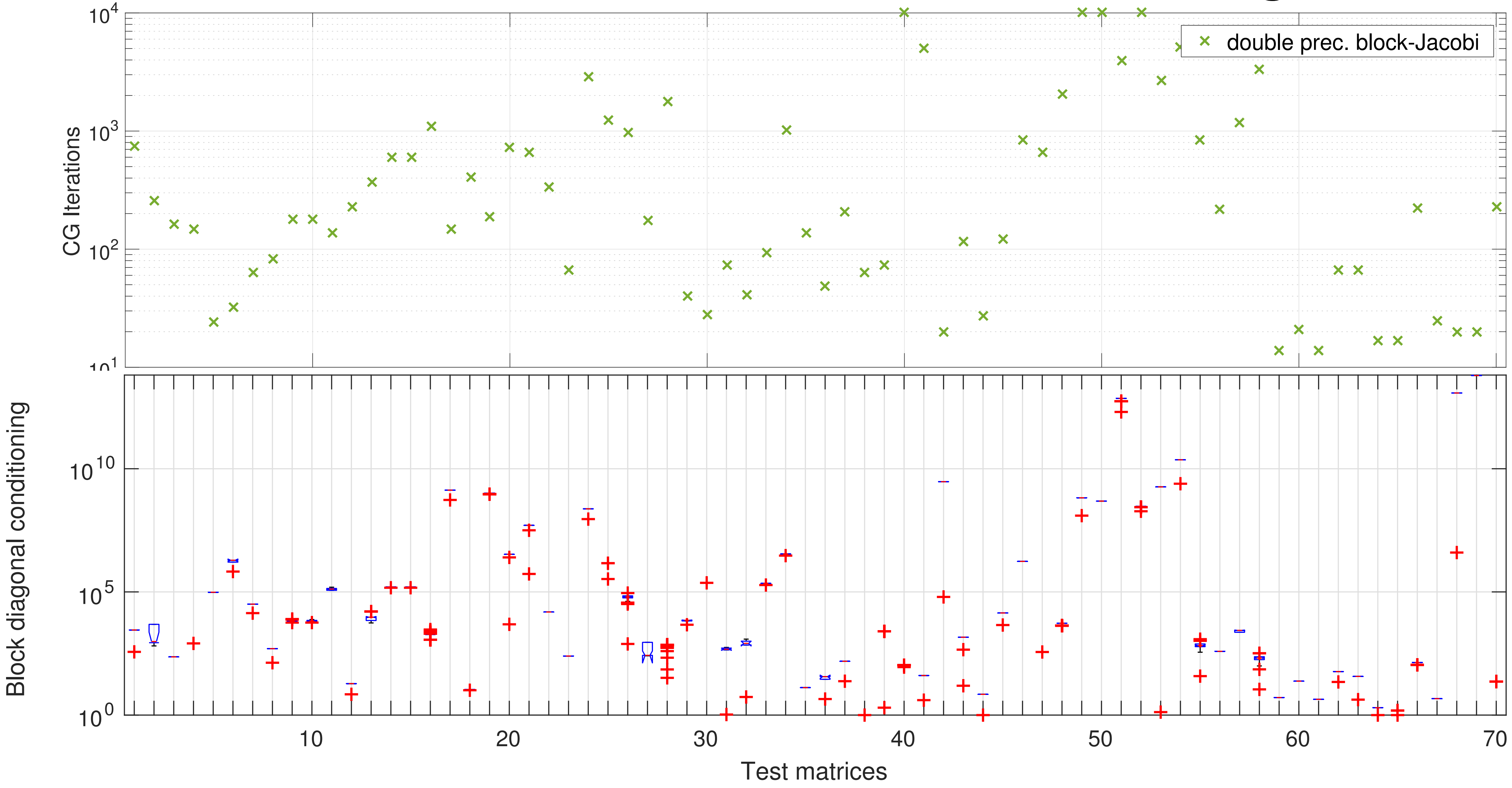


Customized Precision Block-Jacobi Preconditioning

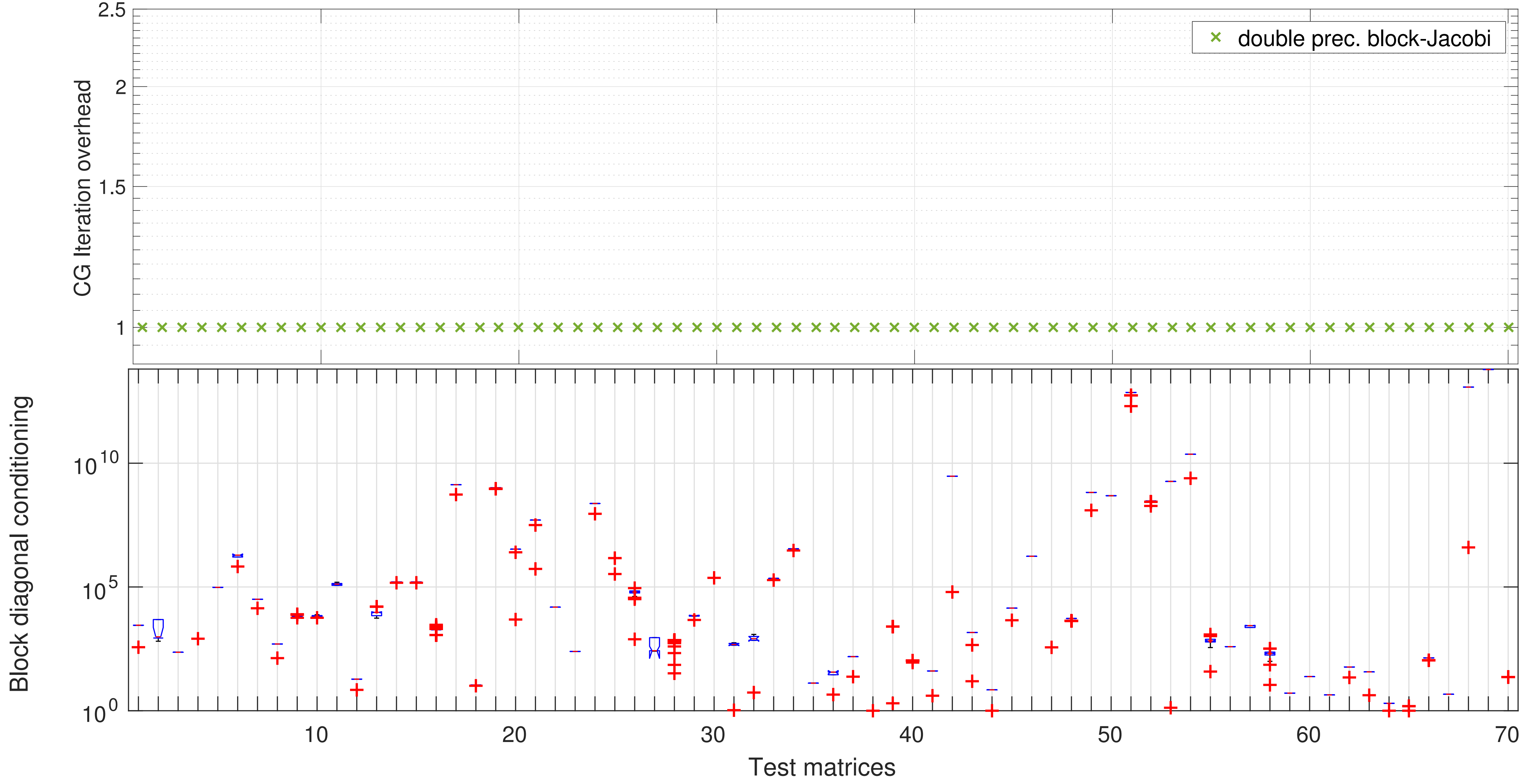
- 70 matrices from the SuiteSparse Matrix Collection
- Use block-size 24 with Super-Variable agglomeration (24 is upper bound for size of blocks)
- Report conditioning of all arising diagonal blocks
- Analyze the impact of storing block-Jacobi in lower precision a top-level Conjugate Gradient solver (CG)



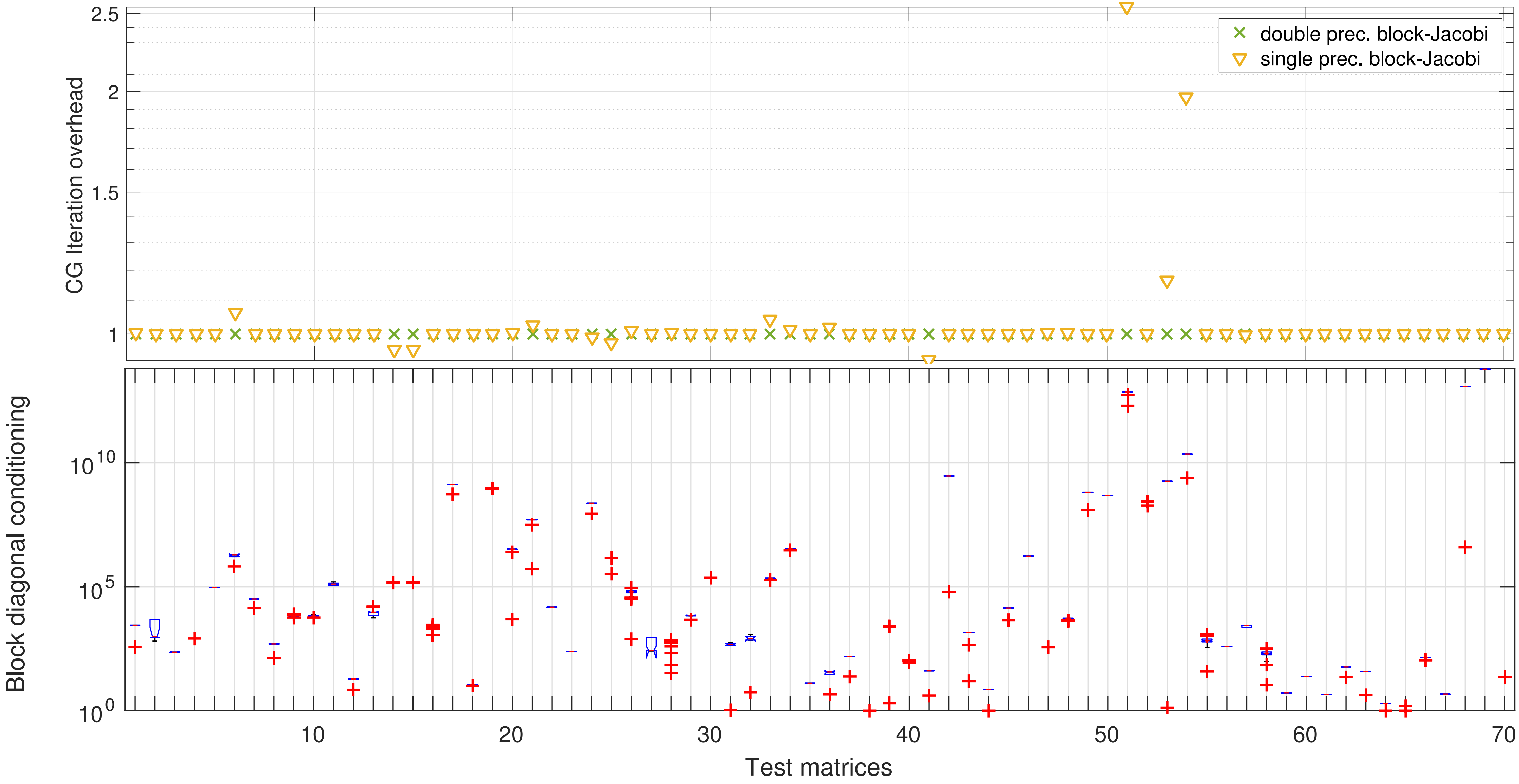
Customized Precision Block-Jacobi Preconditioning



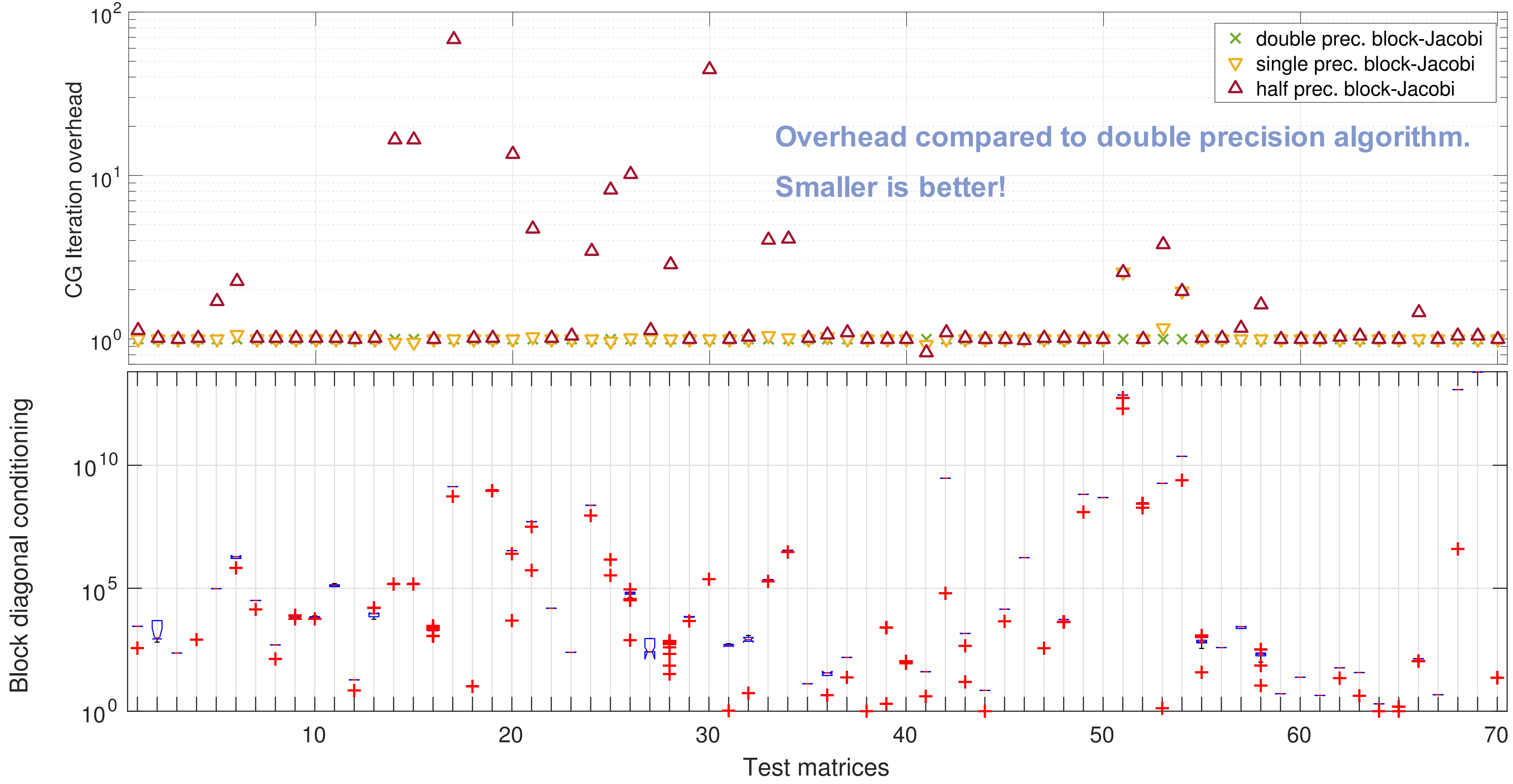
Customized Precision Block-Jacobi Preconditioning



Customized Precision Block-Jacobi Preconditioning



Customized Precision Block-Jacobi Preconditioning



Customized Precision Block-Jacobi Preconditioning

Modular Precision Idea:

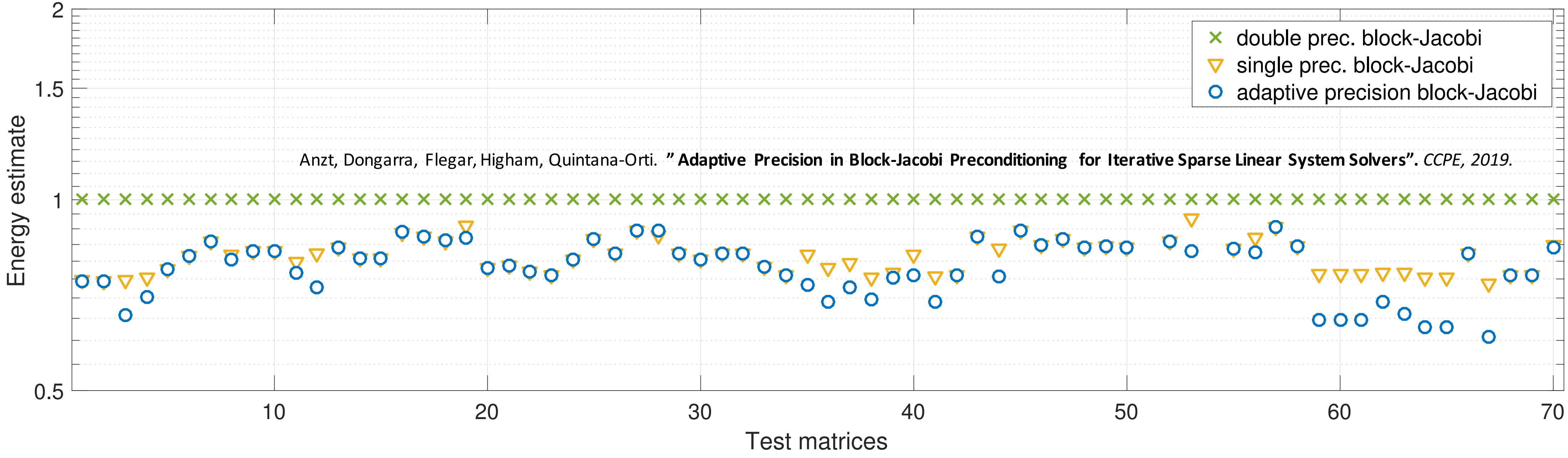
- **All computations use double precision!**
- Store distinct blocks in different formats
- Use **single precision as standard** storage format
- Where **necessary:** switch to **double**
- For well-conditioned blocks use **half precision**

Estimate conditioning of diagonal block

$> 10^6$ → **Store block in double precision**

→ **Store block in single precision**

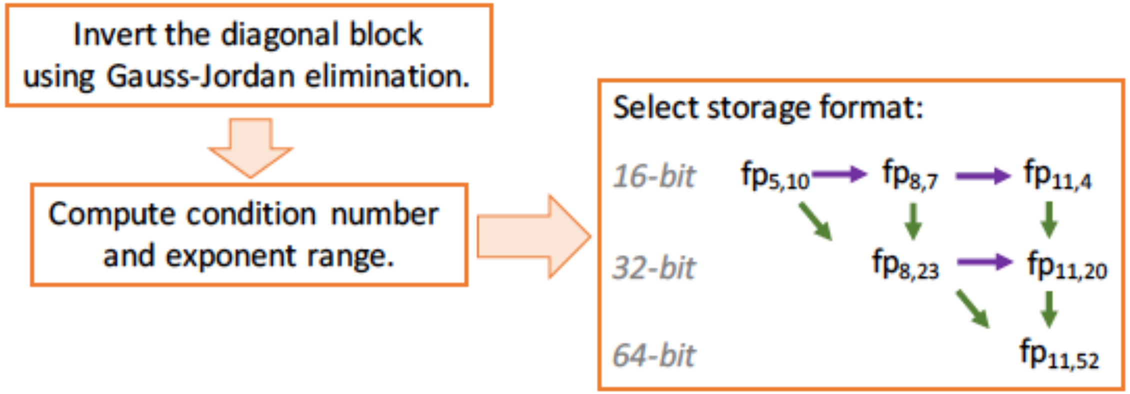
$< 10^1$ → **Store block in half precision**



Customized Precision Block-Jacobi Preconditioning

Modular Precision Idea:

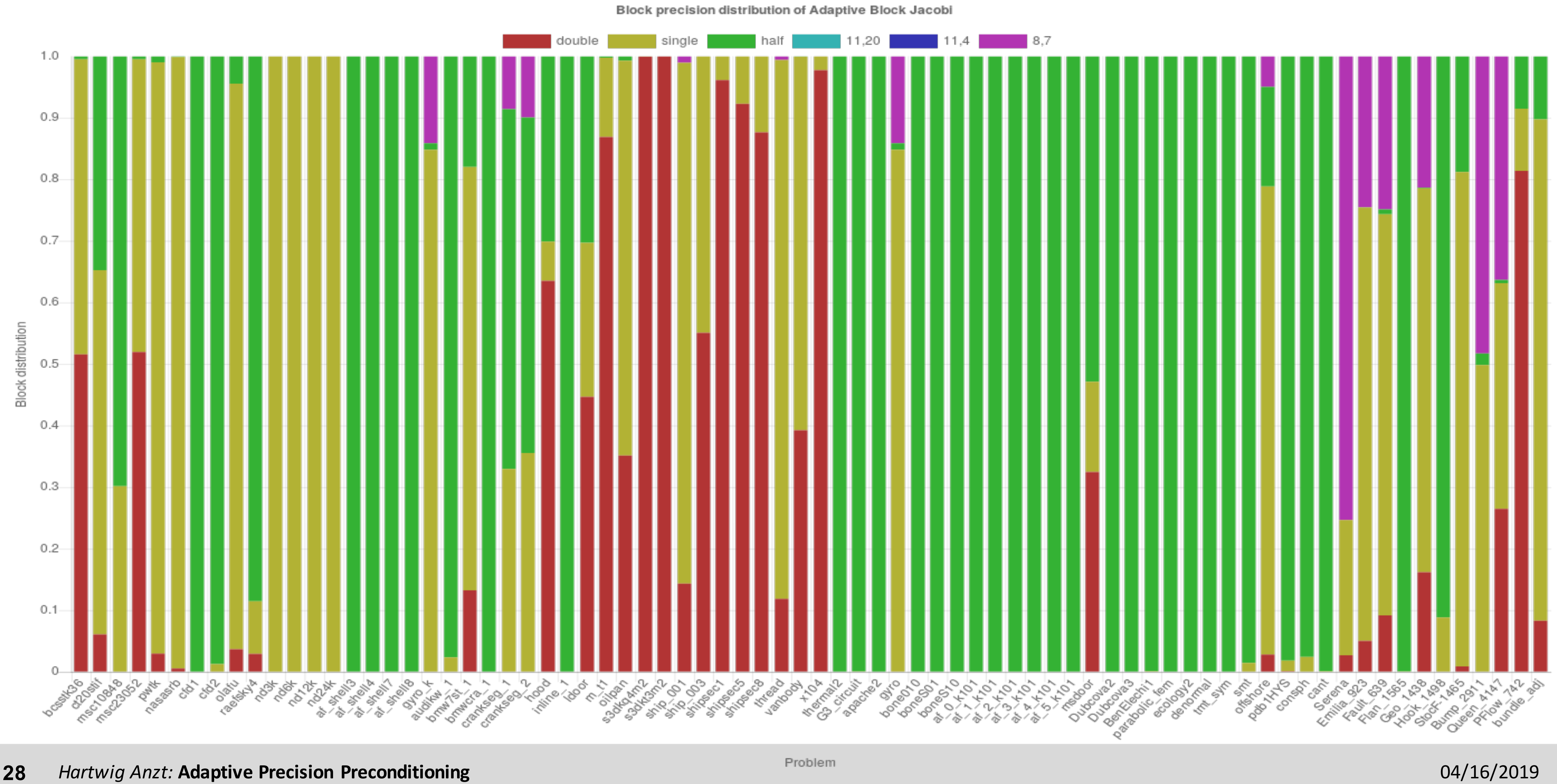
- **All computations use double precision!**
- **Depart from the rigid IEEE precision formats!**
- Preserve either 1 or 2 digits accuracy of the inverted diagonal blocks.



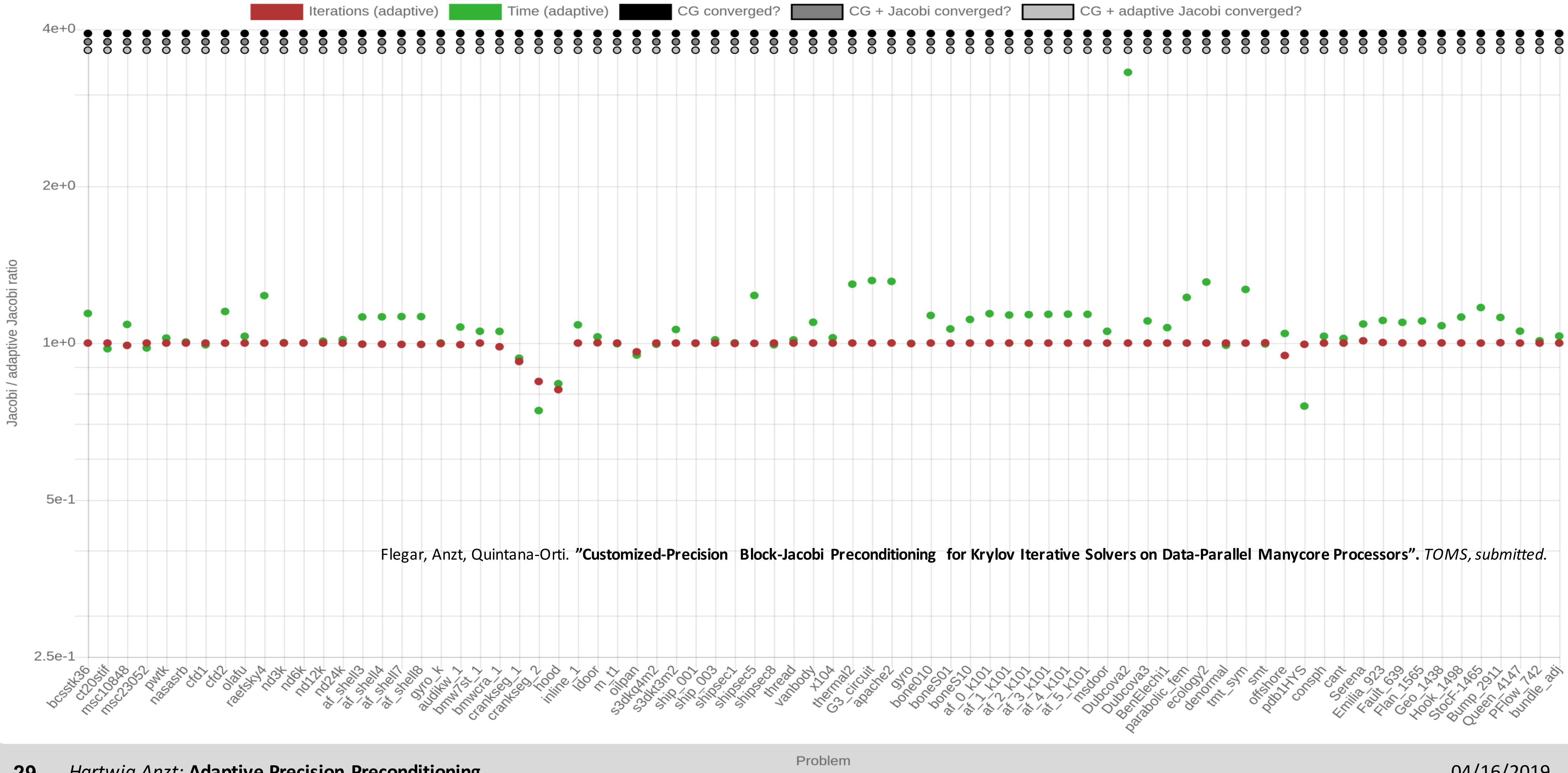
- Regularity preserved;
- Speedups / preconditioner quality problem-dependent;
- No flexible Krylov solver needed (Preconditioner constant operator);
- Can handle non-spd problems (inversion features pivoting);
- Preonditioner for any iterative preconditionable solver;

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

Customized Precision Block-Jacobi Preconditioning



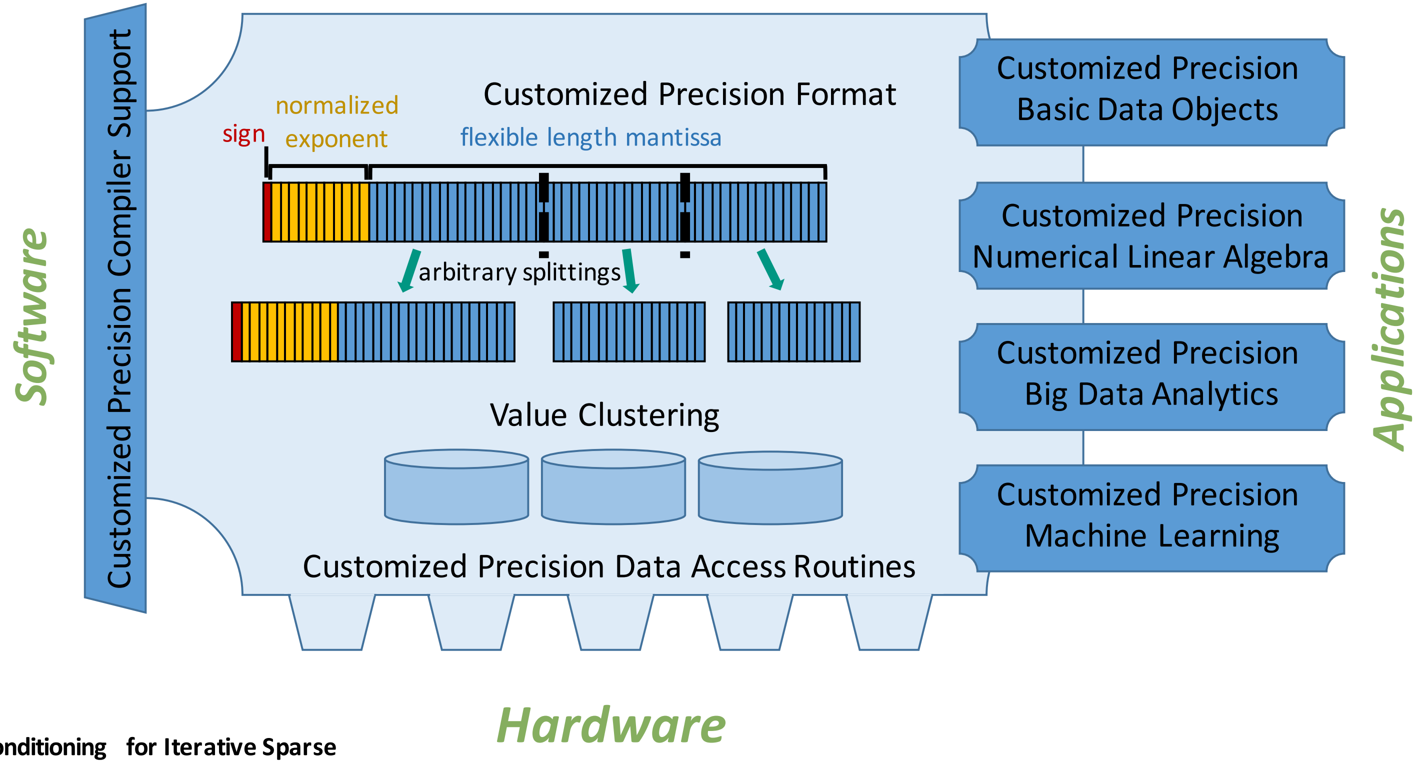
Customized Precision Block-Jacobi Preconditioning



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

Summary and next steps

- **Radically decoupling arithmetic precision from memory precision.**
- Using **customized precisions** for memory operations.
- Speedup of up to 1.3x for adaptive precision block-Jacobi preconditioning[1].
- Creating a **Modular Precision Ecosystem** inside Ginkgo.
  https://github.com/ginkgo-project/ginkgo



[1]Anzt, Dongarra, Flegar, Higham, Quintana-Orti. " **Adaptive Precision in Block-Jacobi Preconditioning for Iterative Sparse Linear System Solvers".** *CCPE, 2018.*