KIT, Karlsruhe, 12 June 2019

UNIVERSITAT JAUME I

Block-Jacobi preconditioning in Ginkgo

Goran Flegar

flegar@uji.es

Introduction

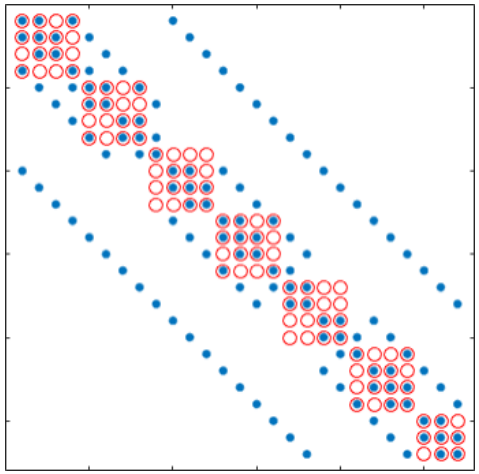
- I’m assuming we are all familiar with:
  - Iterative solution of linear systems
    - Why it’s useful
    - Operations within Krylov solvers (BLAS 1, SpMV, preconditioning)
  - Preconditioning
    - Replacing the original system with the preconditioned system
    - Changes needed within the Krylov solver to incorporate preconditioning

Block-Jacobi preconditioning

- Improve performance for problems with inherent block structure
  - Usually blocks are smaller than 30x30 (blocks can be of different sizes!)

Block-Jacobi preconditioning

- Improve performance for problems with inherent block structure
  - Usually blocks are smaller than 30x30 (blocks can be of different sizes!)



- Block-Jacobi preconditioning
  - Use only diagonal blocks for approximation

$$\mathrm{diag}(A) = [D_1, \ldots, D_k]$$
$$M := \mathrm{diag}(D_1, \ldots, D_k)$$

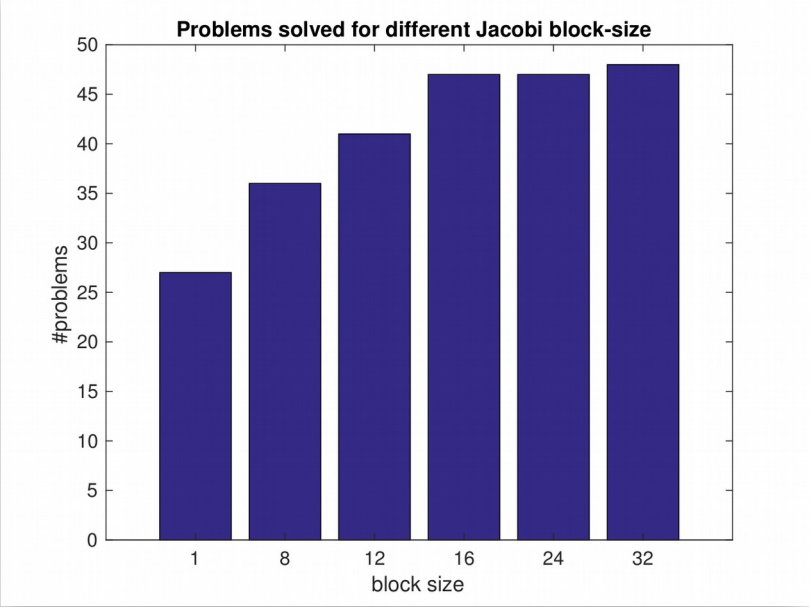
Benefits of block-Jacobi

- 56 matrices from SuiteSparse
- MAGMA-sparse open source library
  - IDR solver
  - Scalar Jacobi preconditioner
  - Supervariable agglomeration
    - Detects block structure of the matrix

M. Goetz and H. Anzt, "Machine Learning-Aided Numerical Linear Algebra: Convolutional Neural Networks for the Efficient Preconditioner Generation," 2018 IEEE/ACM 9th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (scalA), Dalla, TX, USA, 2018, pp. 49-56.

Benefits of block-Jacobi

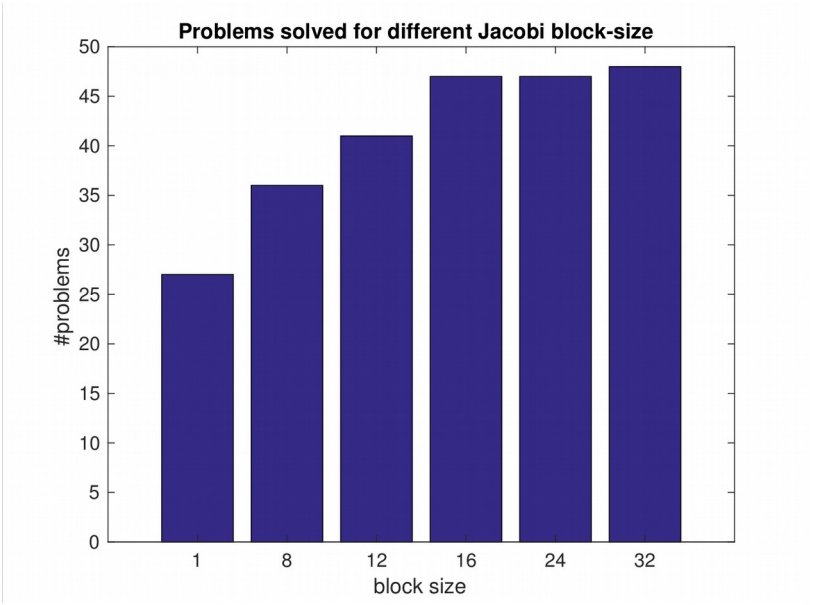
- 56 matrices from SuiteSparse
- MAGMA-sparse open source library
  - IDR solver
  - Scalar Jacobi preconditioner
  - Supervariable agglomeration
    - Detects block structure of the matrix
- Improves the robustness of the solver

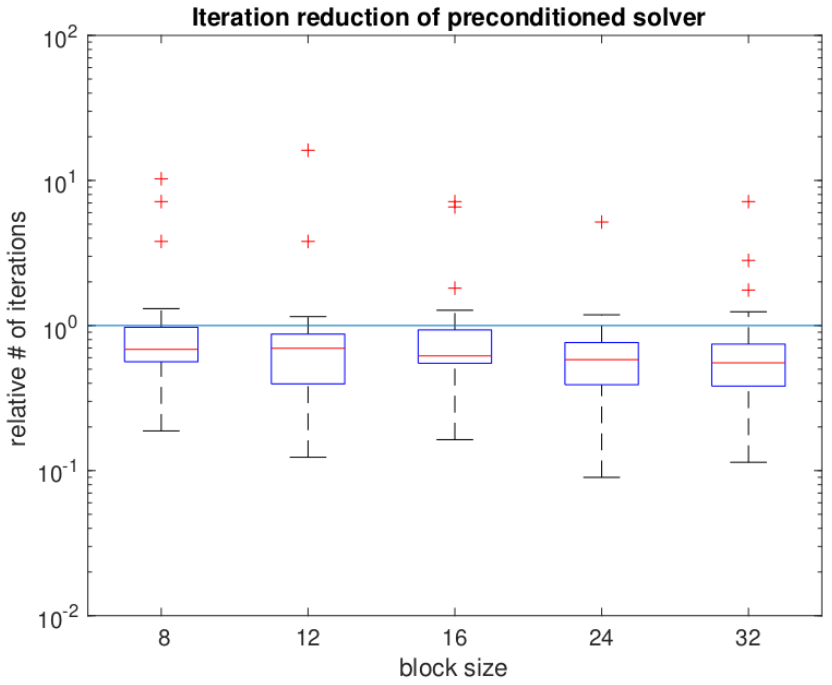


M. Goetz and H. Anzt, "Machine Learning-Aided Numerical Linear Algebra: Convolutional Neural Networks for the Efficient Preconditioner Generation," 2018 IEEE/ACM 9th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (scalA), Dalla, TX, USA, 2018, pp. 49-56.

Benefits of block-Jacobi

- 56 matrices from SuiteSparse
- MAGMA-sparse open source library
  - IDR solver
  - Scalar Jacobi preconditioner
  - Supervariable agglomeration
    - Detects block structure of the matrix
- Improves the robustness of the solver
- Improves convergence of the solver

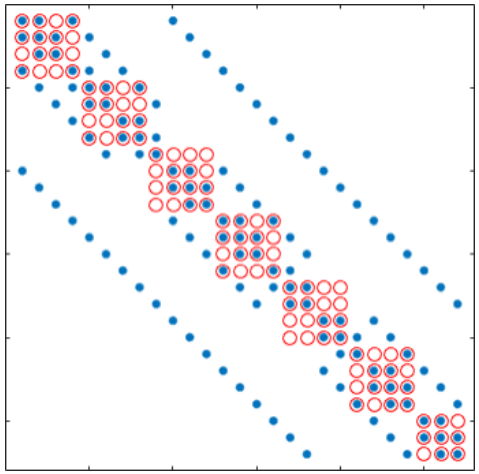




M. Goetz and H. Anzt, "Machine Learning-Aided Numerical Linear Algebra: Convolutional Neural Networks for the Efficient Preconditioner Generation," 2018 IEEE/ACM 9th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (scalA), Dalla, TX, USA, 2018, pp. 49-56.

Block-Jacobi preconditioning

- Improve performance for problems with inherent block structure
  - Usually blocks are smaller than 30x30 (blocks can be of different sizes!)



- Block-Jacobi preconditioning
  - Use only diagonal blocks for approximation

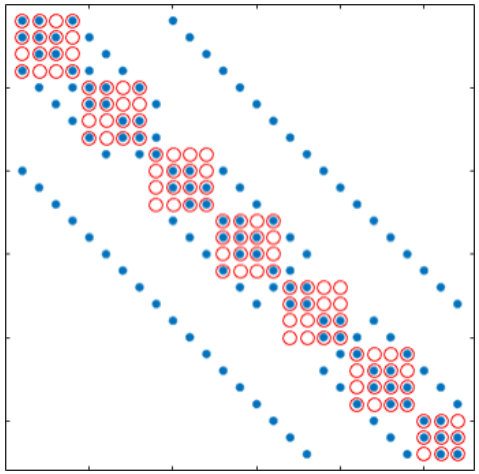
$$\mathrm{diag}(A) = [D_1, \ldots, D_k]$$

$$M := \mathrm{diag}(D_1, \ldots, D_k)$$

$$y := M^{-1} z \longrightarrow y_i := D_i^{-1} z_i, \ \forall i$$

Block-Jacobi preconditioning

- Improve performance for problems with inherent block structure
  - Usually blocks are smaller than 30x30 (blocks can be of different sizes!)



- Block-Jacobi preconditioning
  - Use only diagonal blocks for approximation

$$\mathrm{diag}(A) = [D_1, \ldots, D_k]$$
$$M := \mathrm{diag}(D_1, \ldots, D_k)$$

$$y := M^{-1} z \longrightarrow y_i := D_i^{-1} z_i, \ \forall i$$

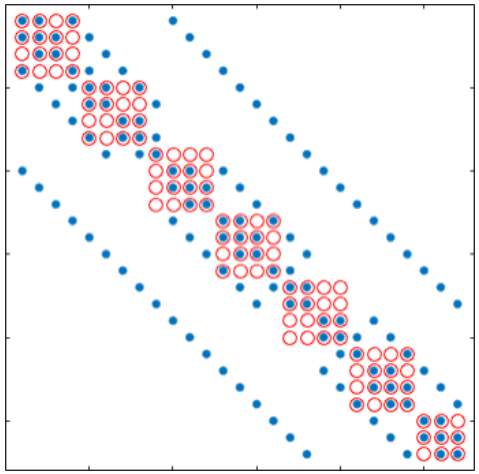
$$\widetilde{D}_i := \mathrm{inv}(D_i)$$

$$y_i := \widetilde{D}_i z_i$$

inv = Gauss-Jordan elimination

Block-Jacobi preconditioning

- Improve performance for problems with inherent block structure
  - Usually blocks are smaller than 30x30 (blocks can be of different sizes!)



- Block-Jacobi preconditioning
  - Use only diagonal blocks for approximation

$$\text{diag}(A) = [D_1, \ldots, D_k]$$
$$M := \text{diag}(D_1, \ldots, D_k)$$

$$y := M^{-1} z \longrightarrow y_i := D_i^{-1} z_i, \ \forall i$$

$$\widetilde{D}_i := \text{inv}(D_i)$$ **Setup**

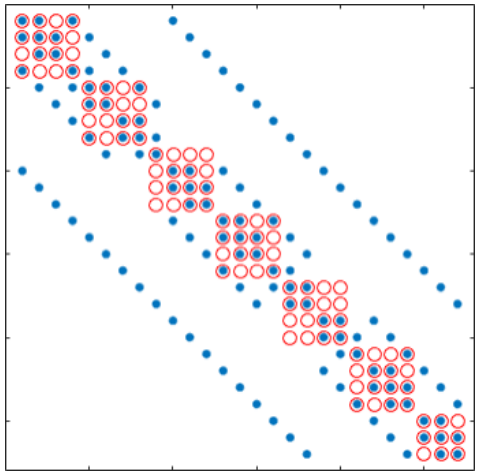
$$y_i := \widetilde{D}_i z_i$$ **Application**

Inversion-based block-Jacobi

inv = Gauss-Jordan elimination

Block-Jacobi preconditioning

- Current focus: improve performance for problems with inherent block structure
  - Usually blocks are smaller than 30x30 (blocks can be of different sizes!)



- Block-Jacobi preconditioning
  - Use only diagonal blocks for approximation

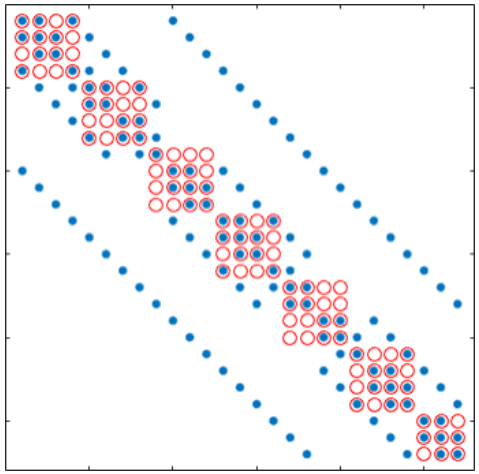
$$\mathrm{diag}(A) = [D_1, \ldots, D_k]$$
$$M := \mathrm{diag}(D_1, \ldots, D_k)$$

$$y := M^{-1} z \longrightarrow y_i := D_i^{-1} z_i, \ \forall i$$

$$\longrightarrow D_i y_i = z_i$$

Block-Jacobi preconditioning

- Current focus: improve performance for problems with inherent block structure
  - Usually blocks are smaller than 30x30 (blocks can be of different sizes!)



- Block-Jacobi preconditioning
  - Use only diagonal blocks for approximation

$$\text{diag}(A) = [D_1, \ldots, D_k]$$
$$M := \text{diag}(D_1, \ldots, D_k)$$

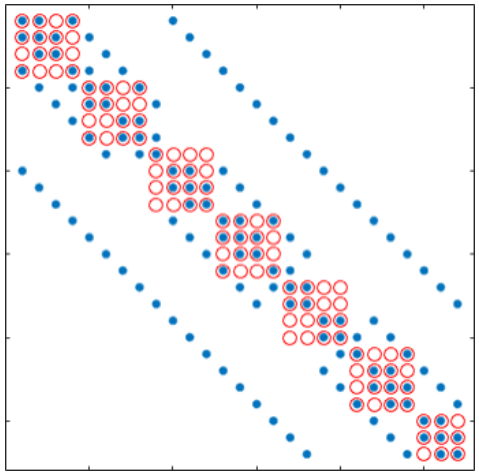
$$y := M^{-1} z \longrightarrow y_i := D_i^{-1} z_i, \ \forall i$$

$$D_i = L_i U_i$$

$$\longrightarrow D_i y_i = z_i \longrightarrow \begin{matrix} U_i y_i = w_i \\ L_i w_i = z_i \end{matrix}$$

Block-Jacobi preconditioning

- Current focus: improve performance for problems with inherent block structure
  - Usually blocks are smaller than 30x30 (blocks can be of different sizes!)



- Block-Jacobi preconditioning
  - Use only diagonal blocks for approximation

$$\text{diag}(A) = [D_1, \ldots, D_k]$$
$$M := \text{diag}(D_1, \ldots, D_k)$$

$$y := M^{-1} z \longrightarrow y_i := D_i^{-1} z_i, \ \forall i$$

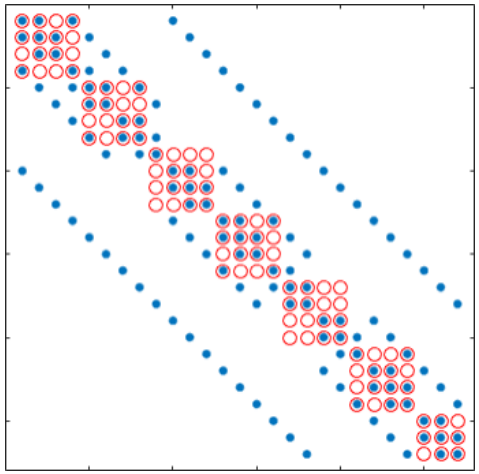
$$D_i = L_i U_i$$ **Setup**

$$\longrightarrow D_i y_i = z_i \longrightarrow \begin{aligned} U_i y_i &= w_i \\ L_i w_i &= z_i \end{aligned}$$ **Application**

Decomposition-based block-Jacobi

Block-Jacobi preconditioning

- Current focus: improve performance for problems with inherent block structure
  - Usually blocks are smaller than 30x30 (blocks can be of different sizes!)



- Block-Jacobi preconditioning
  - Use only diagonal blocks for approximation

$$\mathrm{diag}(A) = [D_1, \ldots, D_k]$$
$$M := \mathrm{diag}(D_1, \ldots, D_k)$$

$$y := M^{-1} z \longrightarrow y_i := D_i^{-1} z_i, \ \forall i$$

$$\longrightarrow D_i y_i = z_i \longrightarrow$$

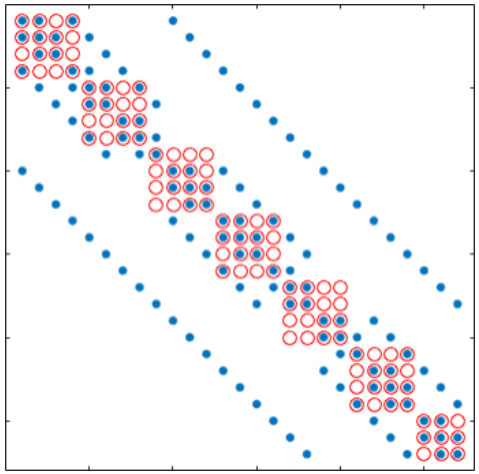
$$\widetilde{D}_i := \mathrm{gh}_d(D_i)$$

$$z_i := \mathrm{gh}_a(\widetilde{D}_i, y_i)$$

gh = Gauss-Huard decomposition / application

Block-Jacobi preconditioning

- Current focus: improve performance for problems with inherent block structure
  - Usually blocks are smaller than 30x30 (blocks can be of different sizes!)



- Block-Jacobi preconditioning
  - Use only diagonal blocks for approximation

$$\mathrm{diag}(A) = [D_1, \ldots, D_k]$$
$$M := \mathrm{diag}(D_1, \ldots, D_k)$$

$$y := M^{-1} z \longrightarrow y_i := D_i^{-1} z_i, \ \forall i$$

$$\longrightarrow D_i y_i = z_i \longrightarrow$$

$$\widetilde{D}_i := \mathrm{gh}_d(D_i) \quad \text{Setup}$$

$$z_i := \mathrm{gh}_a(\widetilde{D}_i, y_i) \quad \text{Application}$$

Decomposition-based block-Jacobi

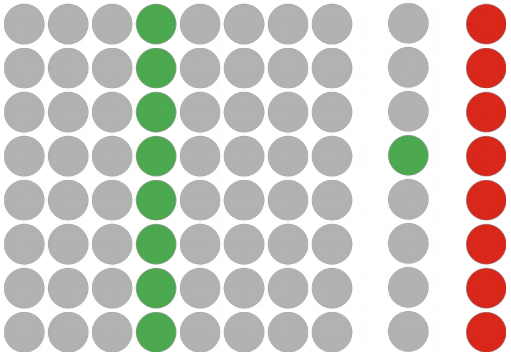
Block-Jacobi setup & application ecosystem

Setup

Application

Gauss-Jordan inversion

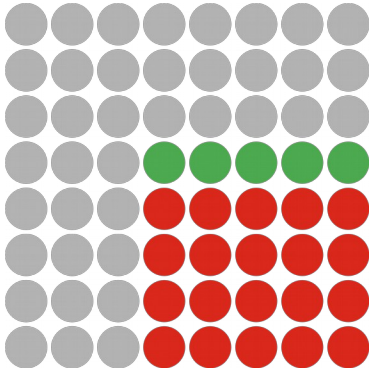
Inversion-based
($2n^3 + 2n^2$ FLOPS)
matrix-vector multiply



H. Anzt et al. “Variable-size batched Gauss–Jordan elimination for block-Jacobi preconditioning on graphics processors”, ParCo’19

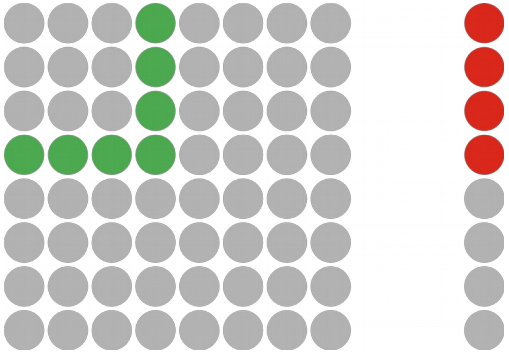
Gauss-Huard decomposition

LU factorization



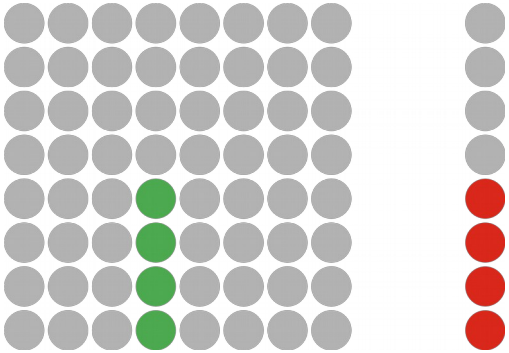
Decomposition-based
($2/3n^3 + 2n^2$ FLOPS)

Gauss-Huard solve



H. Anzt et al. “Variable-Size Batched Gauss-Huard for Block-Jacobi Preconditioning”, ICCS’17

2x triangular solve



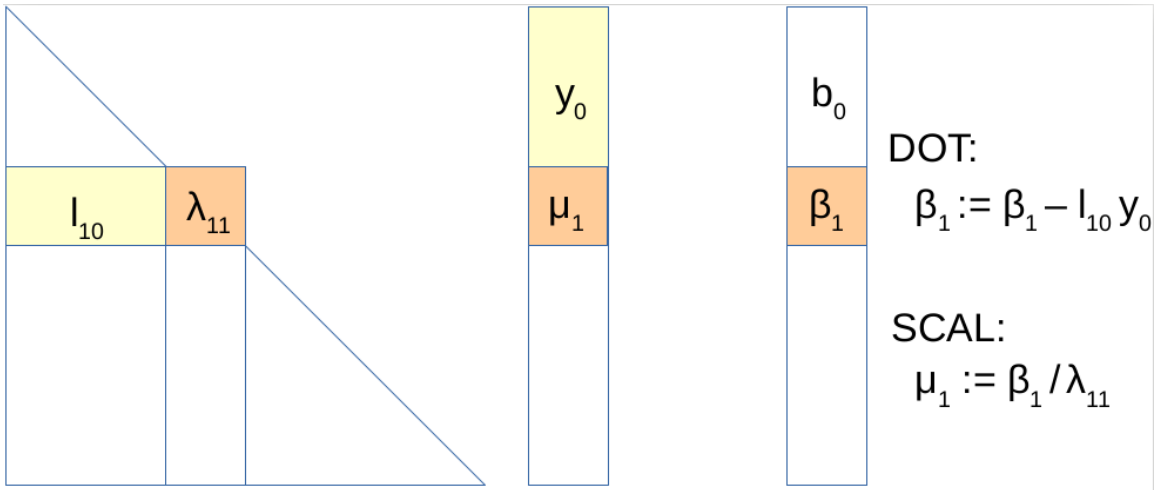
H. Anzt et al. “Flexible-Size Batched LU for Small Matrices and its Integration into Block-Jacobi Preconditioning”, ICPP’17

- read - write

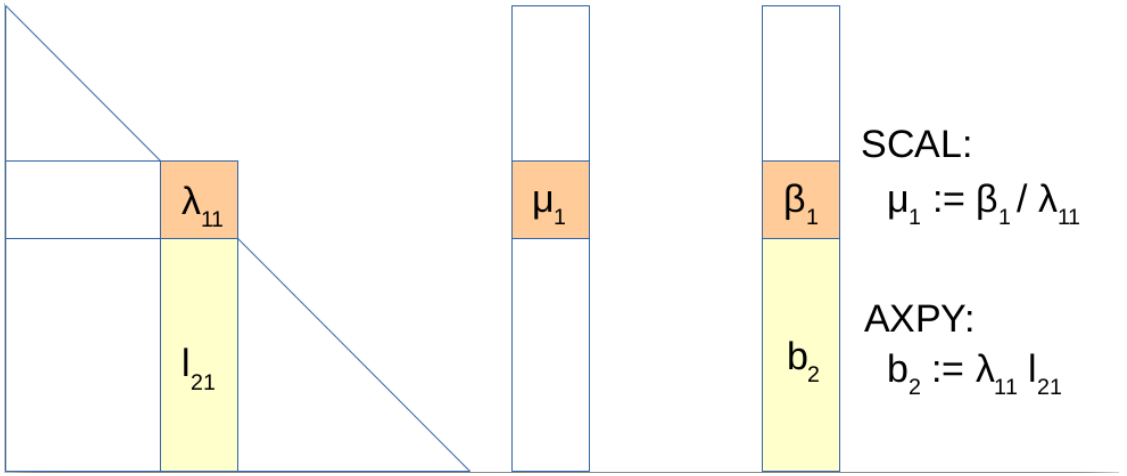
Triangular solves

- Use “eager” triangular solves
  - Cast solution vector updates in terms of axpy, not in terms of dot product!

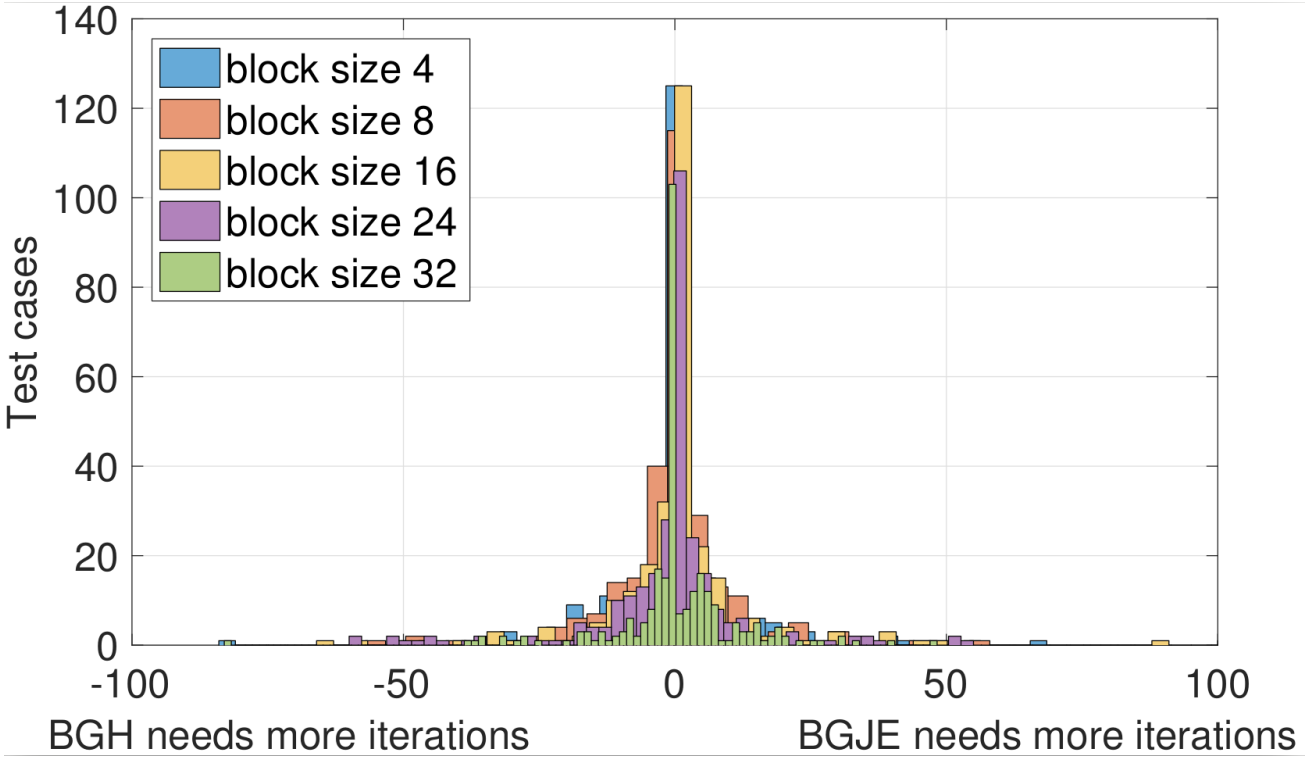
“Lazy” triangular solve



“Eager” triangular solve



No signs of instability in practice



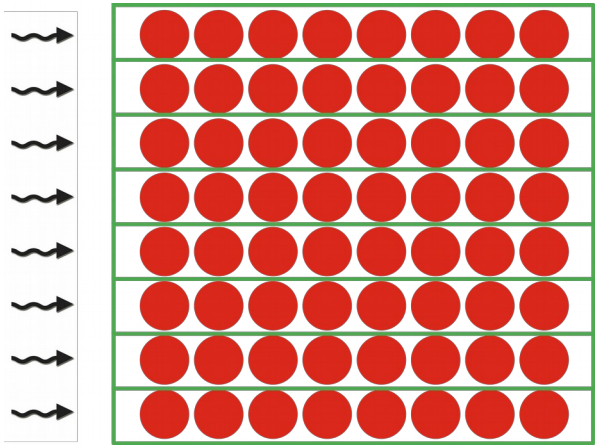
Batched inversion and decomposition

- Assign one warp to each problem
  - hardware SIMD unit, represented as a group of 32 threads in CUDA

Batched inversion and decomposition

- Assign one warp to each problem
  - hardware SIMD unit, represented as a group of 32 threads in CUDA

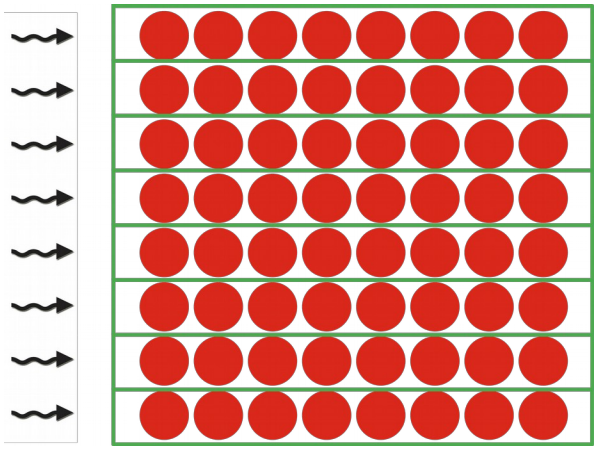
- Process each row (column) by a single thread
  - Able to support problems of size up to 32-by-32
  - Keep the entire row (column) in thread's registers
  - Communicate data between rows via warp-shuffles
  - Current implementation: use padding for problems of smaller sizes
  - Future work: multiple smaller problems per warp



Batched inversion and decomposition

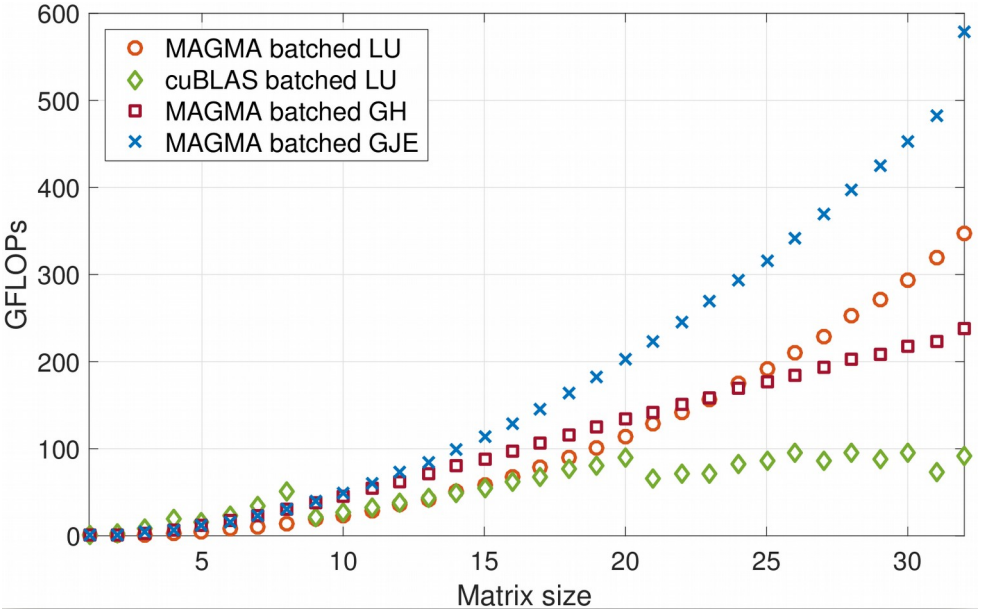
- Assign one warp to each problem
  - hardware SIMD unit, represented as a group of 32 threads in CUDA

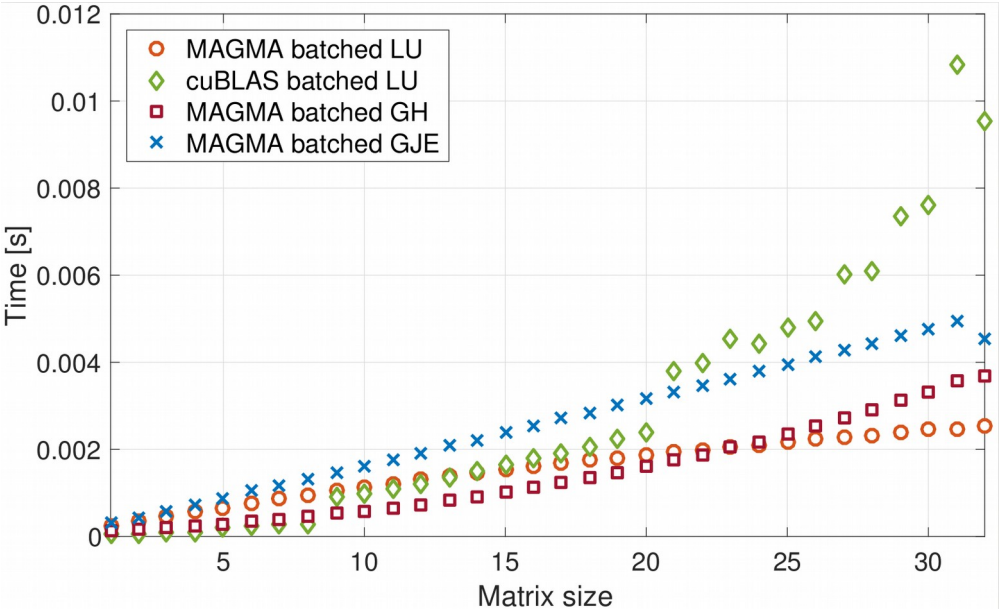
- Process each row (column) by a single thread
  - Able to support problems of size up to 32-by-32
  - Keep the entire row (column) in thread's registers
  - Communicate data between rows via warp-shuffles
  - Current implementation: use padding for problems of smaller sizes
  - Future work: multiple smaller problems per warp



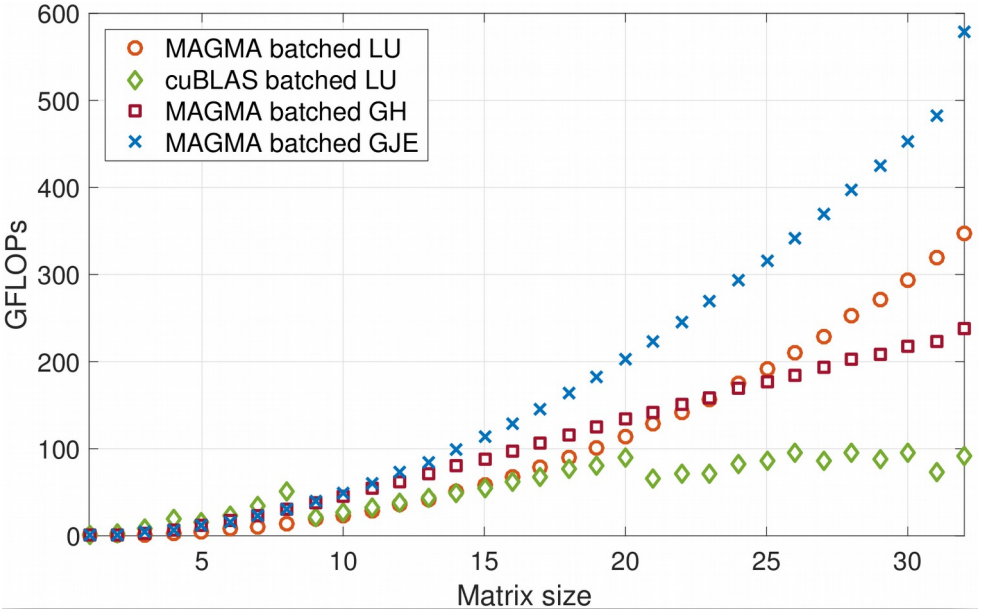
- Use implicit pivoting
  - Do not explicitly swap rows (column), "re-assign" the threads instead

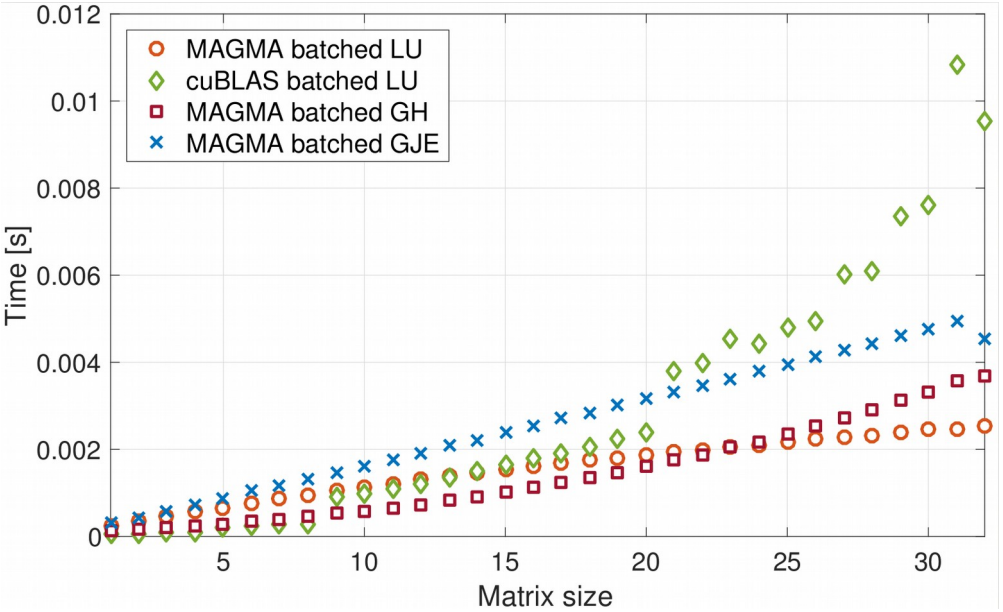
Batched routines performance





Batched routines performance

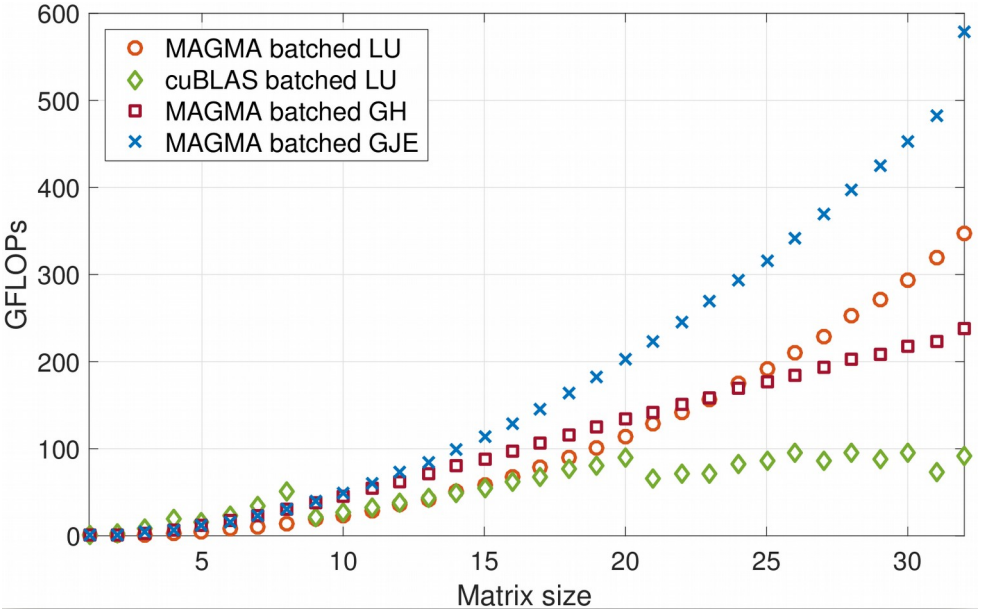


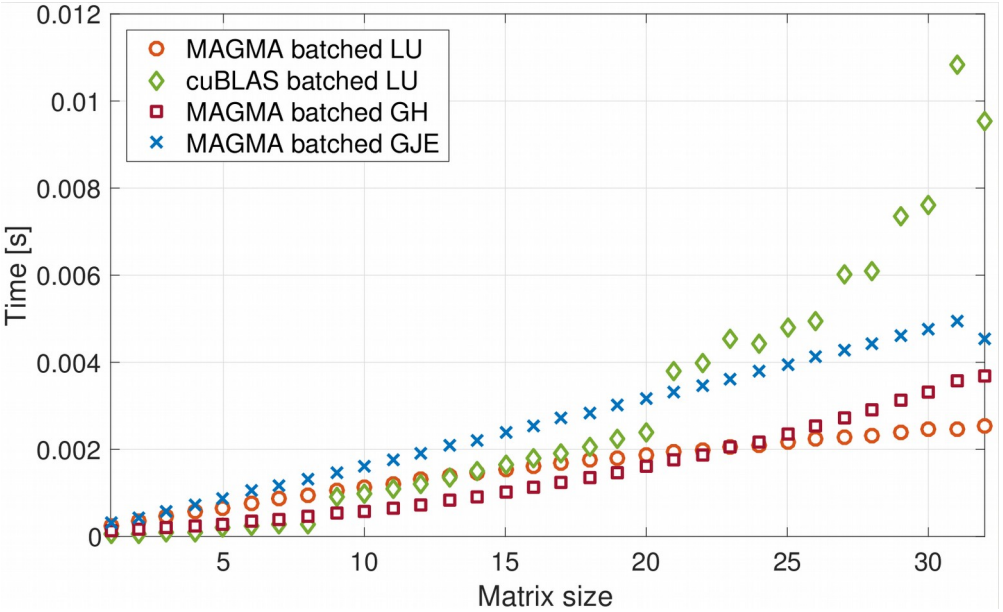


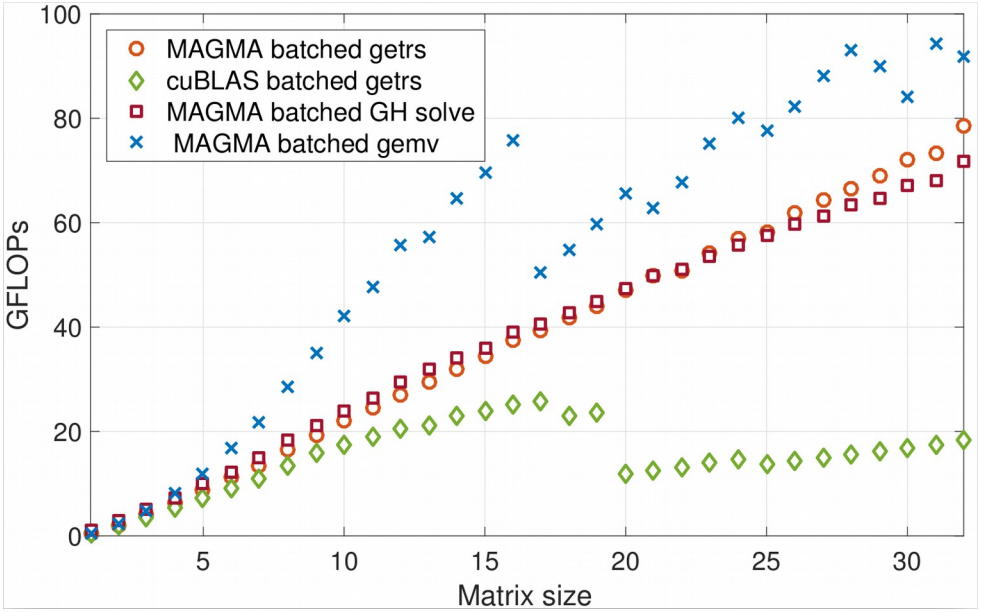
MAGMA-sparse routines can also:

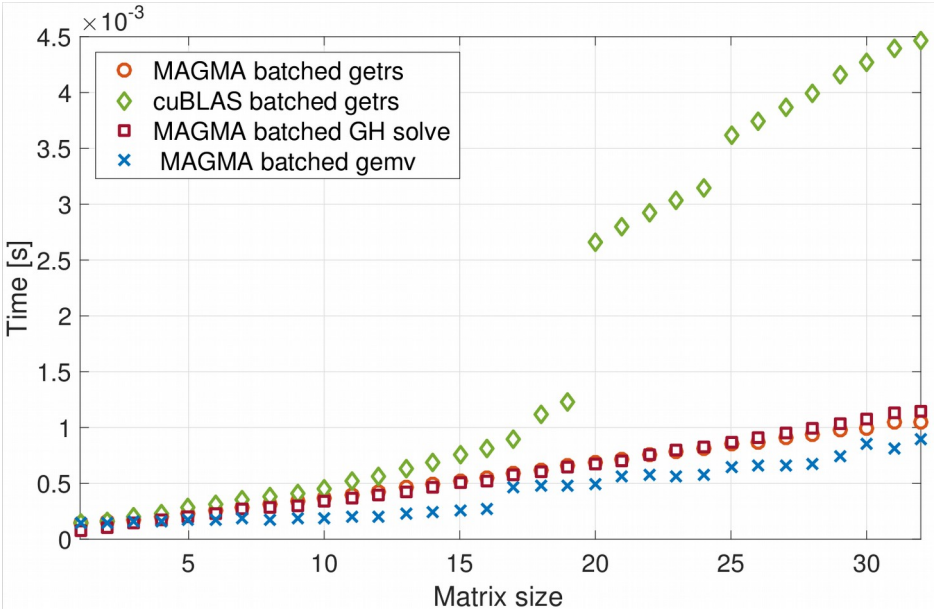
- Handle problems of different sizes
- Integrate diagonal block extraction and diagonal block decomposition / inversion into a single kernel

Batched routines performance



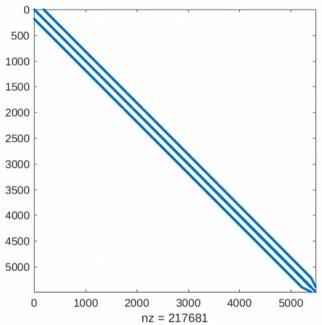


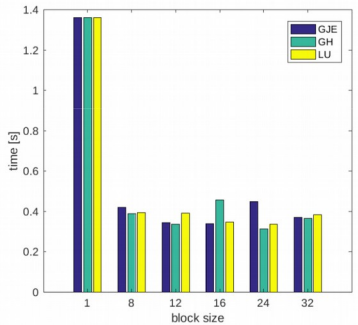




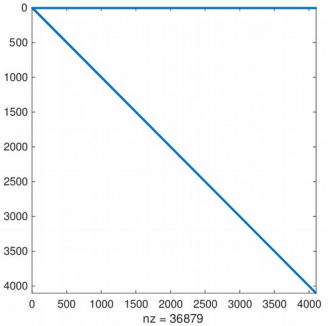
Complete solver runtime

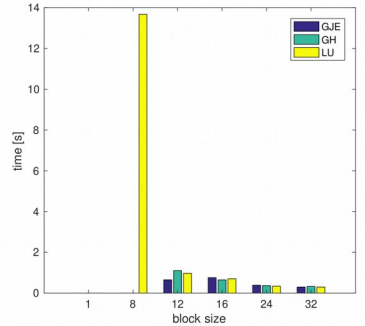
s2rmt3m1



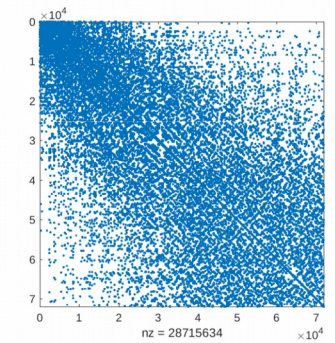


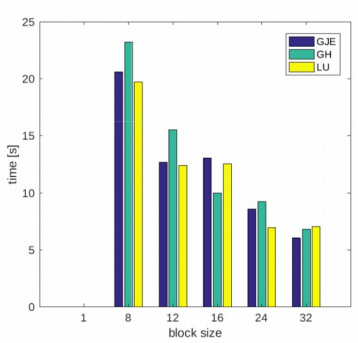
Chebyshev3



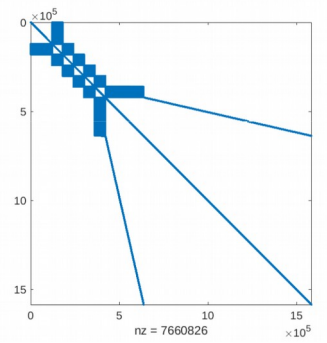


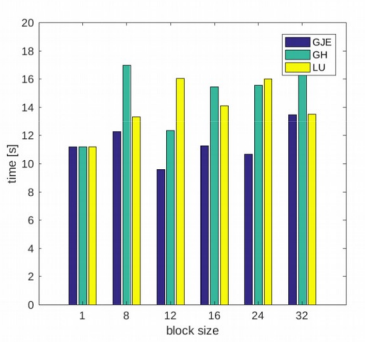
nd24k



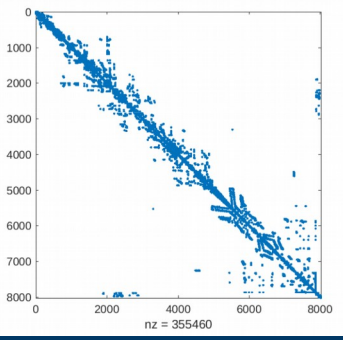


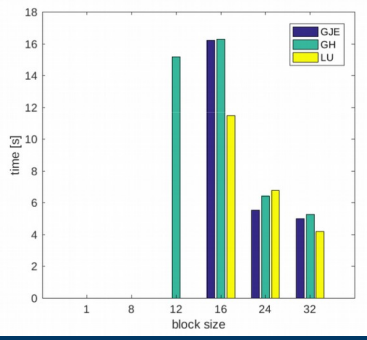
G3_circuit



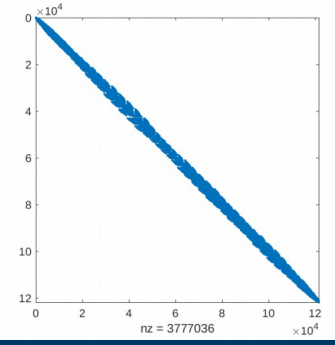


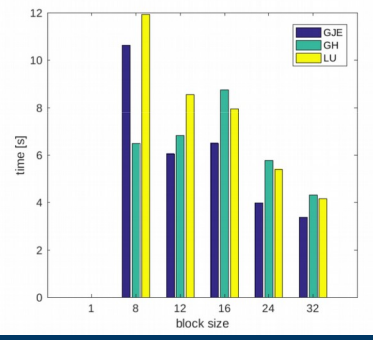
bcsstk38





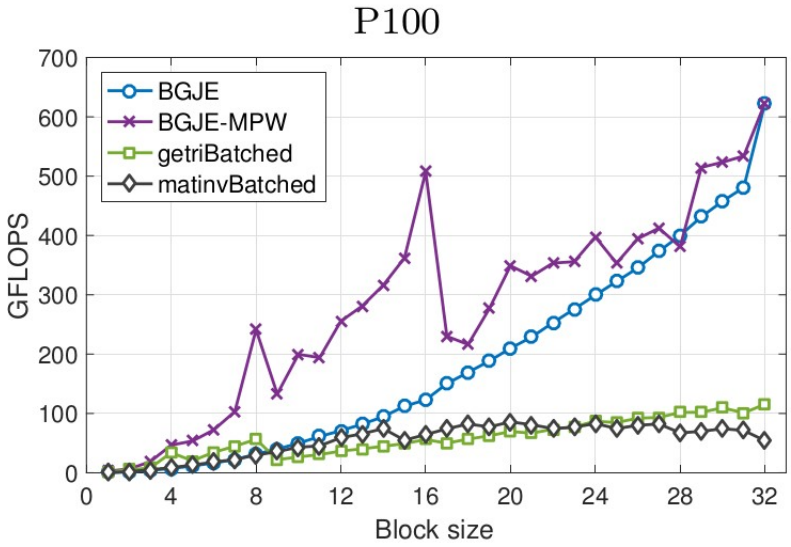
ship_003





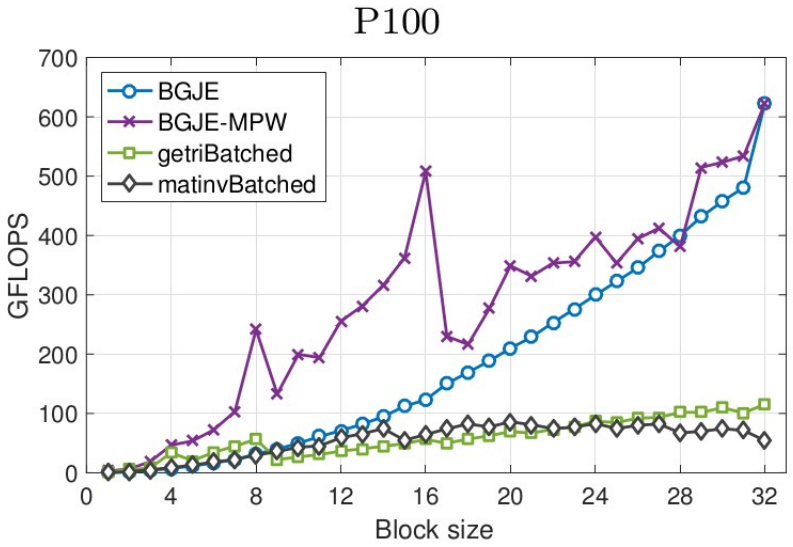
Flexible-size batched routines

- Problems can be to small to effectively use one warp
  - Solution: assign multiple problems per warp

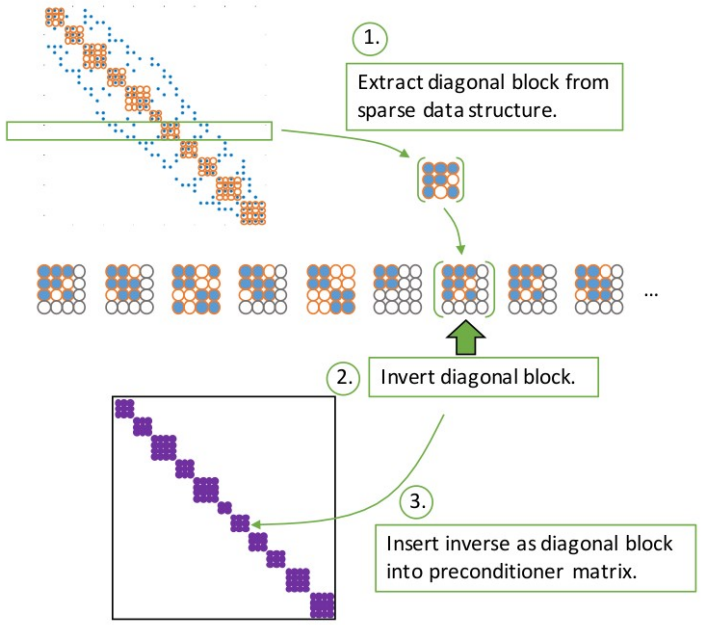


Flexible-size batched routines

- Problems can be to small to effectively use one warp
  - Solution: assign multiple problems per warp



- Allow batches where problems are of different sizes (flexible-size)
  - How to combine this with multiple problems per warp?
    - Remember: entire warp executes the same instruction!
    - Current solution: padding



Adaptive precision block-Jacobi

Preconditioner is an approximation of the system matrix

- Applying a preconditioner inherently carries an error
- For block-Jacobi the relative error of z is usually around 0.01 – 0.1

$$z := M^{-1}y \approx A^{-1}y$$

Anzt, et al. "Adaptive precision in block-Jacobi preconditioning for iterative sparse linear system solvers." Concurrency and Computation: Practice and Experience 31.6 (2019): e4460.

Adaptive precision block-Jacobi

Preconditioner is an approximation of the system matrix

- Applying a preconditioner inherently carries an error
- For block-Jacobi the relative error of z is usually around 0.01 – 0.1

$$z := M^{-1}y \approx A^{-1}y$$

Preconditioner application is memory bounded

- Most of the cost comes from reading the matrix from memory
- Idea: use lower precision to store the matrix

Anzt, et al. "Adaptive precision in block-Jacobi preconditioning for iterative sparse linear system solvers." Concurrency and Computation: Practice and Experience 31.6 (2019): e4460.

Adaptive precision block-Jacobi

Preconditioner is an approximation of the system matrix

- Applying a preconditioner inherently carries an error
- For block-Jacobi the relative error of z is usually around 0.01 – 0.1

$$z := M^{-1}y \approx A^{-1}y$$

Preconditioner application is memory bounded

- Most of the cost comes from reading the matrix from memory
- Idea: use lower precision to store the matrix

Adaptive precision in inversion-based block-Jacobi:

- All computation is done in double precision
- Preconditioner matrix is stored in lower precision, with roundoff error “u”
- Error bound:

$$\frac{||\delta z_i||}{||z_i||} \lesssim (c_i \kappa(D_i) u_d + u)\kappa(D_i)$$

Anzt, et al. "Adaptive precision in block-Jacobi preconditioning for iterative sparse linear system solvers." Concurrency and Computation: Practice and Experience 31.6 (2019): e4460.

Adaptive precision block-Jacobi

Preconditioner is an approximation of the system matrix

- Applying a preconditioner inherently carries an error
- For block-Jacobi the relative error of z is usually around 0.01 – 0.1

$$z := M^{-1}y \approx A^{-1}y$$

Preconditioner application is memory bounded

- Most of the cost comes from reading the matrix from memory
- Idea: use lower precision to store the matrix

Adaptive precision in inversion-based block-Jacobi:

- All computation is done in double precision
- Preconditioner matrix is stored in lower precision, with roundoff error “u”
- Error bound:

$$\frac{||\delta z_i||}{||z_i||} \lesssim (c_i \kappa(D_i) u_d + u)\kappa(D_i)$$

Assuming the preconditioner block is relatively well conditioned

- The error is determined by the product of u, and the condition number
- Choose the precision for each block **independently,** such that at least 1 digit of the result is correct

Experimental results

Determining the precision:

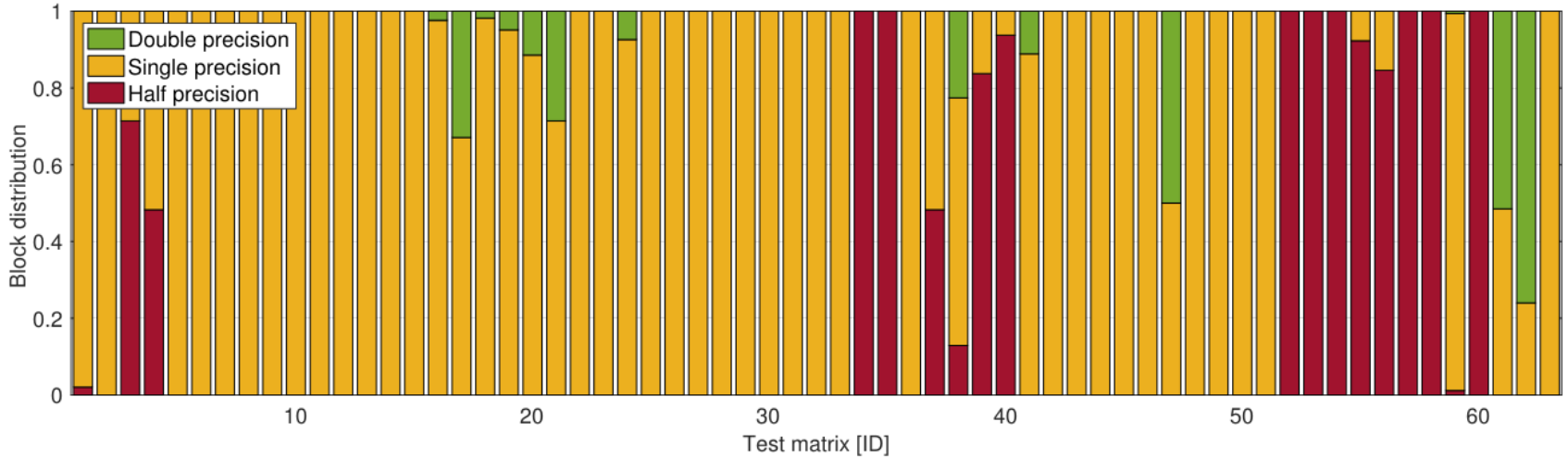
$$\begin{cases} \text{fp16} & \text{if } 0 < \kappa_1(D_i) \leq 10^2, \\ \text{fp32} & \text{if } 10^2 < \kappa_1(D_i) \leq 10^6, \text{ and} \\ \text{fp64} & \text{otherwise,} \end{cases}$$

Experimental results

Determining the precision:

$$\begin{cases} \text{fp16} & \text{if } 0 < \kappa_1(D_i) \leq 10^2, \\ \text{fp32} & \text{if } 10^2 < \kappa_1(D_i) \leq 10^6, \text{ and} \\ \text{fp64} & \text{otherwise,} \end{cases}$$

% of diagonal blocks stored in each precision *:

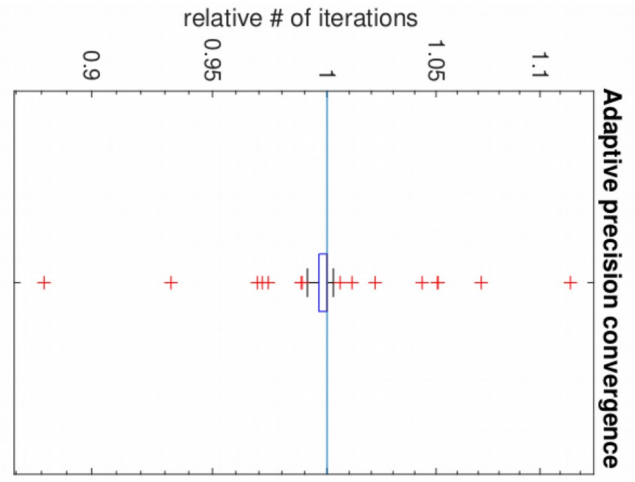


* Prototype implementation in MATLAB,
Results on 63 matrices from SuiteSparse

Experimental results

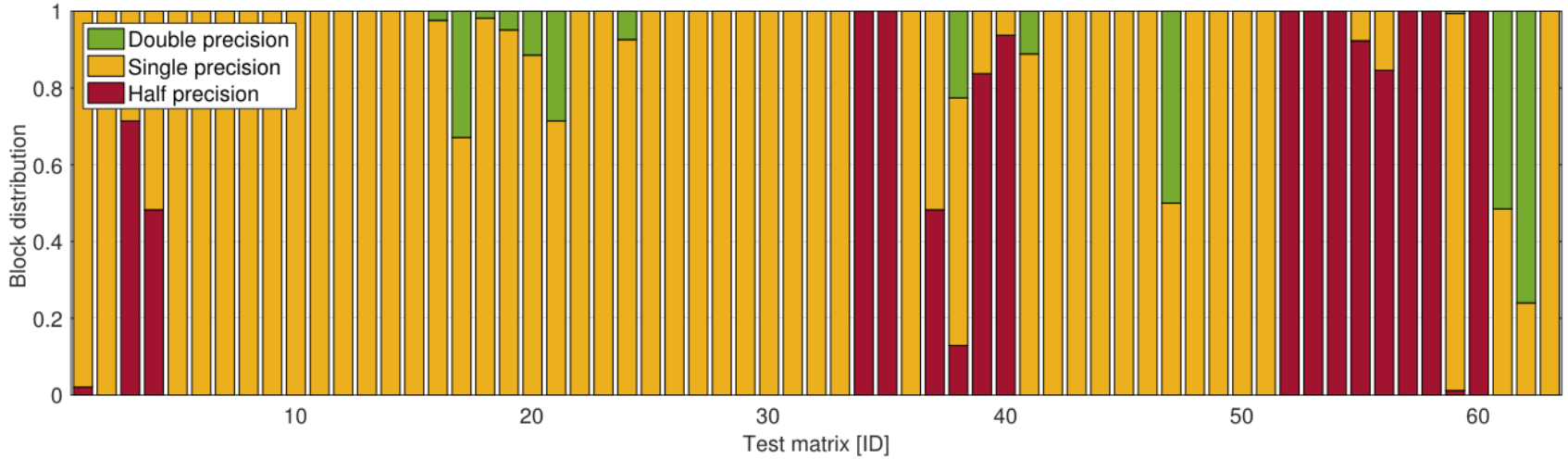
Determining the precision:

$$\begin{cases} \text{fp16} & \text{if } 0 < \kappa_1(D_i) \leq 10^2, \\ \text{fp32} & \text{if } 10^2 < \kappa_1(D_i) \leq 10^6, \text{ and} \\ \text{fp64} & \text{otherwise,} \end{cases}$$



Usually < 5% iteration increase

% of diagonal blocks stored in each precision *:



* Prototype implementation in MATLAB,
Results on 63 matrices from SuiteSparse

Overflow and underflow

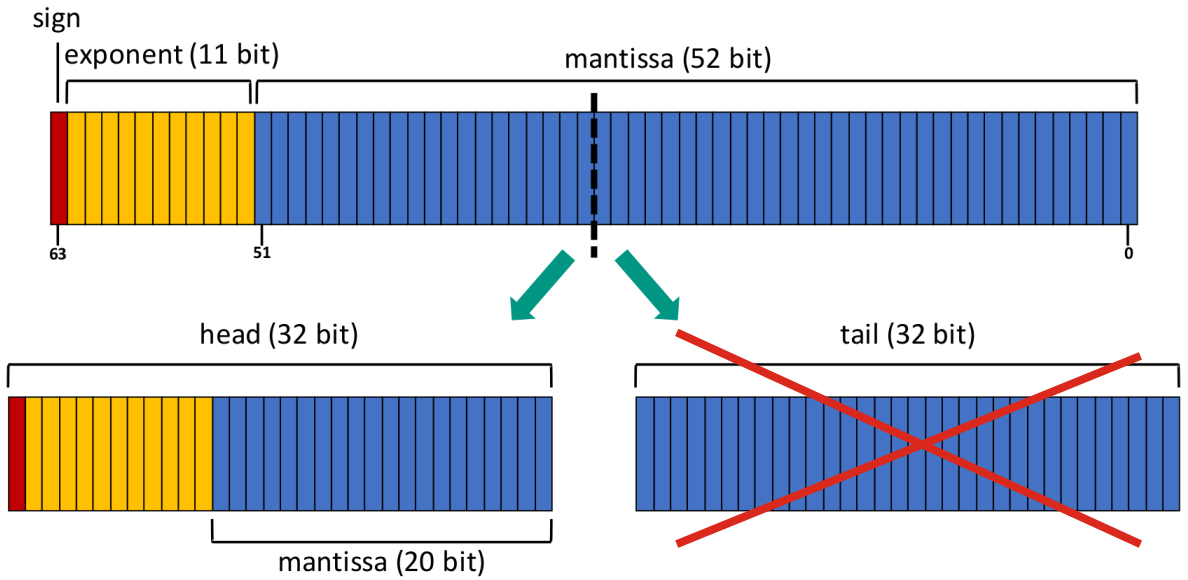
- Storing the block in lower precision can cause underflow or overflow in some values.
- This is not accounted for by the numerical analysis shown in the previous slides

Overflow and underflow

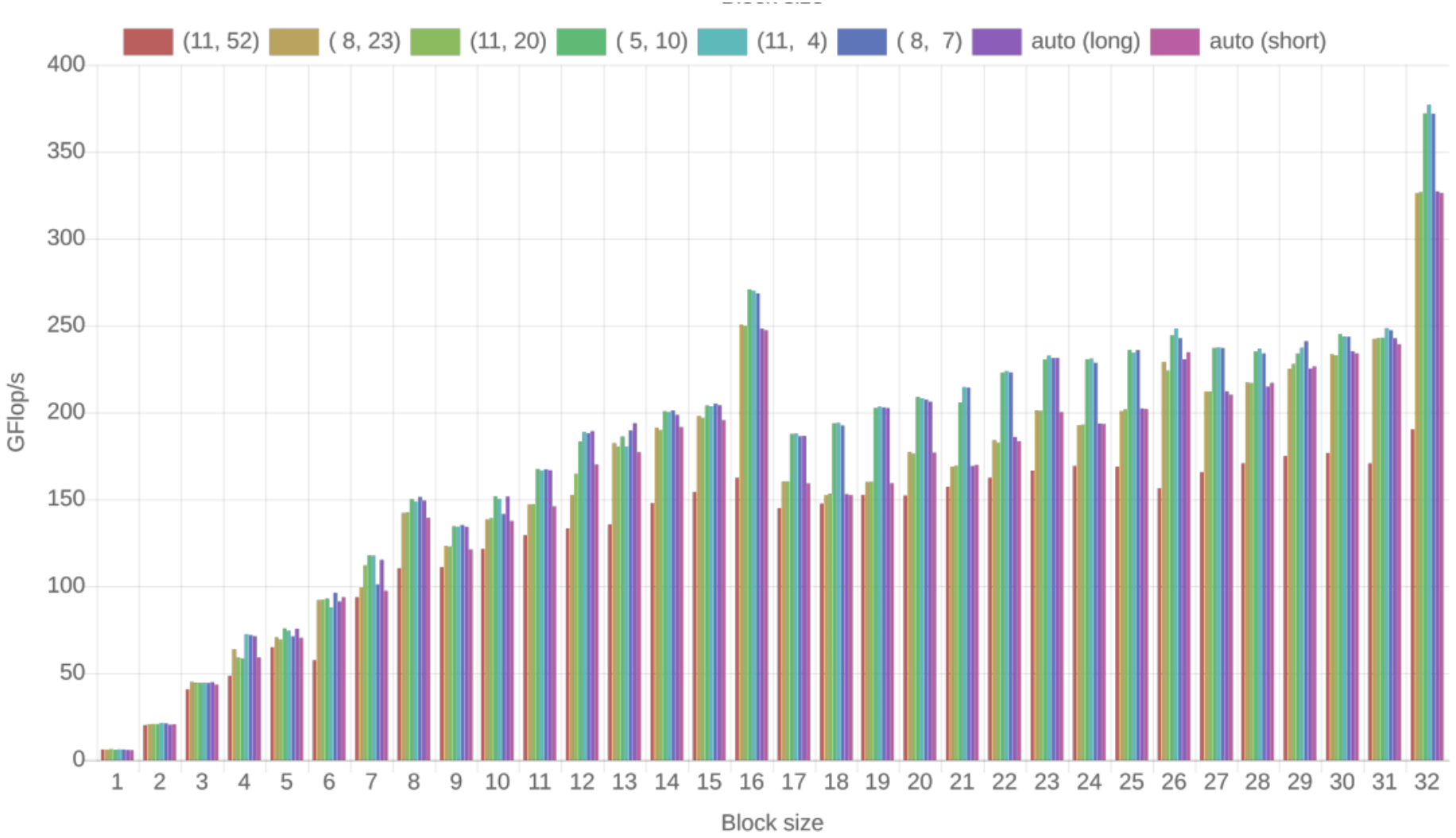
- Storing the block in lower precision can cause underflow or overflow in some values.
- This is not accounted for by the numerical analysis shown in the previous slides
- Two possible solutions:
  1. Check the condition number of the low precision block to verify there were no catastrophic overflows or underflows.

Overflow and underflow

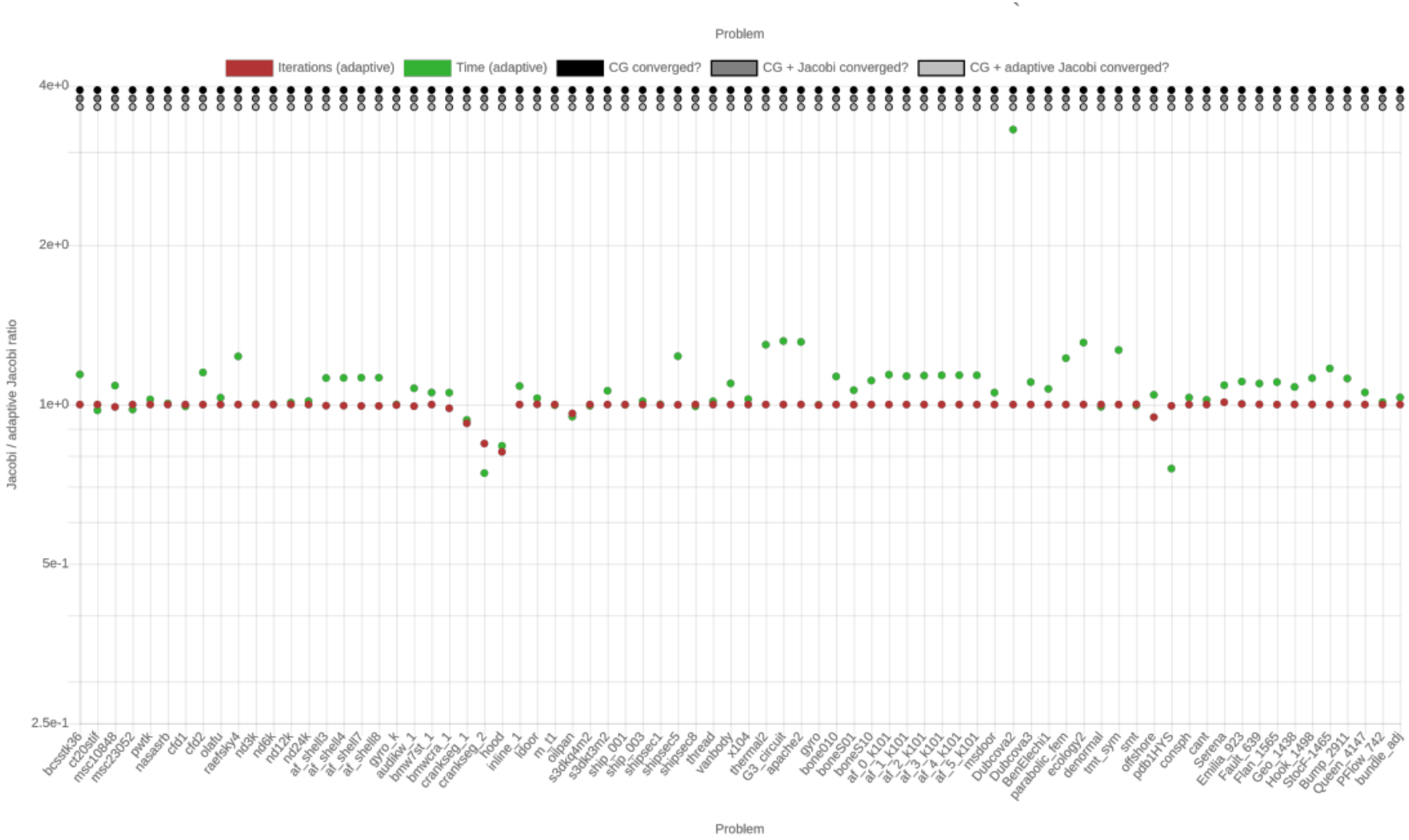
- Storing the block in lower precision can cause underflow or overflow in some values.
- This is not accounted for by the numerical analysis shown in the previous slides
- Two possible solutions:
  1. Check the condition number of the low precision block to verify there were no catastrophic overflows or underflows.
  2. Use a custom storage format that preserves the number of exponent bits:



Preconditioner application performance



Time-to-solution, adaptive- vs full-precision BJ



Summary

- Employ inherently parallel preconditioning schemes on accelerator hardware
- Trade more computation for better access patterns
- Take advantage of large register field of NVIDIA GPUs
- Avoid data exchange by “renaming” CUDA threads
- Take advantage of low precision to accelerate preconditioner application
- Independently select precision for each subproblem to preserve preconditioner quality

KIT, Karlsruhe, 12 June 2019

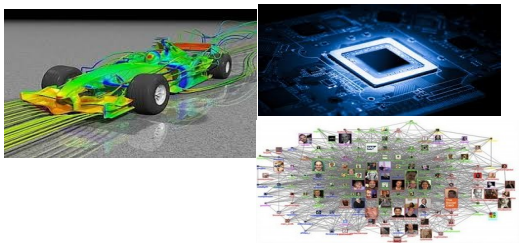
UNIVERSITAT JAUME I

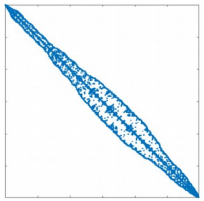
Linear systems, Ginkgo

flegar@uji.es

Sources of linear systems

- Real-world problem transformed into a linear system via:
  - PDE discretizations, graph representations
  - Large number of unknowns (1M+, full matrix 8TB)
  - Most matrix elements are zero
- Possible approach: iterative methods
  - Krylov-subspace based linear solvers
    - SpMV
    - BLAS-1 operations
  - Sparse matrix formats & SpMV
    - accelerate each iteration of the solver
  - Preconditioners
    - reduce the number of iterations

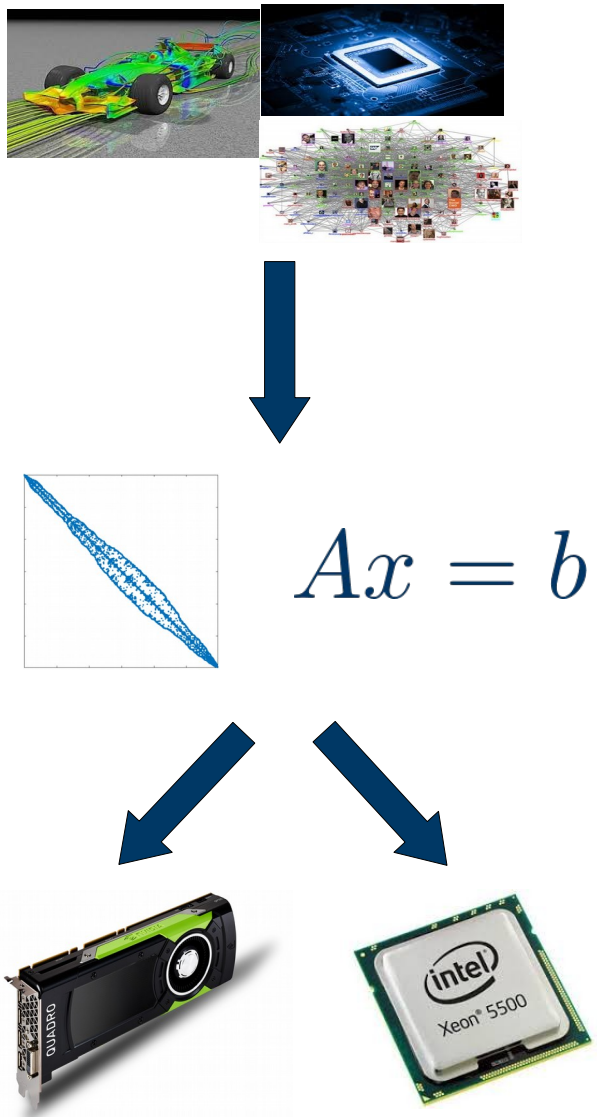




$$Ax = b$$

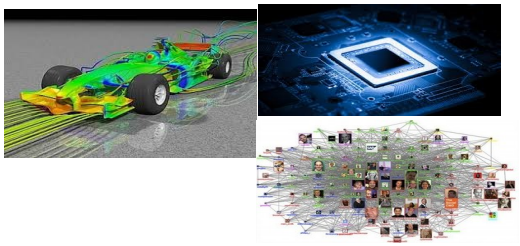
Sources of linear systems

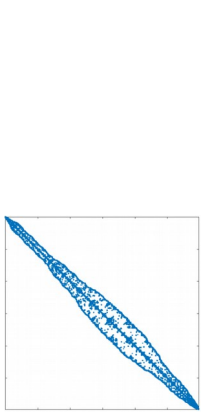
- Real-world problem transformed into a linear system via:
  - PDE discretizations, graph representations
  - Large number of unknowns (1M+, full matrix 8TB)
  - Most matrix elements are zero
- Possible approach: iterative methods
  - Krylov-subspace based linear solvers
    - SpMV
    - BLAS-1 operations
  - Sparse matrix formats & SpMV
    - accelerate each iteration of the solver
  - Preconditioners
    - reduce the number of iterations
  - Special hardware (e.g. GPUs)
  - Probably not a good idea to implement everything from scratch…



Sources of linear systems

- Real-world problem transformed into a linear system via:
  - PDE discretizations, graph representations
  - Large number of unknowns (1M+, full matrix 8TB)
  - Most matrix elements are zero
- Possible approach: iterative methods
  - Krylov-subspace based linear solvers
    - SpMV
    - BLAS-1 operations
  - Sparse matrix formats & SpMV
    - accelerate each iteration of the solver
  - Preconditioners
    - reduce the number of iterations
  - Special hardware (e.g. GPUs)
  - Probably not a good idea to implement everything from scratch…
  - Use a library instead: Ginkgo





$$Ax = b$$

Ginkgo

KIT, Karlsruhe, 12 June 2019

UNIVERSITAT JAUME I

Preconditioning

flegar@uji.es

Preconditioning

$$Ax = b, \;\; A \in \mathbb{R}^{n \times n}$$

Preconditioning

$$Ax = b,\ A \in \mathbb{R}^{n \times n} \longrightarrow M^{-1}Ax = M^{-1}b$$

Replace the original system with an equivalent preconditioned system

Preconditioning

$$Ax = b,\ A \in \mathbb{R}^{n \times n} \longrightarrow M^{-1}Ax = M^{-1}b$$

Replace the original system with an equivalent preconditioned system

$$M \approx A \qquad M^{-1} \text{ easy to compute}$$

Preconditioning

$$Ax = b,\ A \in \mathbb{R}^{n \times n} \quad \Longrightarrow \quad M^{-1}Ax = M^{-1}b$$

Replace the original system with an equivalent preconditioned system

$$M \approx A \qquad M^{-1} \text{ easy to compute}$$

~~$M^{-1}A$~~

Do not compute the preconditioned system matrix explicitly!

Preconditioning

$$Ax = b,\ A \in \mathbb{R}^{n \times n} \quad \Longrightarrow \quad M^{-1}Ax = M^{-1}b$$

Replace the original system with an equivalent preconditioned system

$$M \approx A \qquad M^{-1} \text{ easy to compute}$$

~~$M^{-1}A$~~

$$y := (M^{-1}A)x$$

Do not compute the preconditioned system matrix explicitly!

Preconditioning

$$Ax = b,\ A \in \mathbb{R}^{n \times n} \quad \Rightarrow \quad M^{-1}Ax = M^{-1}b$$

Replace the original system with an equivalent preconditioned system

$$M \approx A \qquad M^{-1} \text{ easy to compute}$$

~~$M^{-1}A$~~

Do not compute the preconditioned system matrix explicitly!

$$y := (M^{-1}A)x$$

$$z := Ax$$

$$y := M^{-1}z$$

Preconditioner application

Preconditioning

$$Ax = b,\ A \in \mathbb{R}^{n \times n} \quad \Rightarrow \quad M^{-1}Ax = M^{-1}b$$

Replace the original system with an equivalent preconditioned system

$$M \approx A \qquad M^{-1} \text{ easy to compute}$$

~~$M^{-1}A$~~

Do not compute the preconditioned system matrix explicitly!

$$y := (M^{-1}A)x$$

$$z := Ax$$

$$y := M^{-1}z$$

Generate the preconditioner matrix, and store it in a form suitable for application

$$A \rightsquigarrow M$$

Preconditioner setup

Preconditioner application

Preconditioning

$$Ax = b,\ A \in \mathbb{R}^{n \times n} \quad \longrightarrow \quad M^{-1}Ax = M^{-1}b$$

Replace the original system with an equivalent preconditioned system

$$M \approx A \qquad M^{-1} \text{ easy to compute}$$

~~$M^{-1}A$~~

Do not compute the preconditioned system matrix explicitly!

$$y := (M^{-1}A)x$$

$$z := Ax$$

$$y := M^{-1}z$$

Generate the preconditioner matrix, and store it in a form suitable for application

$$A \rightsquigarrow M$$

Preconditioner setup

Preconditioner application

Trade-off:
faster convergence,
but more work per iteration