

Accelerating the Solution of Sparse Linear Systems Using GPUs

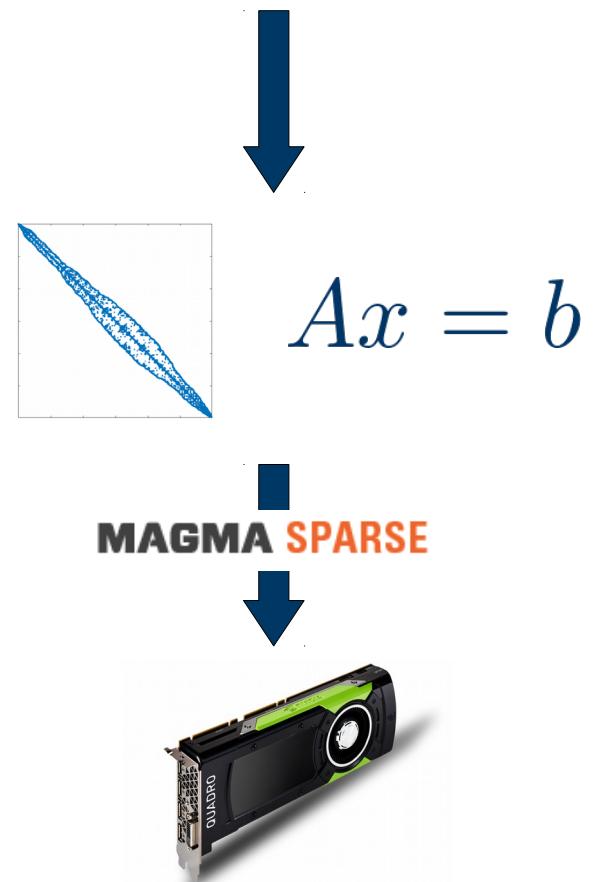
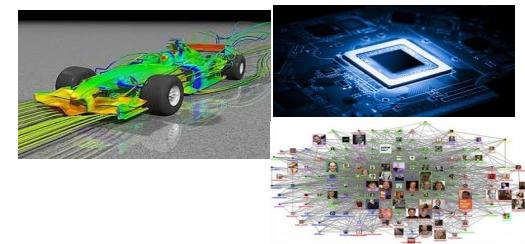
Goran Flegar

Joint work with: Hartwig Anzt, Nicholas Higham, Enrique S. Quintana-Ortí



MAGMA-sparse software library

- GPU-accelerated sparse linear algebra library
 - Focus: linear systems

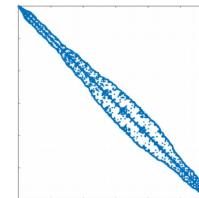
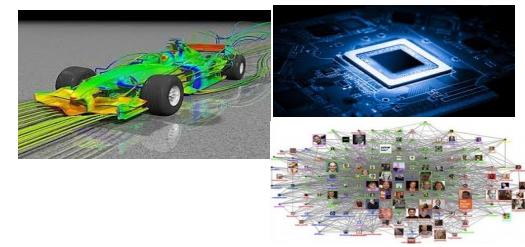


Joint effort: Innovative Computing Lab at University of Tennessee, Knoxville; Karlsruhe Institute of Technology; Universidad Jaume I



MAGMA-sparse software library

- GPU-accelerated sparse linear algebra library
 - Focus: linear systems
 - Iterative, Krylov-subspace based linear solvers
 - SpMV
 - BLAS-1 operations



$$Ax = b$$

MAGMA SPARSE

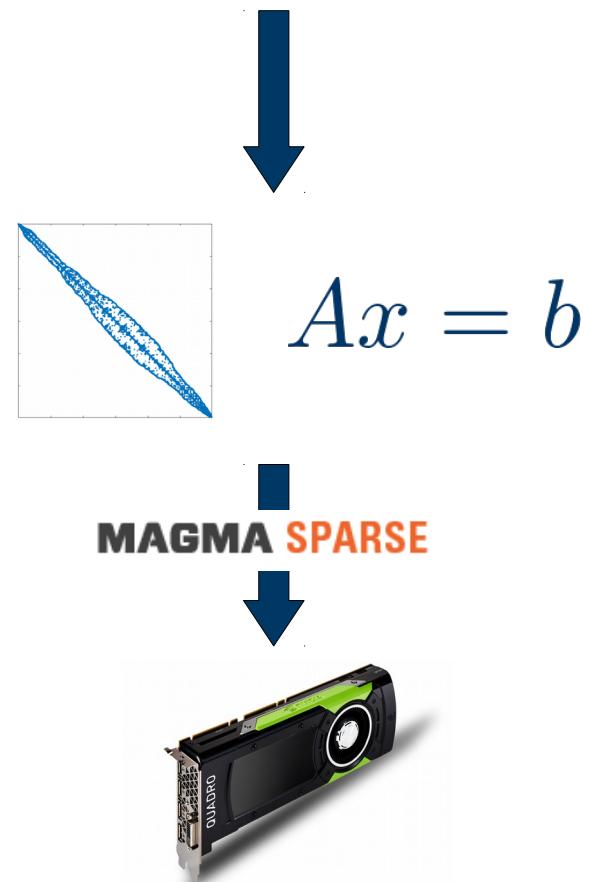
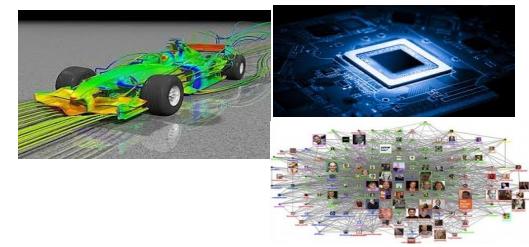


Joint effort: Innovative Computing Lab at University of Tennessee, Knoxville; Karlsruhe Institute of Technology; University Jaume I



MAGMA-sparse software library

- GPU-accelerated sparse linear algebra library
 - Focus: linear systems
 - Iterative, Krylov-subspace based linear solvers
 - SpMV
 - BLAS-1 operations
 - Sparse matrix formats & SpMV
 - accelerate each iteration of the solver



Joint effort: Innovative Computing Lab at University of Tennessee, Knoxville; Karlsruhe Institute of Technology; University Jaume I



THE UNIVERSITY OF
TENNESSEE
KNOXVILLE



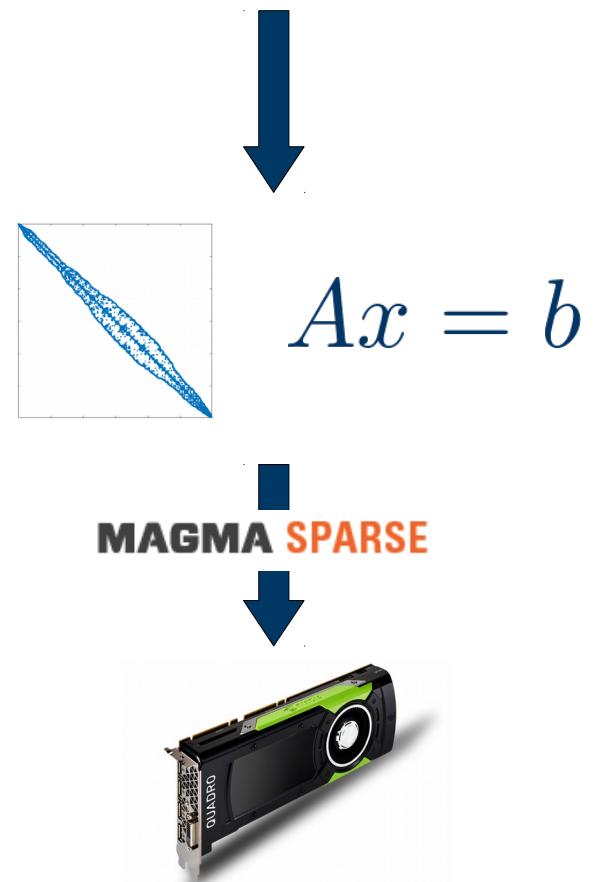
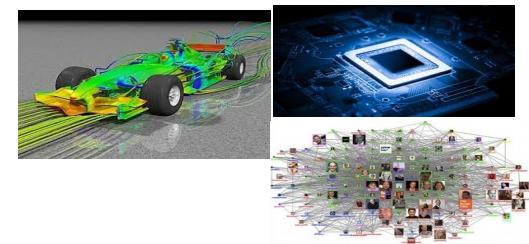
Karlsruher Institut für Technologie



UNIVERSITAT
JAUME I

MAGMA-sparse software library

- GPU-accelerated sparse linear algebra library
 - Focus: linear systems
 - Iterative, 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



Joint effort: Innovative Computing Lab at University of Tennessee, Knoxville; Karlsruhe Institute of Technology; University Jaume I



THE UNIVERSITY OF
TENNESSEE
KNOXVILLE



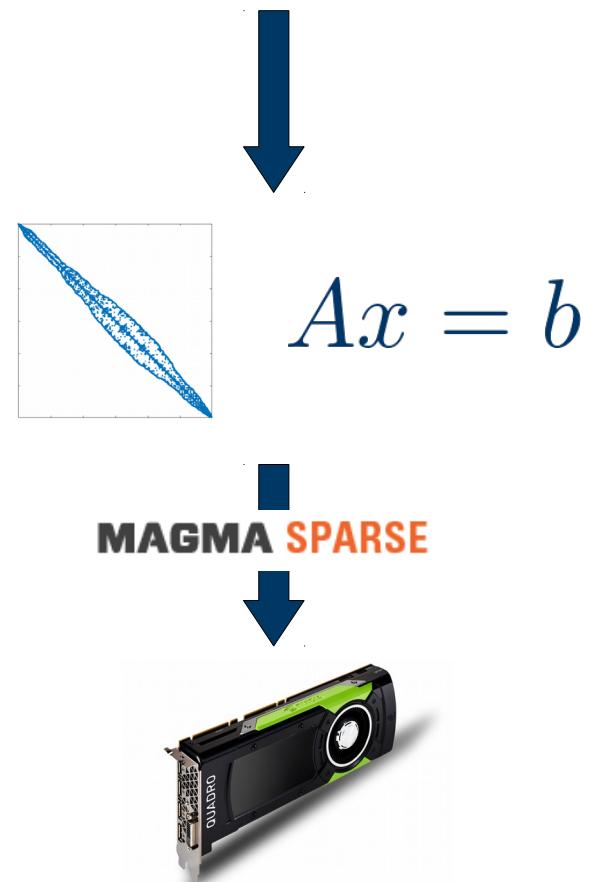
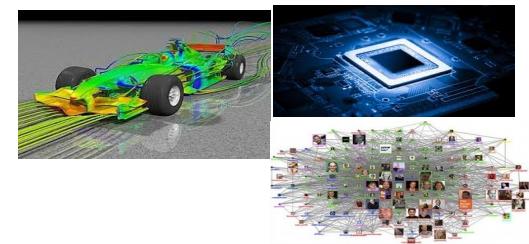
Karlsruher Institut für Technologie



UNIVERSITAT
JAUME I

MAGMA-sparse software library

- GPU-accelerated sparse linear algebra library
 - Focus: linear systems
 - Iterative, 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



Joint effort: Innovative Computing Lab at University of Tennessee, Knoxville; Karlsruhe Institute of Technology; Universitat Jaume I



Preconditioning

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

Preconditioning

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

Replace the original system with an equivalent preconditioned system

Preconditioning

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

Replace the original system with an equivalent preconditioned system

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

Preconditioning

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

Replace the original system with an equivalent preconditioned system

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

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

Do not compute the preconditioned system matrix explicitly!

Preconditioning

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

Replace the original system with an equivalent preconditioned system

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

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

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

Do not compute the preconditioned system matrix explicitly!

Preconditioning

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

Replace the original system with an equivalent preconditioned system

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

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

Do not compute the preconditioned system matrix explicitly!

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



$$\boxed{\begin{aligned} z &:= Ax \\ y &:= M^{-1}z \end{aligned}}$$

Preconditioner application

Preconditioning

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

Replace the original system with an equivalent preconditioned system

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

$$\cancel{M^{-1}A}$$

Do not compute the preconditioned system matrix explicitly!

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

$$A \rightsquigarrow M$$

Preconditioner setup

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



$$z := Ax$$

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

Preconditioner application

Preconditioning

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

Replace the original system with an equivalent preconditioned system

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

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

Do not compute the preconditioned system matrix explicitly!

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



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

$$A \rightsquigarrow M$$

Preconditioner setup

$$z := Ax$$

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

Preconditioner application

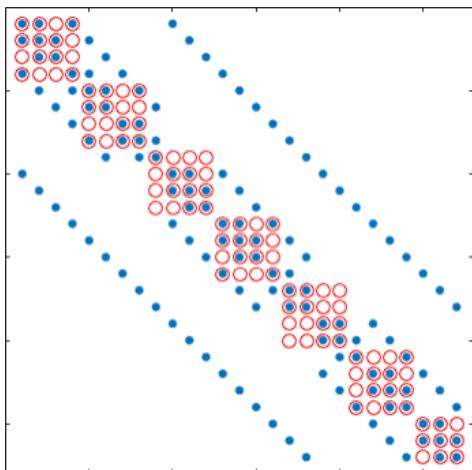
Trade-off:
faster convergence,
but more work per iteration

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

- 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

$$\text{diag}(A) = [D_1, \dots, D_k]$$

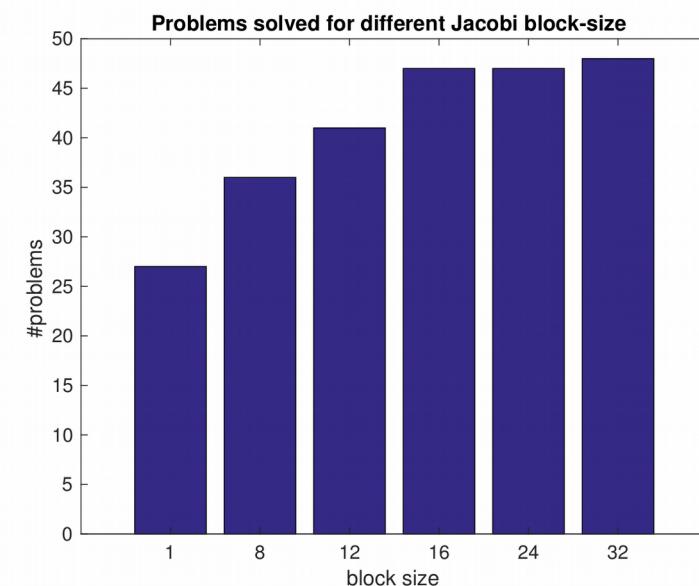
$$M := \text{diag}(D_1, \dots, D_k)$$

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

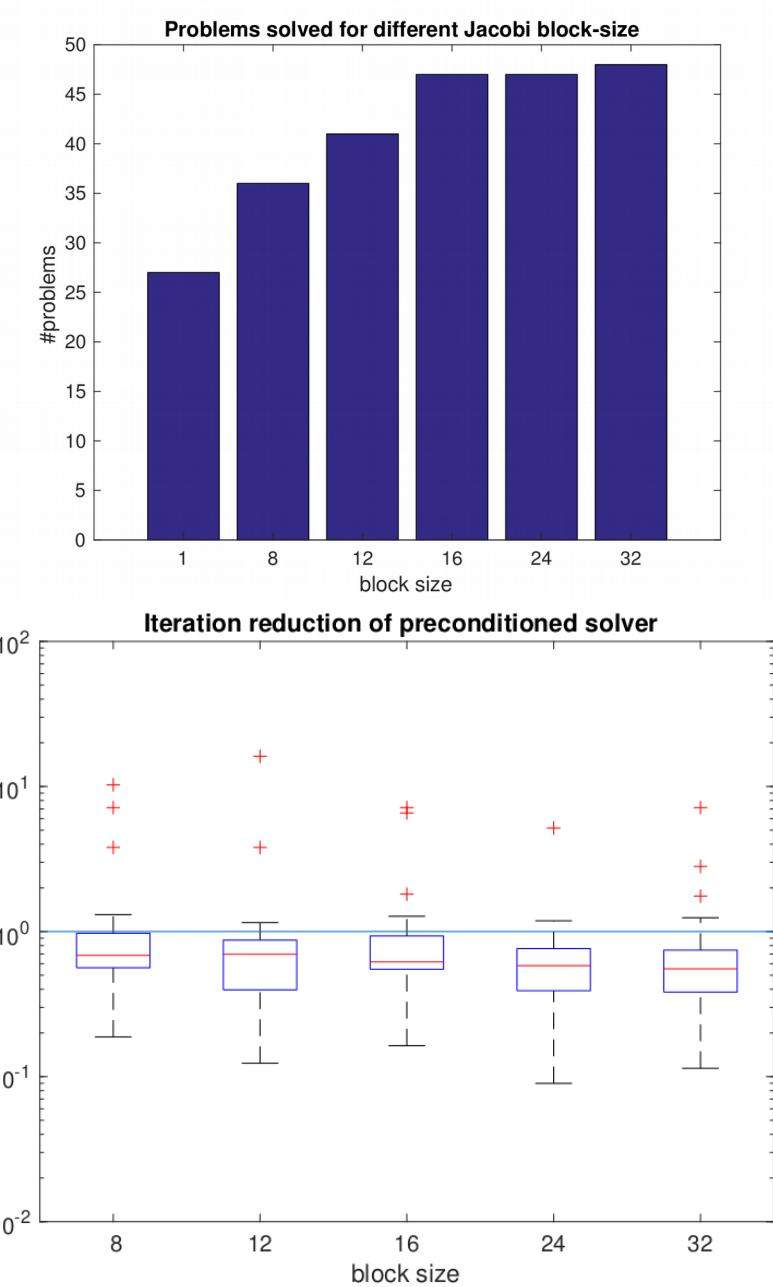
Benefits of block-Jacobi

- 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



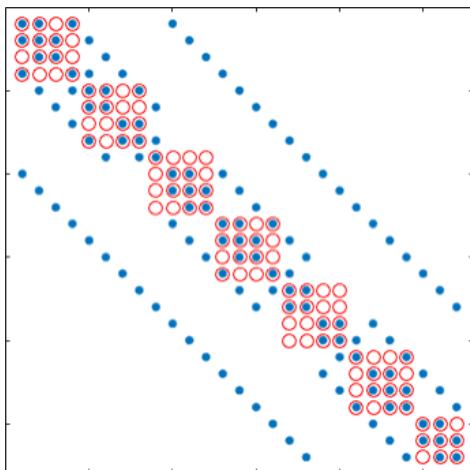
Benefits of block-Jacobi

- 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



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

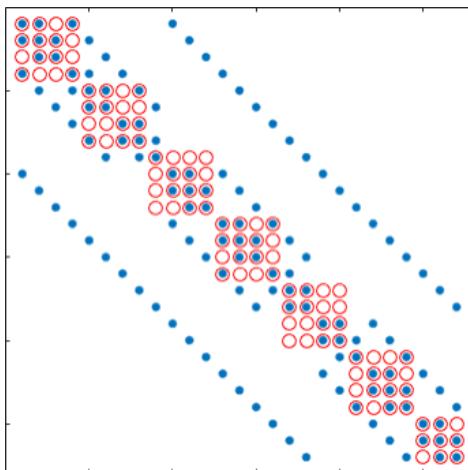
$$\text{diag}(A) = [D_1, \dots, D_k]$$

$$M := \text{diag}(D_1, \dots, D_k)$$

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

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

$$\text{diag}(A) = [D_1, \dots, D_k]$$

$$M := \text{diag}(D_1, \dots, D_k)$$

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

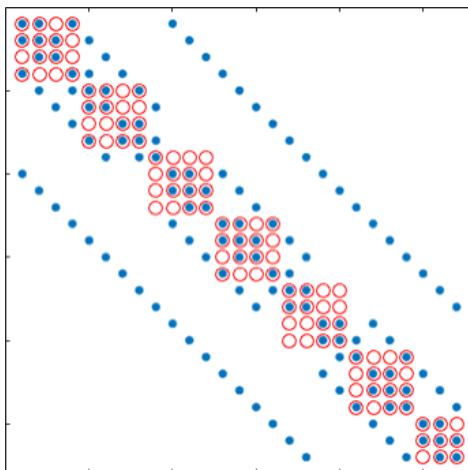
$$\tilde{D}_i := \text{inv}(D_i)$$

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

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

$$\text{diag}(A) = [D_1, \dots, D_k]$$

$$M := \text{diag}(D_1, \dots, D_k)$$

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

$$\tilde{D}_i := \text{inv}(D_i)$$

Setup

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

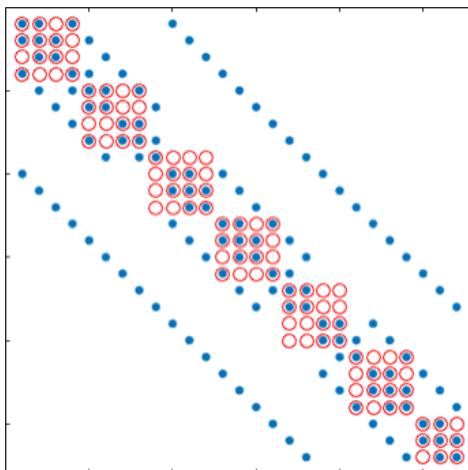
Application

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

$$\text{diag}(A) = [D_1, \dots, D_k]$$

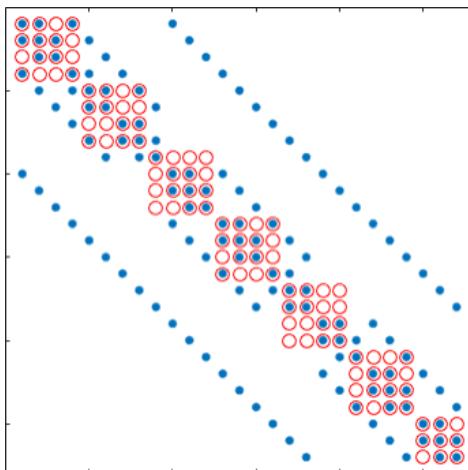
$$M := \text{diag}(D_1, \dots, D_k)$$

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

$$D_i y_i = z_i$$

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

$$\text{diag}(A) = [D_1, \dots, D_k]$$

$$M := \text{diag}(D_1, \dots, D_k)$$

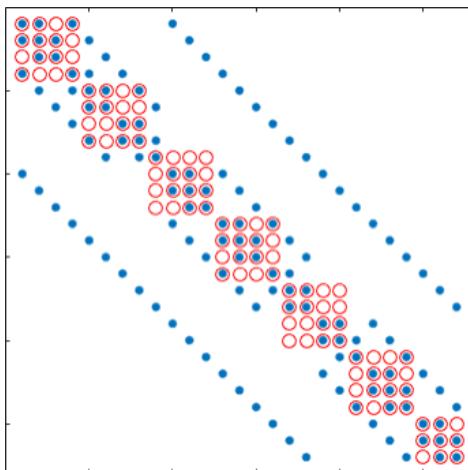
$$y := M^{-1}z \longrightarrow y_i := D_i^{-1}z_i, \quad \forall i$$

$$D_i = L_i U_i$$

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

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

$$\text{diag}(A) = [D_1, \dots, D_k]$$

$$M := \text{diag}(D_1, \dots, D_k)$$

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

$$D_i = L_i U_i$$

Setup

$$D_i y_i = z_i \longrightarrow$$

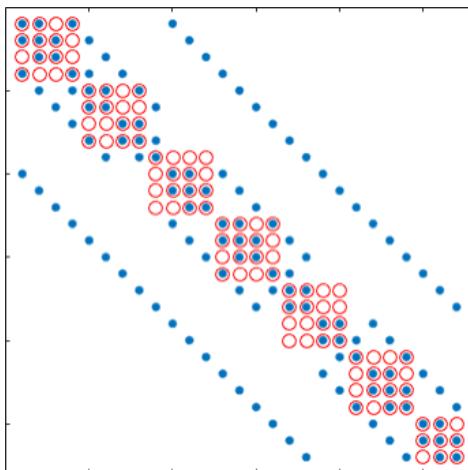
$$U_i y_i = w_i$$
$$L_i w_i = z_i$$

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

$$\text{diag}(A) = [D_1, \dots, D_k]$$

$$M := \text{diag}(D_1, \dots, D_k)$$

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

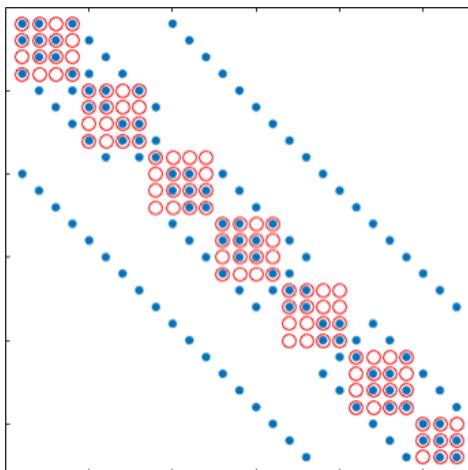
$$\tilde{D}_i := \text{gh}_d(D_i)$$

$$\longrightarrow D_i y_i = z_i \longrightarrow z_i := \text{gh}_a(\tilde{D}_i, y_i)$$

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

$$\text{diag}(A) = [D_1, \dots, D_k]$$

$$M := \text{diag}(D_1, \dots, D_k)$$

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

$$\tilde{D}_i := \text{gh}_d(D_i)$$

Setup

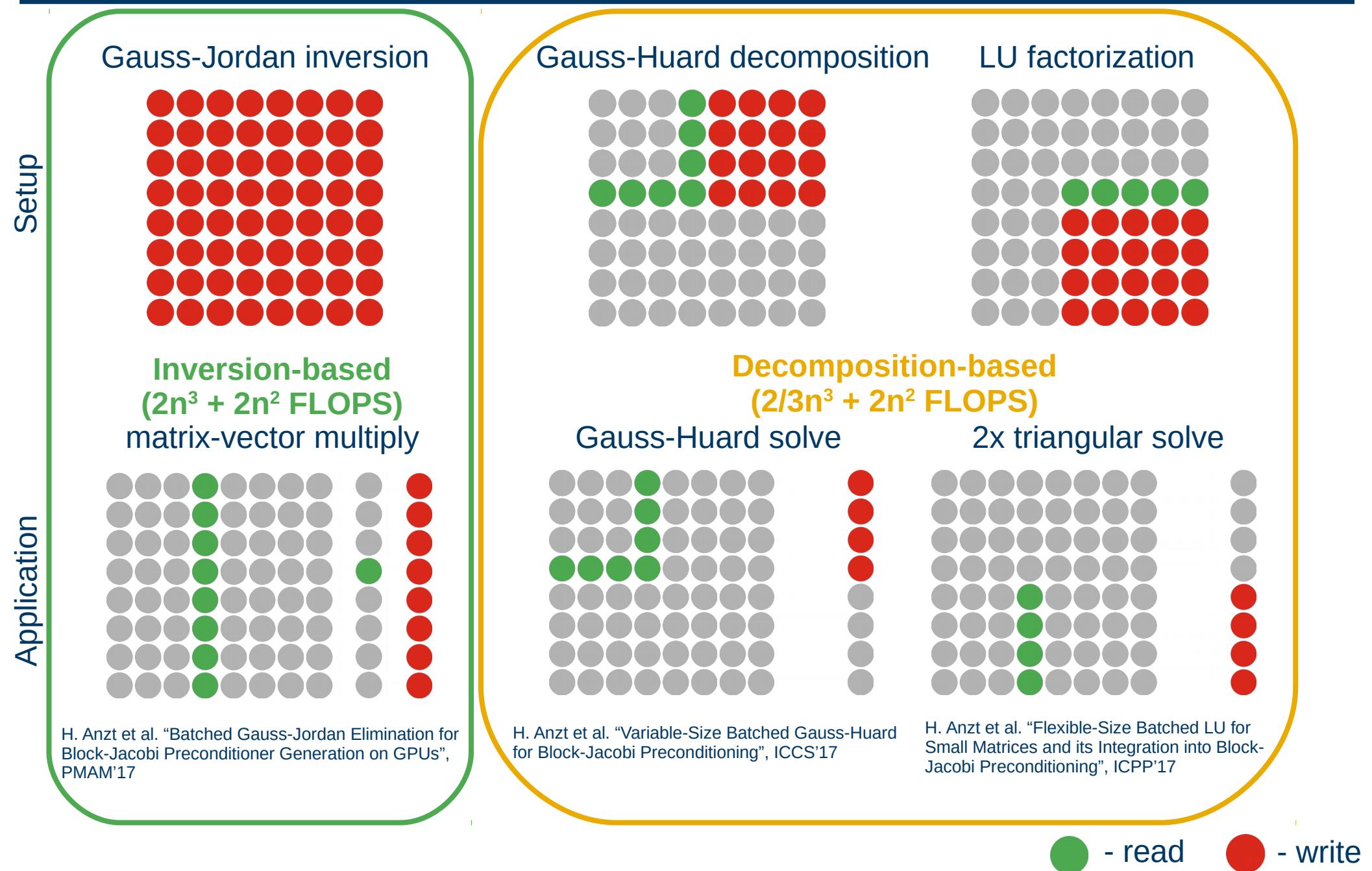
$$z_i := \text{gh}_a(\tilde{D}_i, y_i)$$

Application

$$D_i y_i = z_i \longrightarrow$$

Decomposition-based block-Jacobi

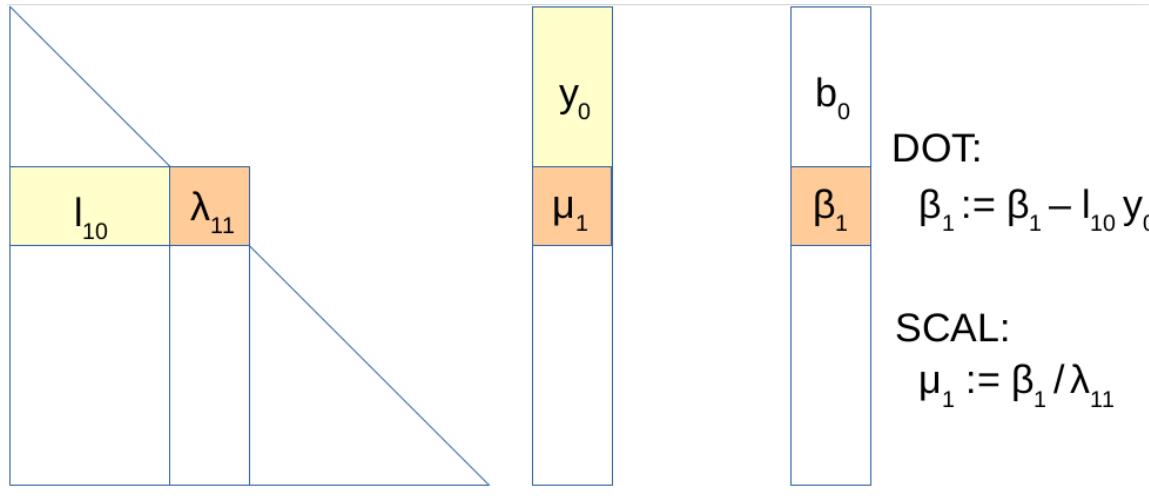
Block-Jacobi setup & application ecosystem



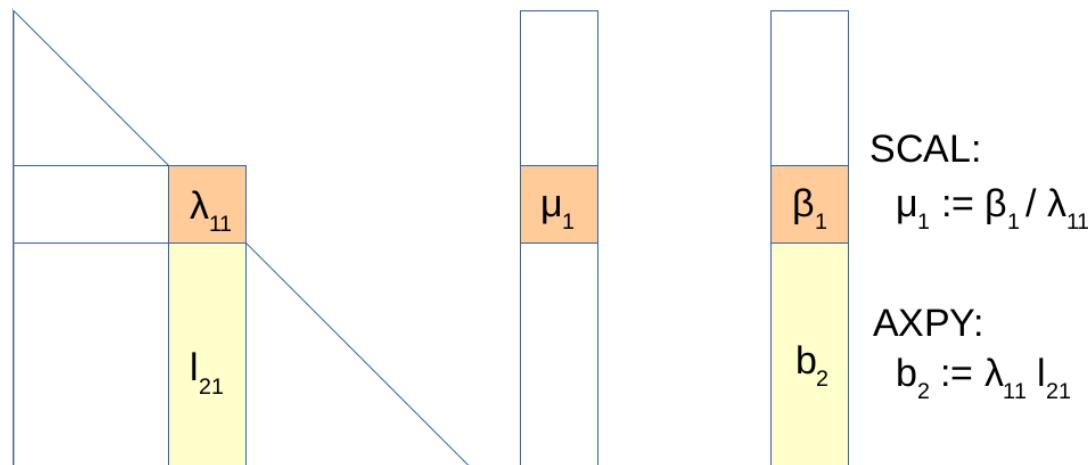
MAGMA-sparse triangular solves

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

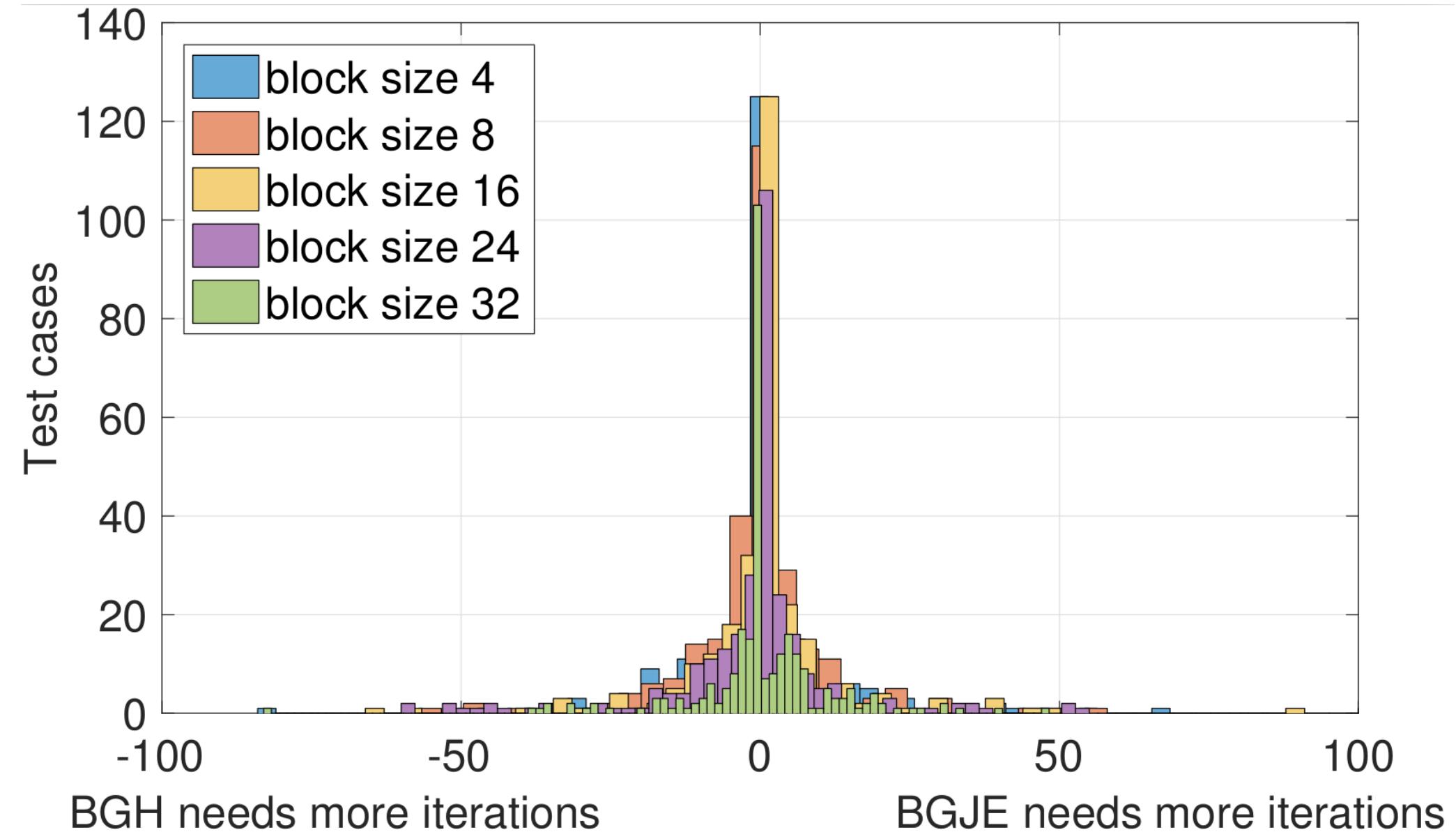
“Lazy” triangular solve



“Eager” triangular solve



Inversion?!



GPU programming 101

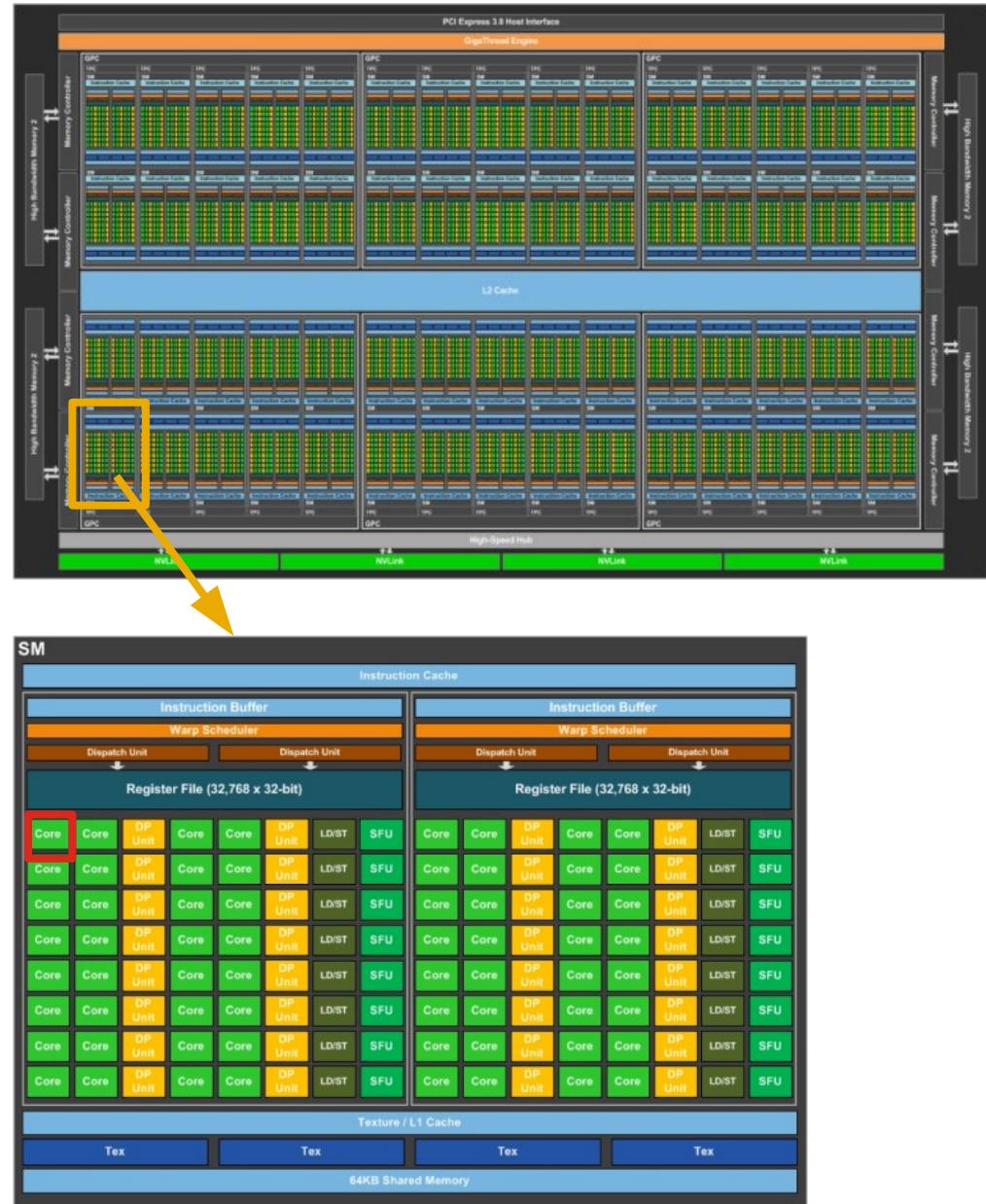
- NVIDIA P100 GPU
 - 4.7 TFLOPs DP performance
 - Up to 740 GB/s (1 : 51)
 - 56 SMs x 64 cores = 3584 cores!



source: devblogs.nvidia.com/parallelforall/

GPU programming 101

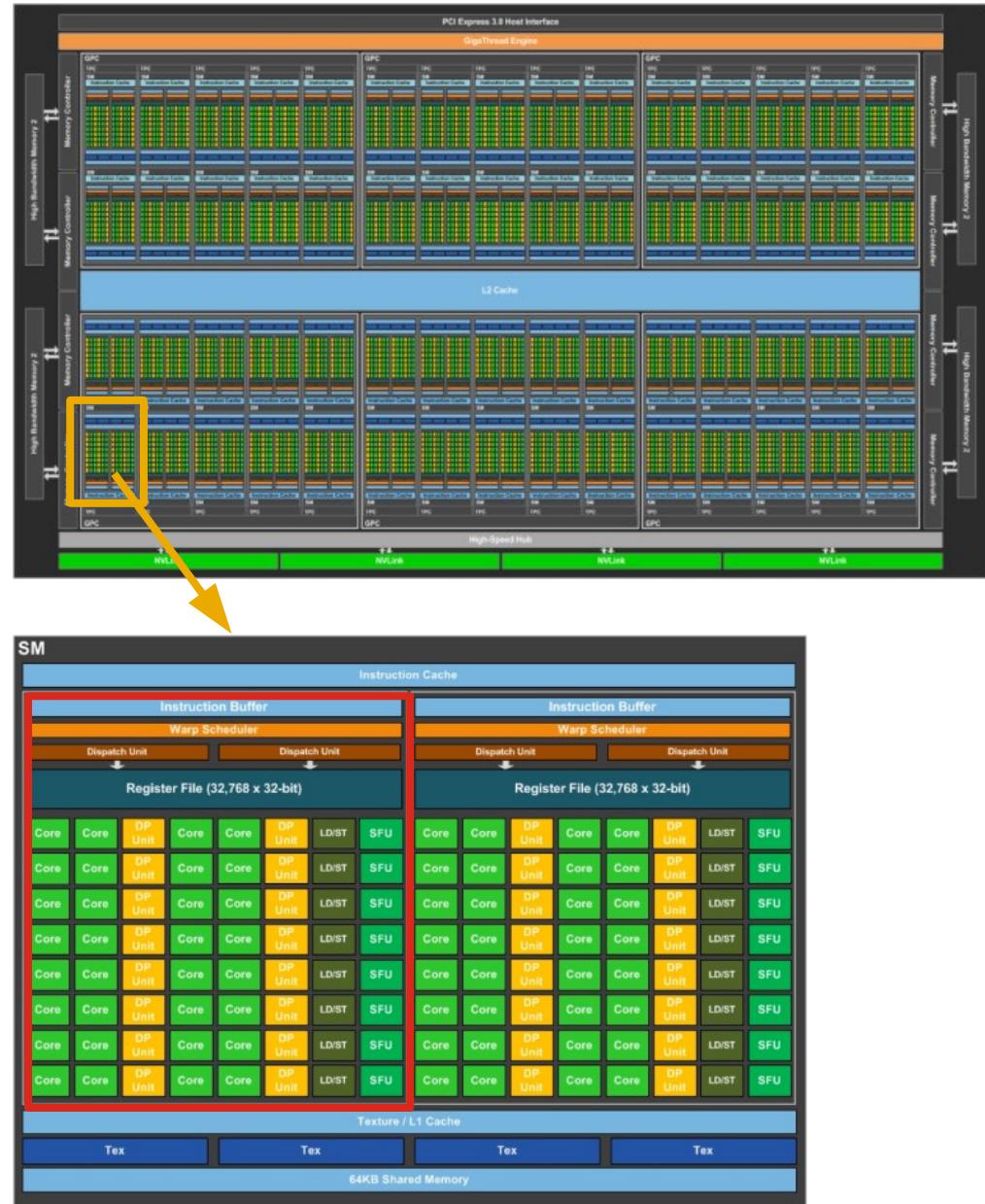
- NVIDIA P100 GPU
 - 4.7 TFLOPs DP performance
 - Up to 740 GB/s (1 : 51)
 - 56 SMs x 64 cores = 3584 cores!
- Hierarchical architecture / programming model:
 - Thread
 - Basic building block, assigned to 1 core



source: devblogs.nvidia.com/parallelforall/

GPU programming 101

- NVIDIA P100 GPU
 - 4.7 TFLOPs DP performance
 - Up to 740 GB/s (1 : 51)
 - 56 SMs x 64 cores = 3584 cores!
- Hierarchical architecture / programming model:
 - Thread
 - Basic building block, assigned to 1 core
 - Warp
 - Group of 32 threads
 - Perfectly synchronized execution
 - Can share values directly from the registers (1KB / thread)
 - Cannot execute different instructions (warp divergence)

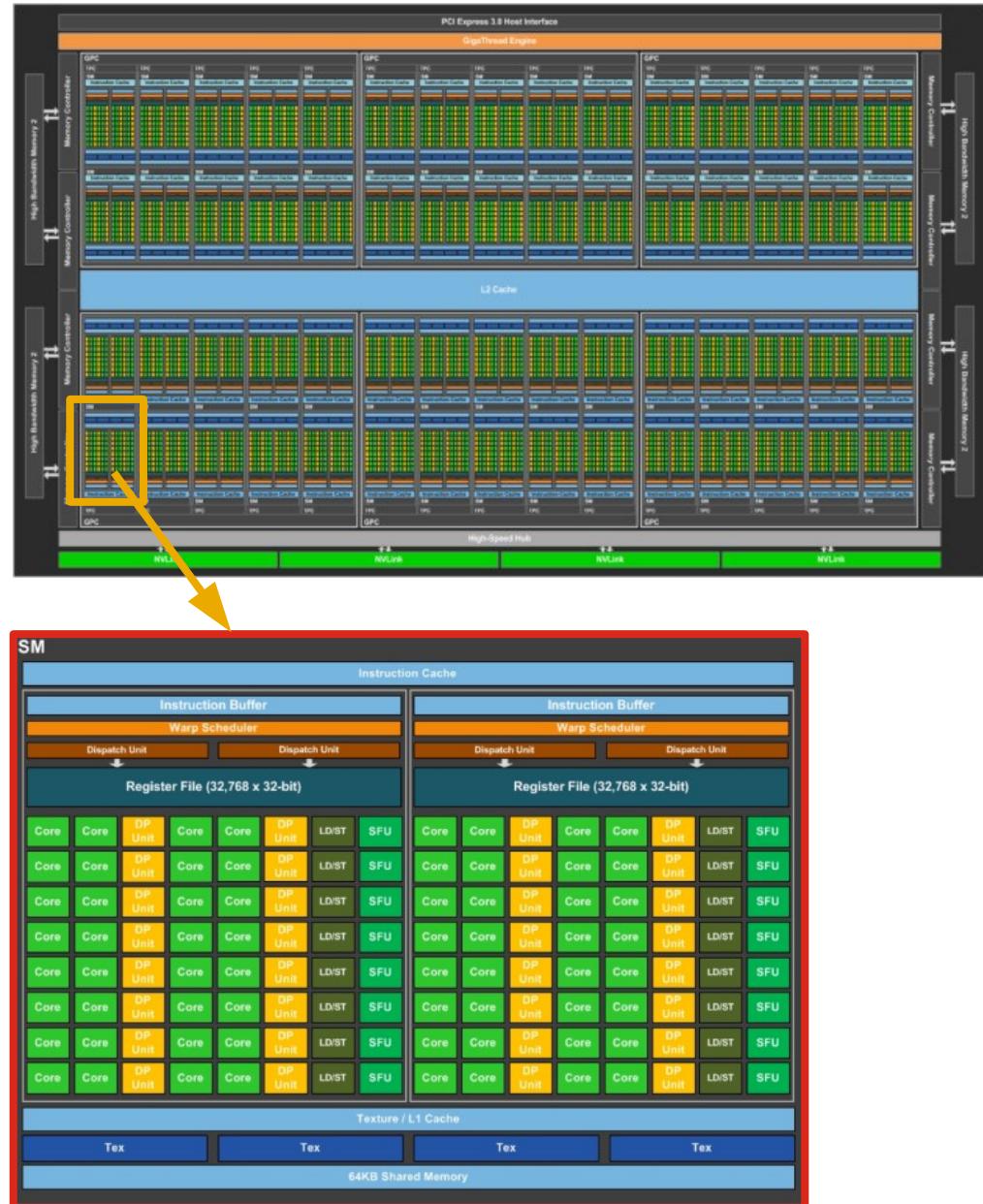


source: devblogs.nvidia.com/parallelforall/

GPU programming 101

- Block

- Group of several warps (≤ 64)
- Can be explicitly synchronized
- Can share data via shared memory (64KB)



source: devblogs.nvidia.com/parallelforall/

GPU programming 101

- Block
 - Group of several warps (≤ 64)
 - Can be **explicitly synchronized**
 - Can **share data via shared memory** (64KB)
- Grid
 - Group of blocks
 - **Cannot synchronize!**
 - Global memory (12 or 16GB)
 - Simple caches
 - **Atomics**



source: devblogs.nvidia.com/parallelforall/

Block-Jacobi preconditioning

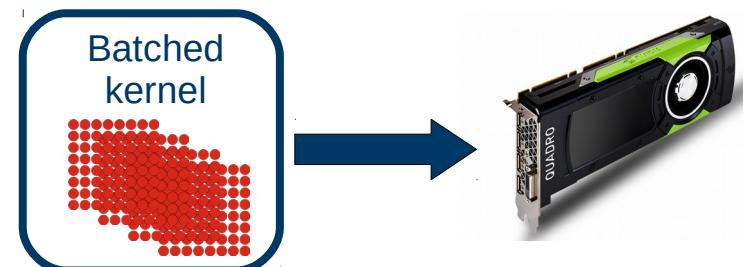
Solving 30-by-30 systems in sequence on
a GPU with several thousand cores
wastes computational resources!

N x GEMM vs batched GEMM comparison:

<https://devblogs.nvidia.com/parallelforall/cublas-strided-batched-matrix-multiply>

Batched routines

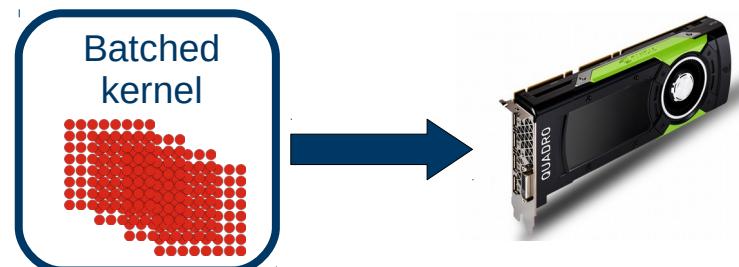
Launch a **single kernel** which applies an operation to **multiple** independent **data entities** in parallel.



* Measured on NVIDIA P100

Batched routines

Launch a **single kernel** which applies an operation to **multiple** independent **data entities** in parallel.



- There is no standard BLAS & LAPACK interface
- Most implementations only support problems of equal sizes
- High performance libraries are not optimized for small blocks
 - cuBLAS batched trsv: ~25 Gflop/s *
 - MAGMA-sparse SpMV: 60 – 90 Gflop/s *

* Measured on NVIDIA P100

MAGMA-sparse batched system solves

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

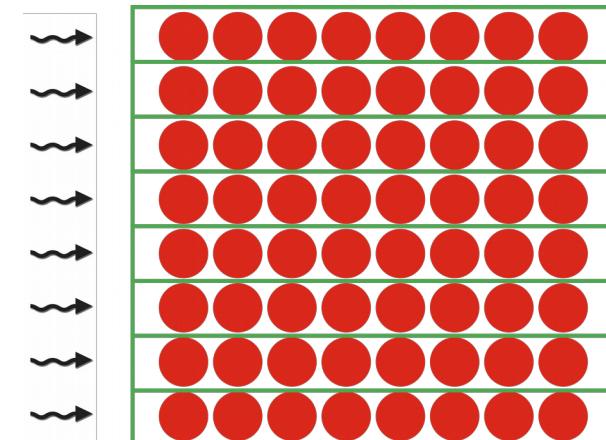


MAGMA-sparse batched system solves

- 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

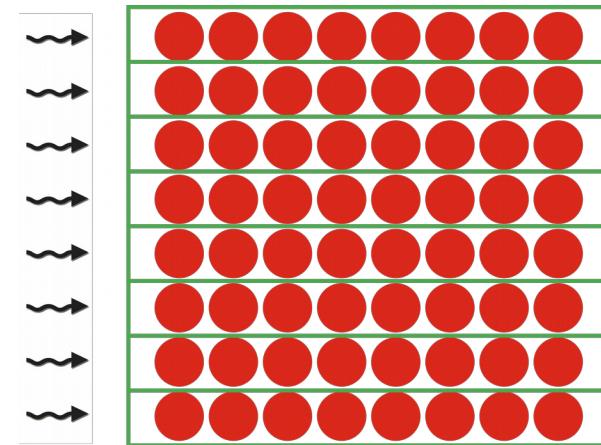


MAGMA-sparse batched system solves

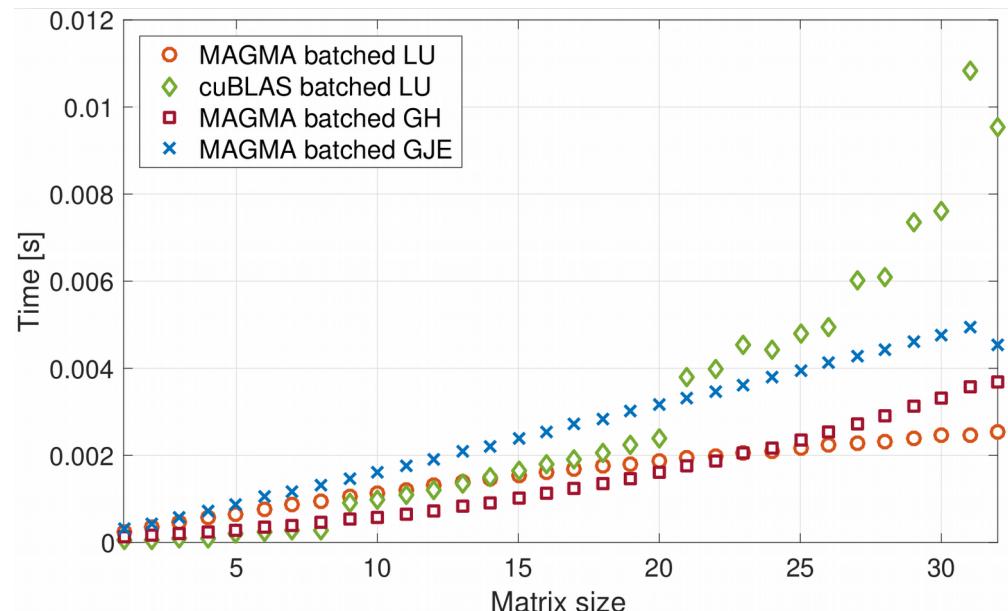
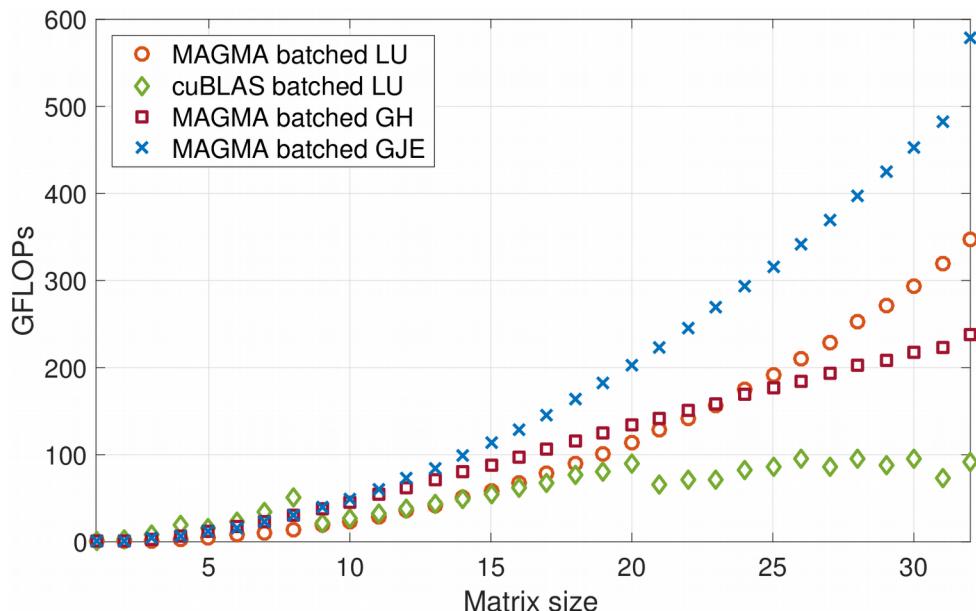
- Assign one warp to each problem
 - hardware SIMD unit, represented as a group of 32 threads in CUDA



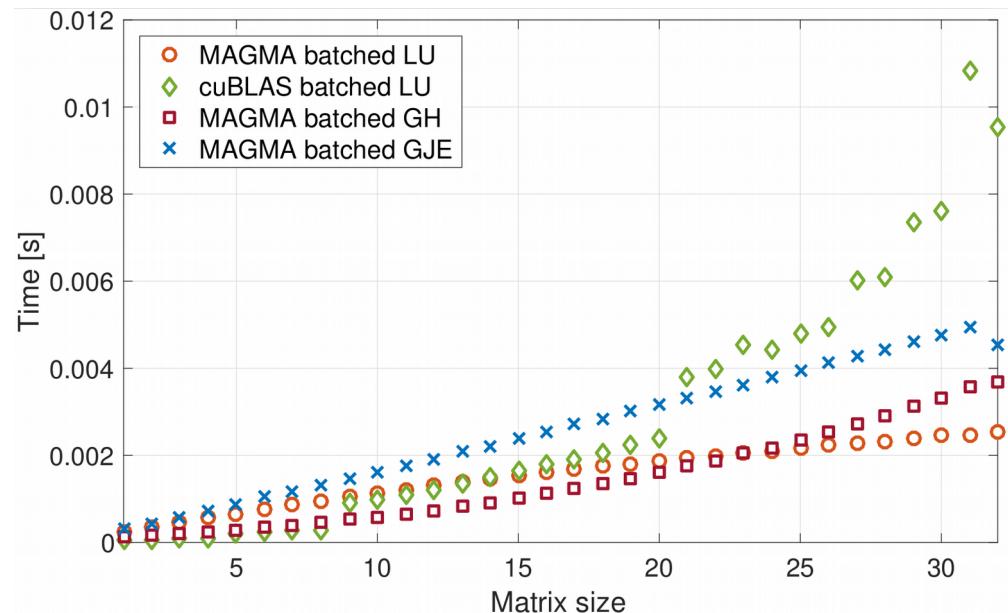
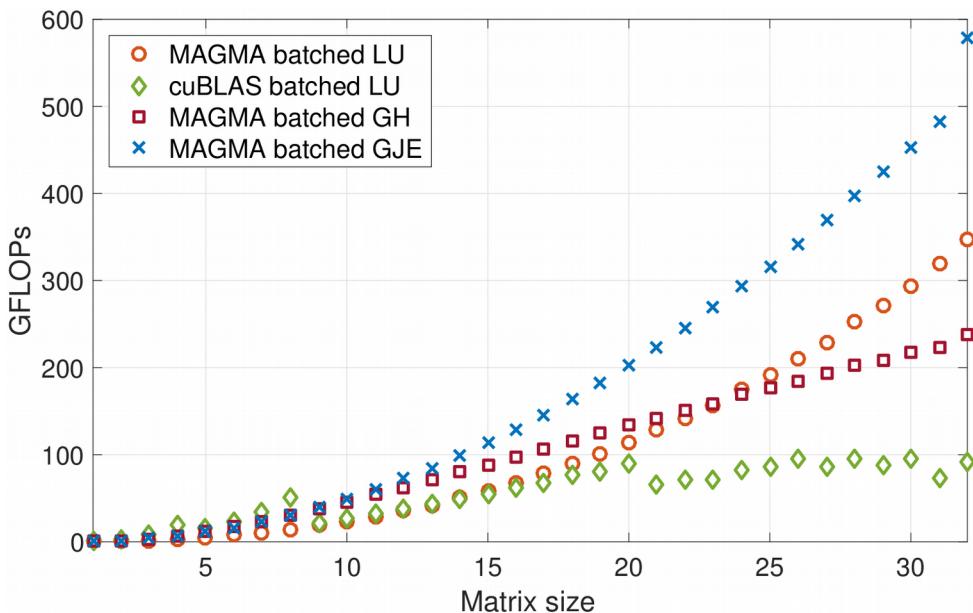
- Process each row (column) by a single thread
 - Able to support problems of size up to 32-by-32
 - keep the entire row (column) in thread's registers
 - Communicate data between rows via warp-shuffles
 - Current implementation: **use padding for problems of smaller sizes**
 - Future work: **multiple smaller problems per warp**
- Use implicit pivoting
 - Do not explicitly swap rows (column), “re-assign” the threads instead



Batched routines performance



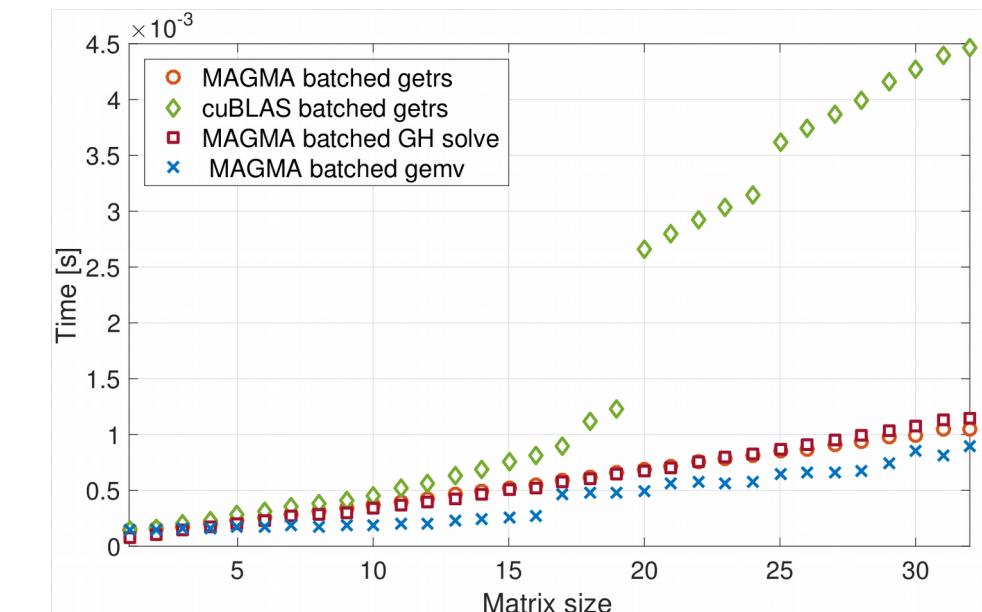
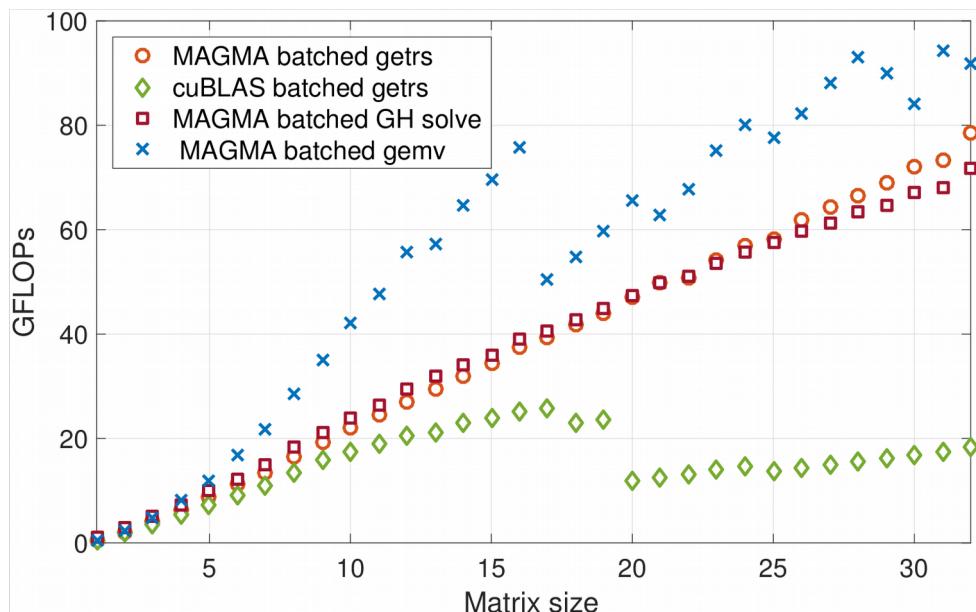
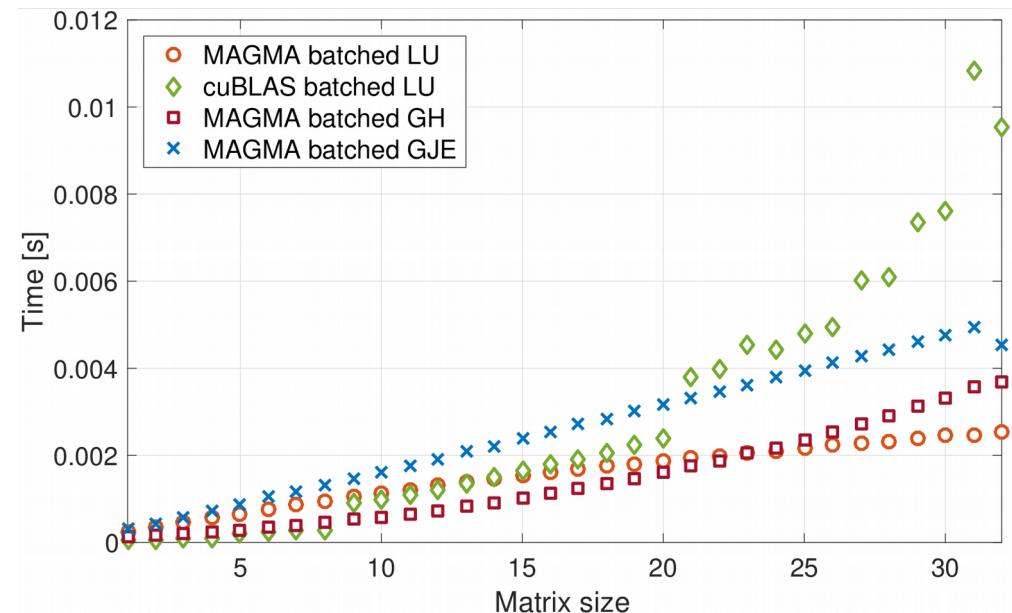
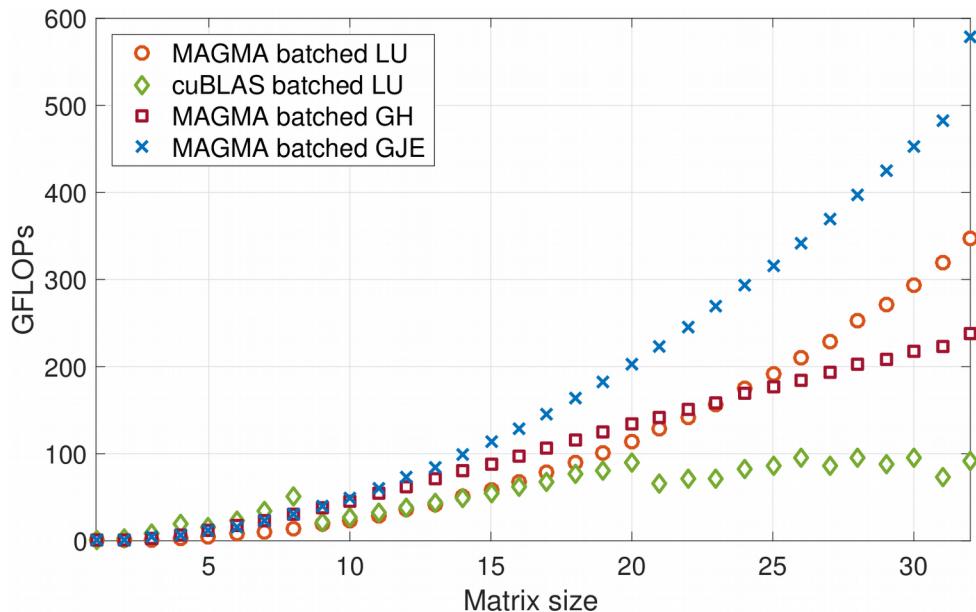
Batched routines performance



MAGMA-sparse routines can also:

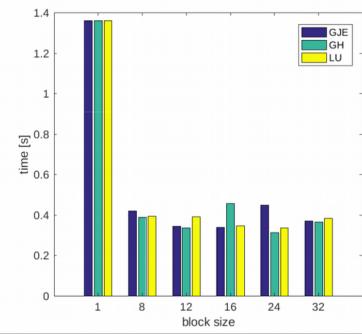
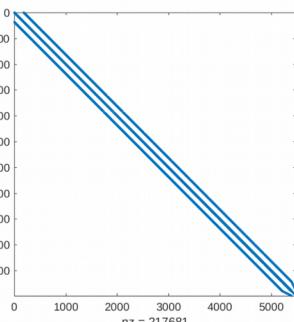
- Handle problems of different sizes
- Integrate diagonal block extraction and diagonal block decomposition / inversion into a single kernel

Batched routines performance

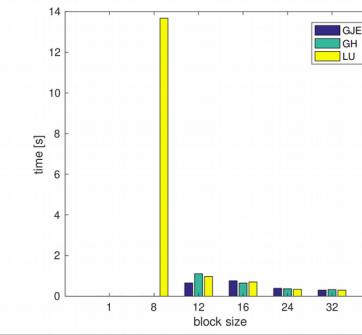
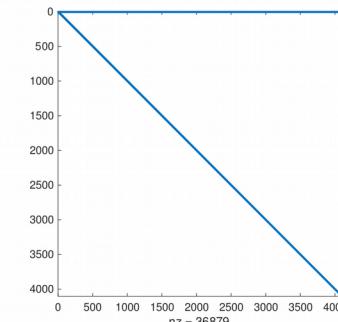


Complete solver runtime

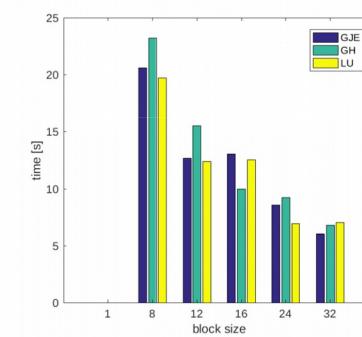
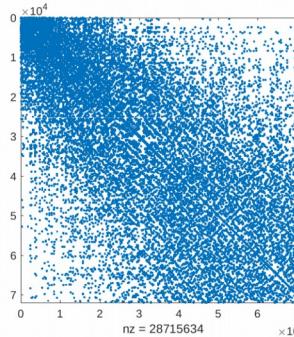
s2rmt3m1



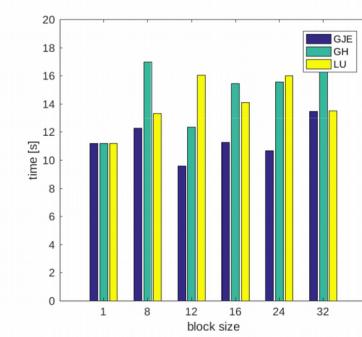
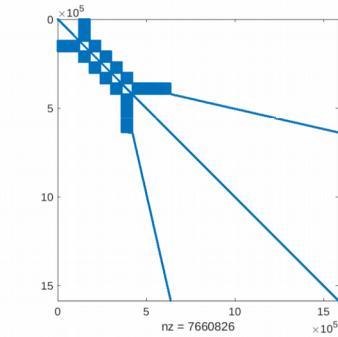
Chebyshev3



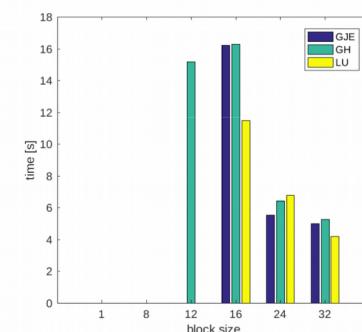
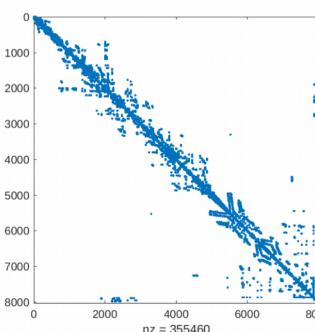
nd24k



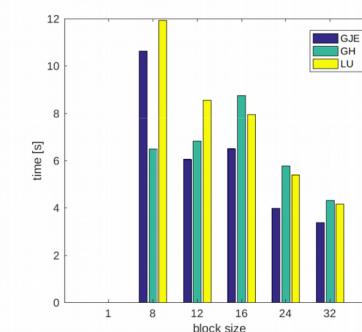
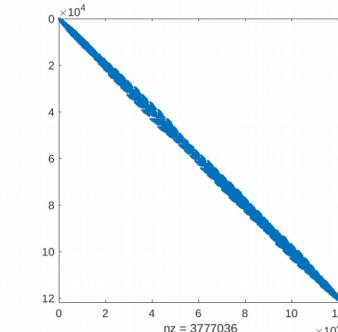
G3_circuit



bcsstk38

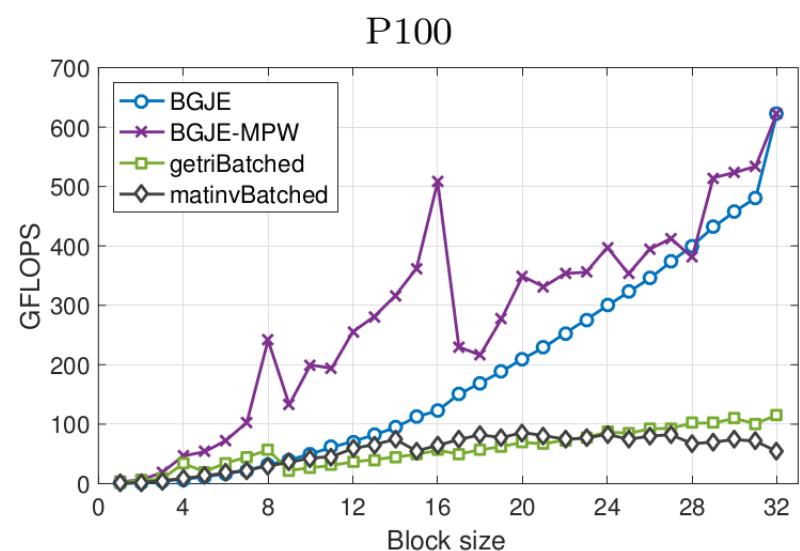


ship_003



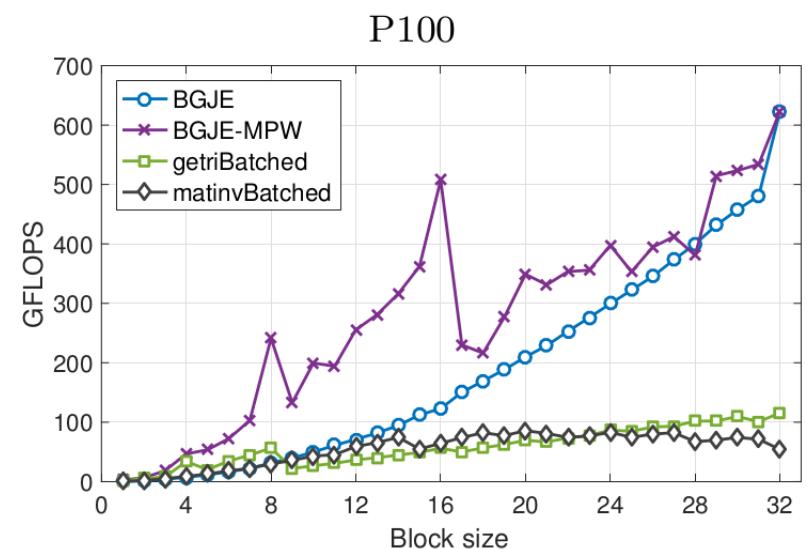
Flexible-size batched routines & future research

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

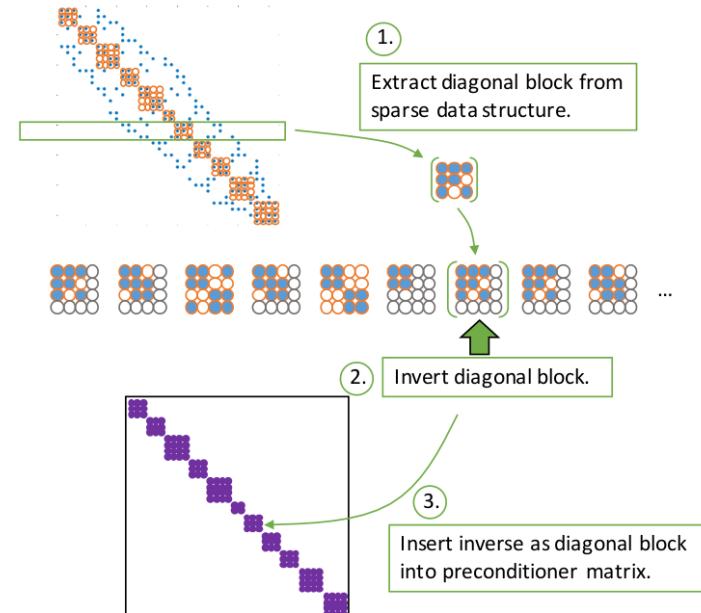


Flexible-size batched routines & future research

- Problems can be **too 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

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

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 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:

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

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:

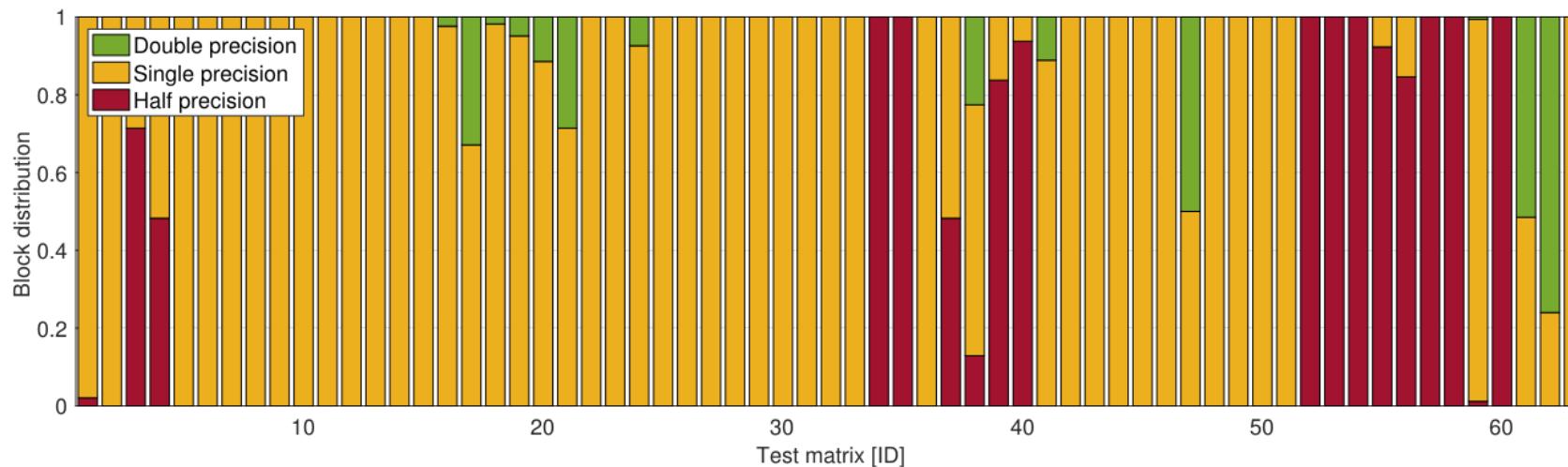
$$\begin{cases} \text{fp16} & \text{if } 0 < \kappa_1(D_i) \leq 10^2, \\ \text{fp32} & \text{if } 10^2 < \kappa_1(D_i) \leq 10^6, \text{ and} \\ \text{fp64} & \text{otherwise,} \end{cases}$$

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 *:

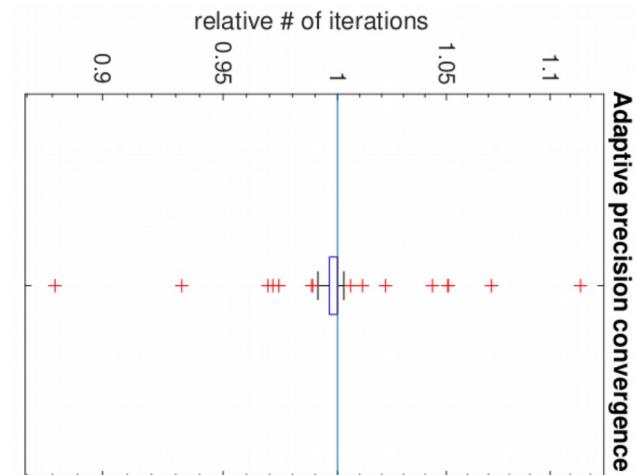


* Prototype implementation in MATLAB,
Results on 63 matrices from SuiteSparse

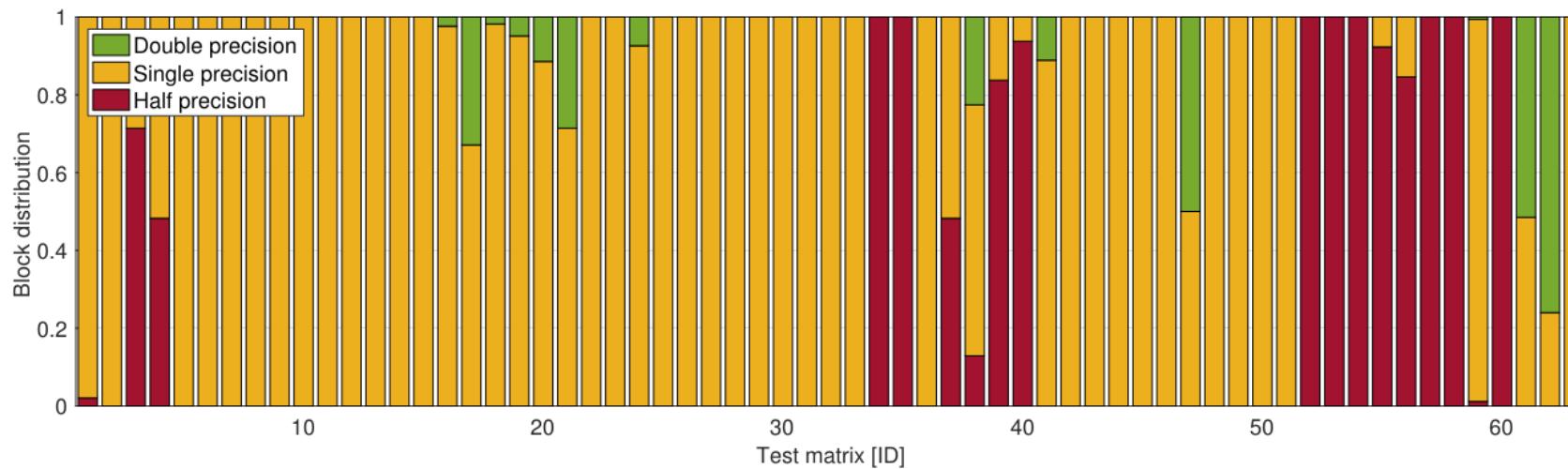
Experimental results

Determining the precision:

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



% of diagonal blocks stored in each precision *:



Usually < 5% iteration increase

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

Energy efficiency of adaptive precision block-Jacobi

Memory access is an order of magnitude more expensive than computation! [Shalf 2013]

Energy efficiency of adaptive precision block-Jacobi

Memory access is an order of magnitude more expensive than computation! [Shalf 2013]

Energy model assumptions:

- Accessing 1 bit of data has a cost of 1 (energy unit)
- Disregard energy cost of arithmetic operations
- Total energy cost of each iteration:

$$\underbrace{14n \cdot \text{fp64}}_{\text{vector memory transfers}} + \underbrace{(2n + n_z) \cdot \text{fp64} + (n + n_z) \cdot \text{int32}}_{\text{CSR-SpMV memory transfers}} + \underbrace{2n \cdot \text{fp64} + \sum_{i=1}^N m_i^2 \cdot \text{fpxx}_i}_{\text{preconditioner memory transfers}}$$

Energy efficiency of adaptive precision block-Jacobi

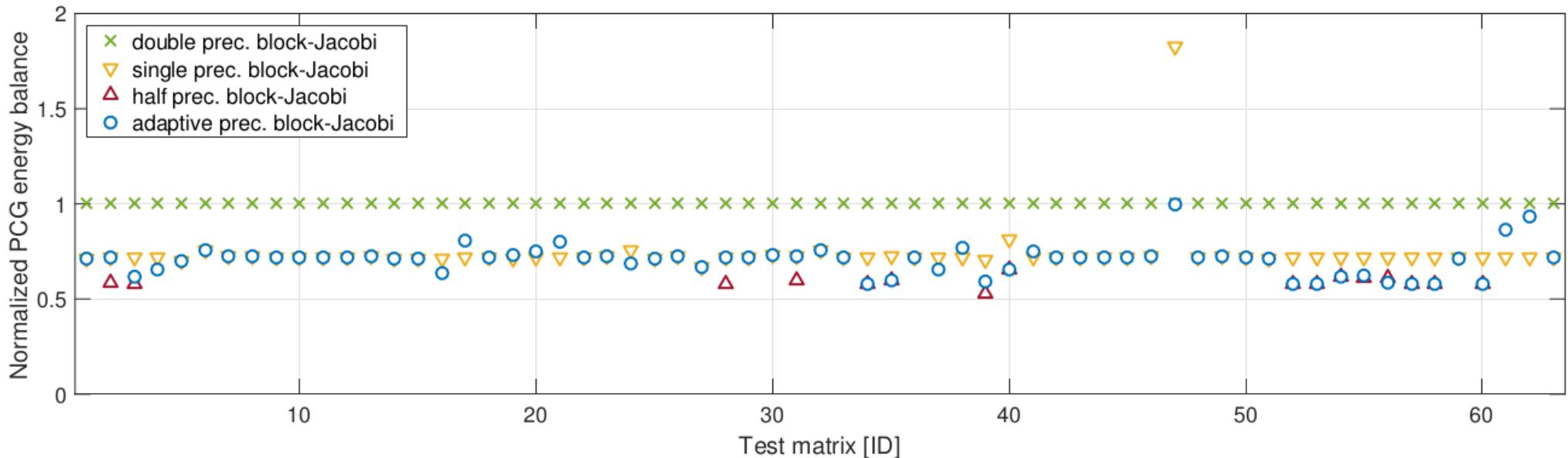
Memory access is an order of magnitude more expensive than computation! [Shalf 2013]

Energy model assumptions:

- Accessing 1 bit of data has a cost of 1 (energy unit)
- Disregard energy cost of arithmetic operations
- Total energy cost of each iteration:

$$\underbrace{14n \cdot \text{fp64}}_{\text{vector memory transfers}} + \underbrace{(2n + n_z) \cdot \text{fp64} + (n + n_z) \cdot \text{int32}}_{\text{CSR-SpMV memory transfers}} + \underbrace{2n \cdot \text{fp64} + \sum_{i=1}^N m_i^2 \cdot \text{fpxx}_i}_{\text{preconditioner memory transfers}}$$

Predicted energy savings of adaptive precision block-Jacobi:



Thank you! Questions?

All functionalities are part of the MAGMA-sparse project.

MAGMA SPARSE

ROUTINES BiCG, BiCGSTAB, Block-Asynchronous Jacobi, CG, CGS, GMRES, IDR, Iterative refinement, LOBPCG, LSQR, QMR, TFQMR

PRECONDITIONERS ILU / IC, Jacobi, ParILU, ParILUT, Block Jacobi, ISAI

KERNELS SpMV, SpMM

DATA FORMATS CSR, ELL, SELL-P, CSR5, HYB

<http://icl.cs.utk.edu/magma/>

Scan me
for slides!



github.com/gflegar/talks/tree/master/pmf_zagreb_2017_12

This research is based on a cooperation between Hartwig Anzt (Karlsruhe Institute of Technology), Jack Dongarra (University of Tennessee), Nicholas Higham (University of Manchester), Goran Flegar and Enrique S. Quintana-Ortí (Universidad Jaume I).

