

Flexible-Size Batched LU for Small Matrices and its Integration into Block-Jacobi Preconditioning

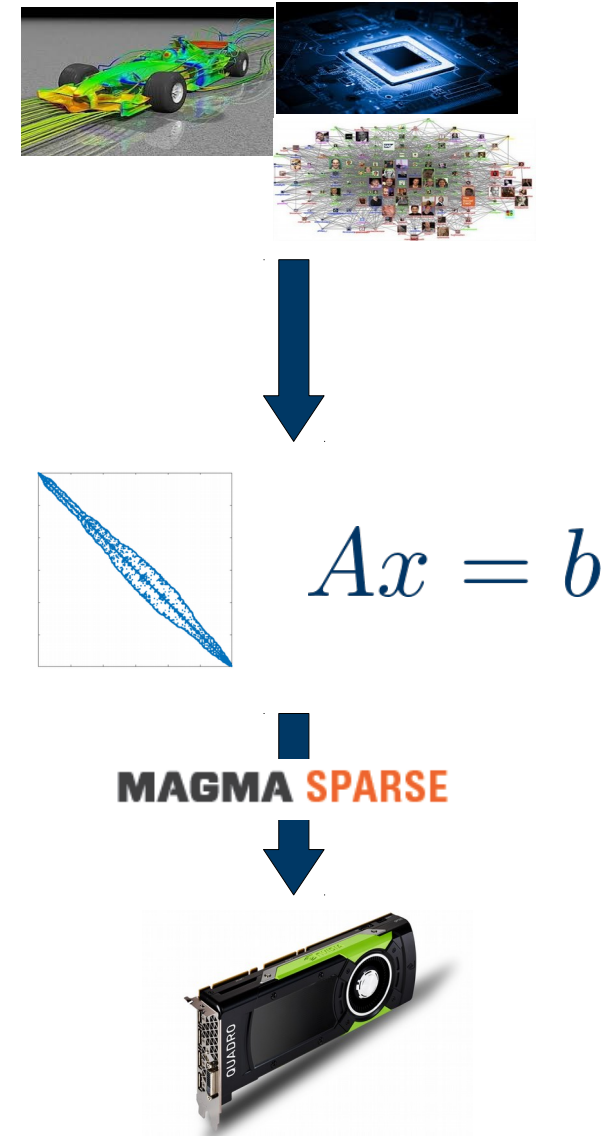
Hartwig Anzt, Jack Dongarra, [Goran Flegar](#), Enrique S. Quintana-Ortí



Scan me
for slides!

MAGMA-sparse software library

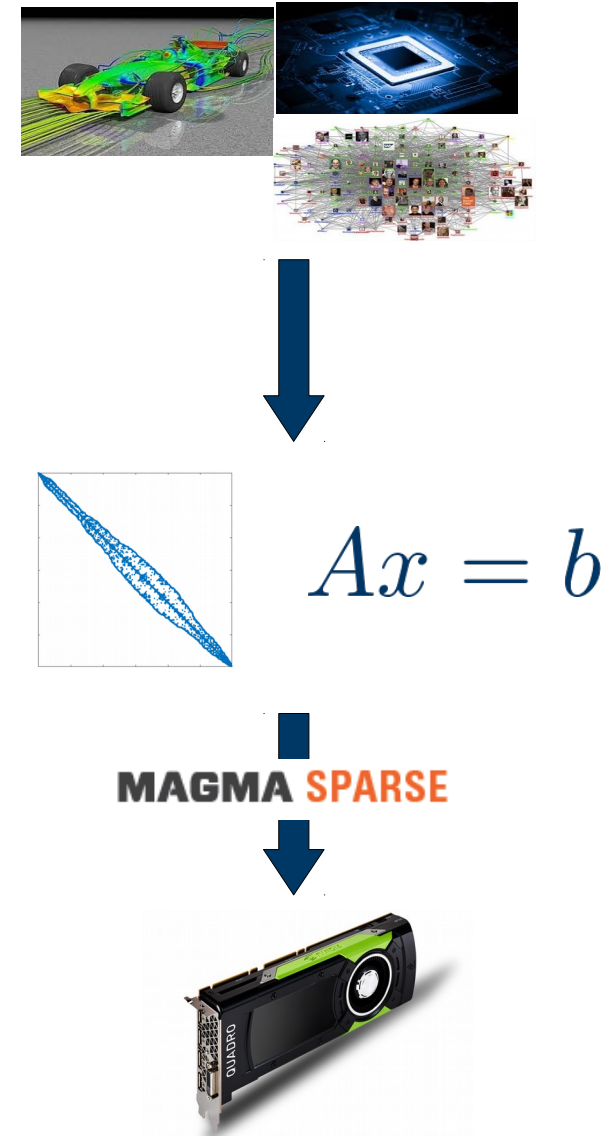
- GPU-accelerated sparse linear algebra library
 - Focus: linear systems



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

MAGMA-sparse software library

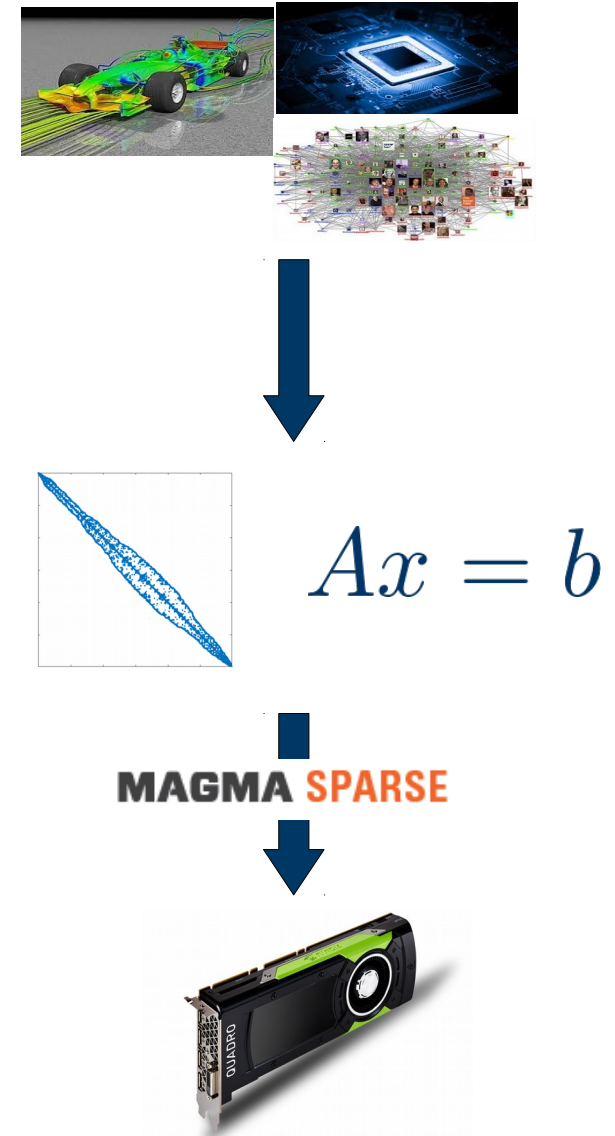
- GPU-accelerated sparse linear algebra library
 - Focus: linear systems
 - Iterative, Krylov-subspace based linear solvers
 - SpMV
 - BLAS-1 operations



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

MAGMA-sparse software library

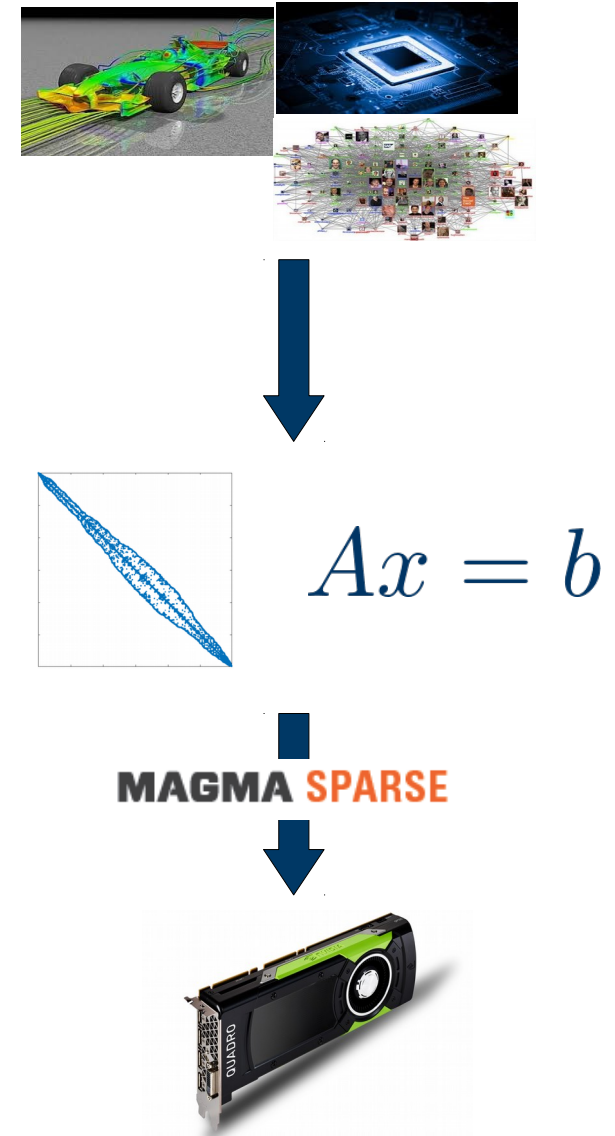
- GPU-accelerated sparse linear algebra library
 - Focus: linear systems
 - Iterative, Krylov-subspace based linear solvers
 - SpMV
 - BLAS-1 operations
 - Sparse matrix formats & SpMV
 - accelerate each iteration of the solver



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

MAGMA-sparse software library

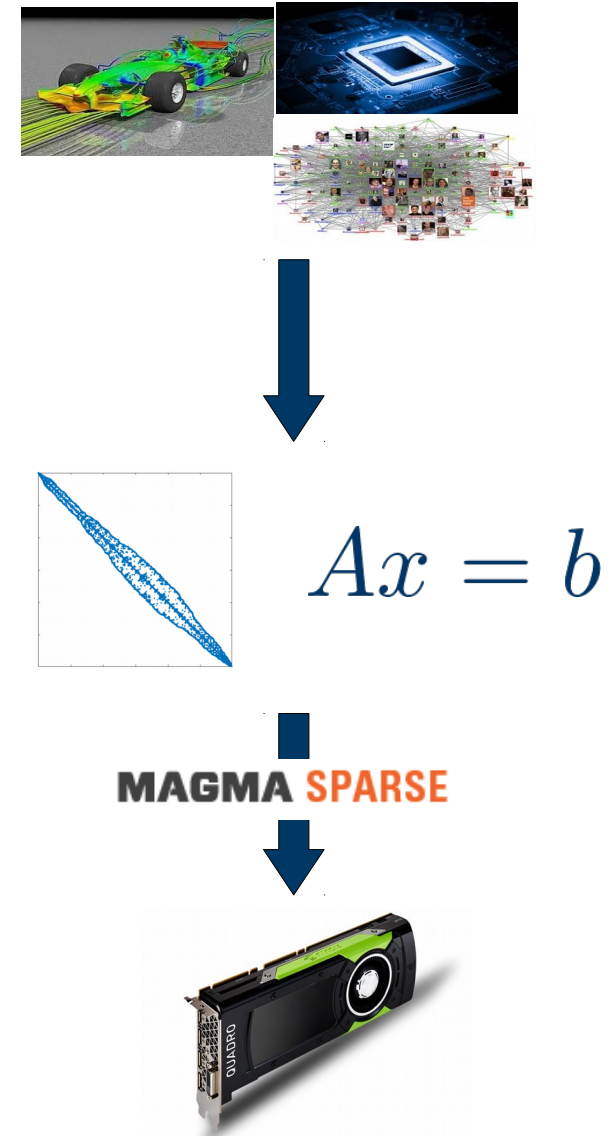
- GPU-accelerated sparse linear algebra library
 - Focus: linear systems
 - Iterative, Krylov-subspace based linear solvers
 - SpMV
 - BLAS-1 operations
 - Sparse matrix formats & SpMV
 - accelerate each iteration of the solver
 - Preconditioners
 - reduce the number of iterations



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

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

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

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

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

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

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

**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!

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



$$z := Ax$$

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

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



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

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

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

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

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



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

Preconditioner application

Trade-off:

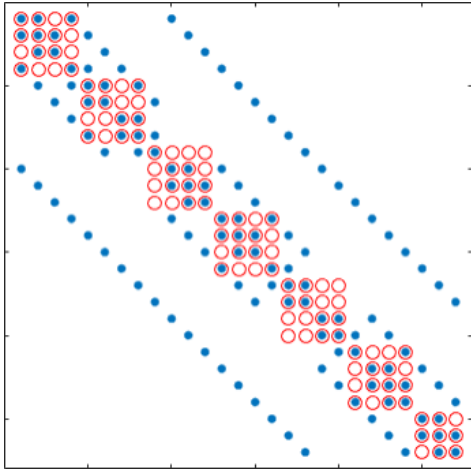
faster convergence,
but more work per iteration

Block-Jacobi Preconditioning

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

Block-Jacobi Preconditioning

- Current focus: improve performance for problems with inherent block structure
 - Usually up to 30 unknowns per block (**blocks can be of different sizes!**)



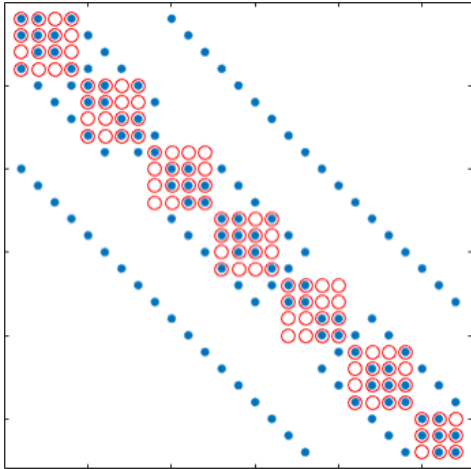
- Block-Jacobi preconditioning
 - Use only diagonal blocks for approximation

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

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

Block-Jacobi Preconditioning

- Current focus: improve performance for problems with inherent block structure
 - Usually up to 30 unknowns per block (**blocks can be of different sizes!**)



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

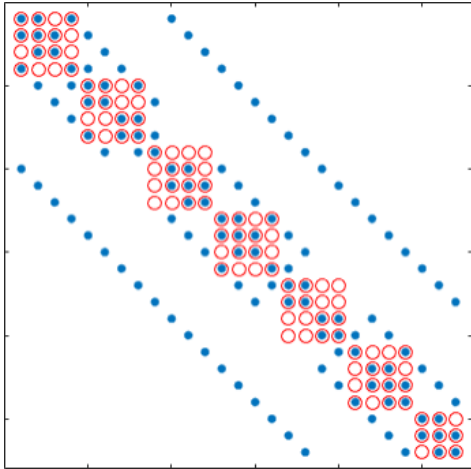
$$\text{diag}(A) = [D_1, \dots, D_k]$$

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

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

Block-Jacobi Preconditioning

- Current focus: improve performance for problems with inherent block structure
 - Usually up to 30 unknowns per block (**blocks can be of different sizes!**)



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

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

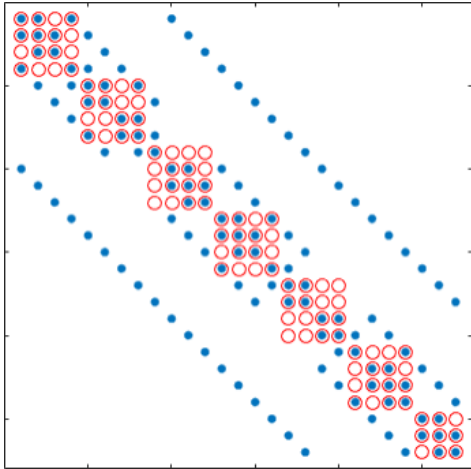
$$M := \text{diag}(D_1, \dots, D_k)$$

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

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

Block-Jacobi Preconditioning

- Current focus: improve performance for problems with inherent block structure
 - Usually up to 30 unknowns per block (**blocks can be of different sizes!**)



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

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

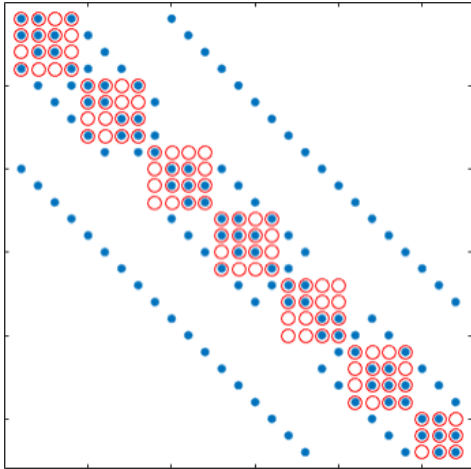
$$M := \text{diag}(D_1, \dots, D_k)$$

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

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

Block-Jacobi Preconditioning

- Current focus: improve performance for problems with inherent block structure
 - Usually up to 30 unknowns per block (**blocks can be of different sizes!**)



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

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

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

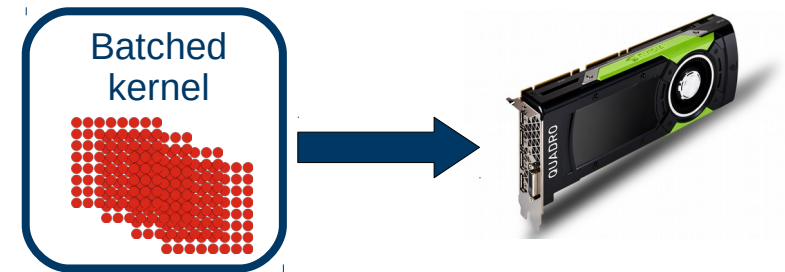
$$y := M^{-1}z \longrightarrow y_i := D_i^{-1}z_i, \quad \forall i$$

$$\longrightarrow D_i y_i = z_i \longrightarrow \begin{array}{l} \boxed{D_i = L_i U_i} \text{ Setup} \\ \boxed{\begin{array}{l} U_i y_i = w_i \\ L_i w_i = z_i \end{array}} \text{ Application} \end{array}$$

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

Batched routines

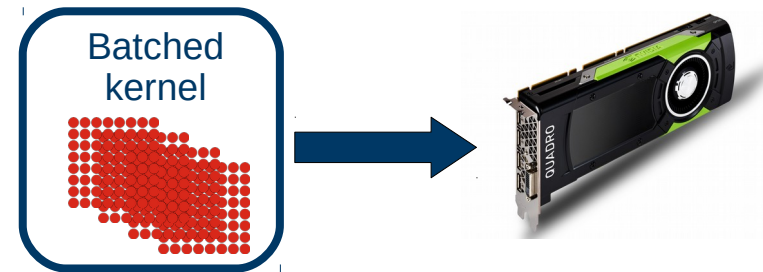
Launch a **single kernel** which applies an operation to **multiple independent data entities** in parallel.



** 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*

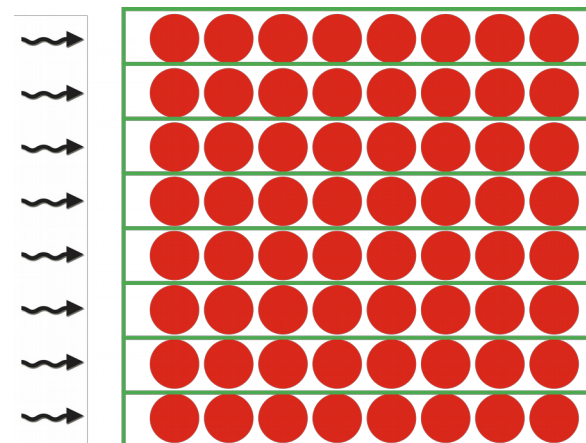
MAGMA-sparse LU & trsv outline

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



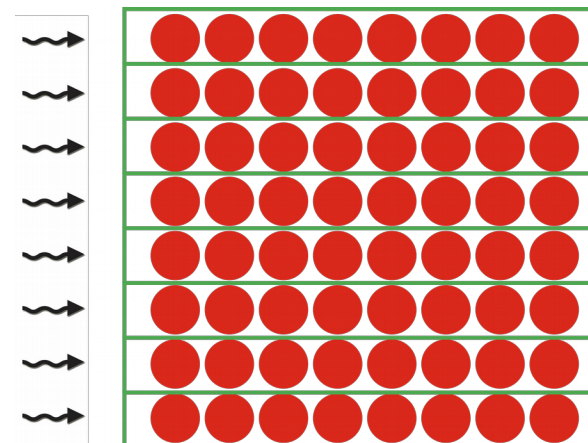
MAGMA-sparse LU & trsv outline

- Assign **one warp to each problem**
 - hardware SIMD unit, represented as a group of 32 threads in CUDA
- Process each row by a single thread
 - Able to support problems of size up to 32-by-32
 - keep the entire row 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 LU & trsv outline

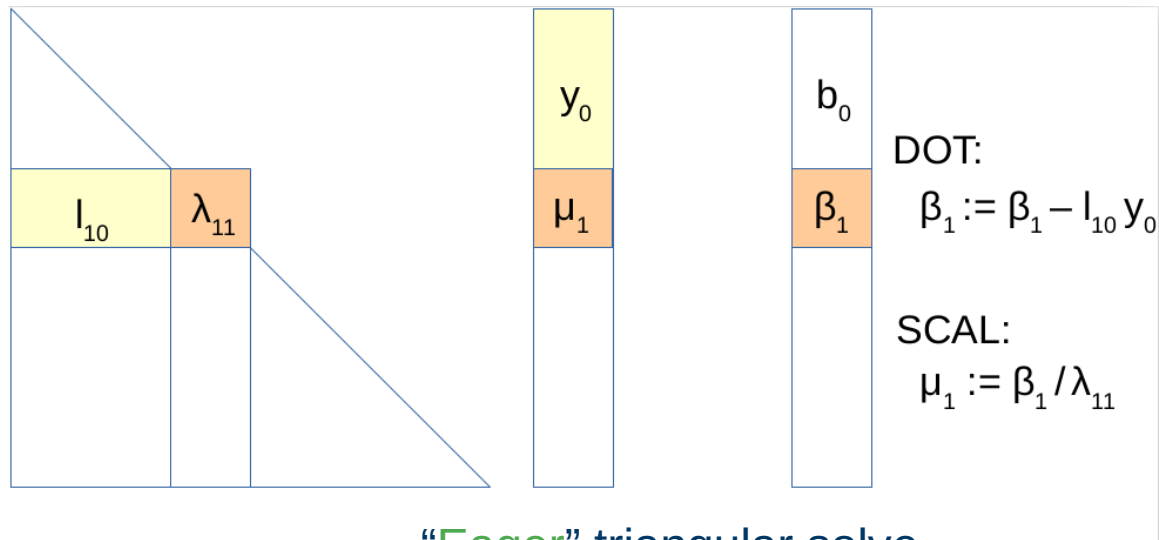
- Assign **one warp to each problem**
 - hardware SIMD unit, represented as a group of 32 threads in CUDA
- Process each row by a single thread
 - Able to support problems of size up to 32-by-32
 - keep the entire row in thread's **registers**
 - Communicate data between rows via **warp-shuffles**
 -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



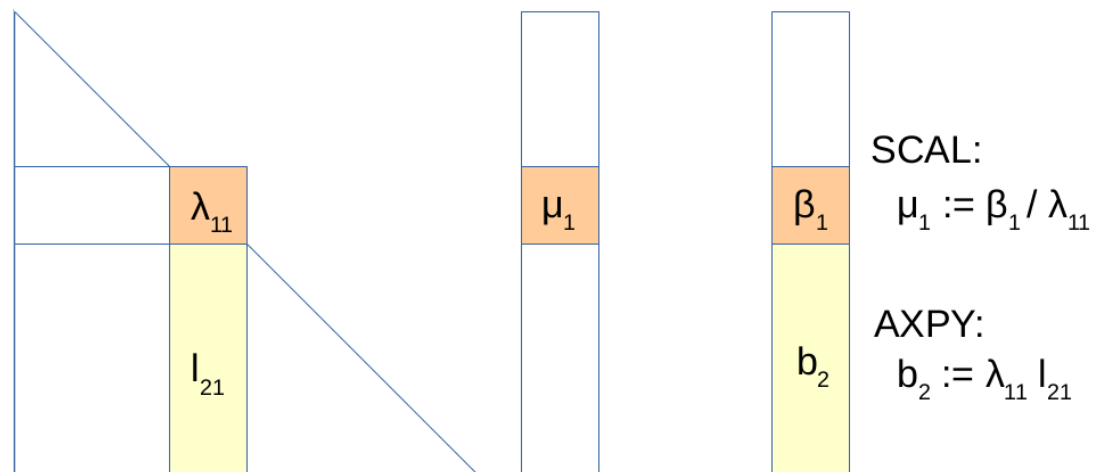
MAGMA-sparse LU & trsv outline

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

“Lazy” triangular solve



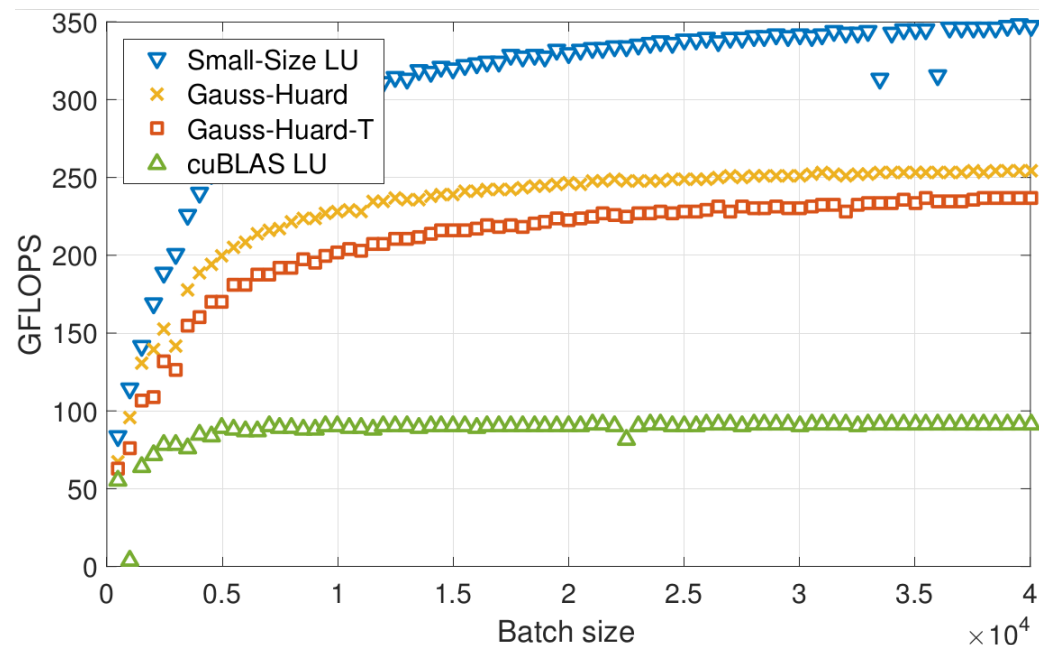
“Eager” triangular solve



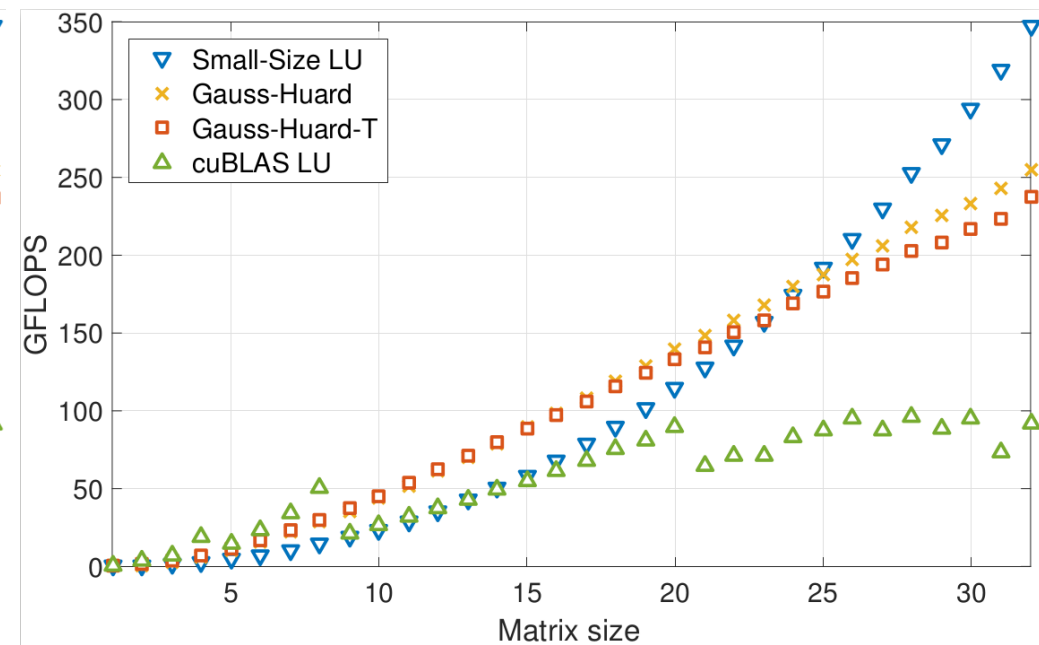
LU decomposition performance

- Comparisson of MAGMA-sparse vs cuBLAS batched LU decomposition
- Gauss-Huard(-T) is a similar approach, using a different algorithm for decomposition/solves *

Fixed problem size (32)



Fixed batch size (40,000)

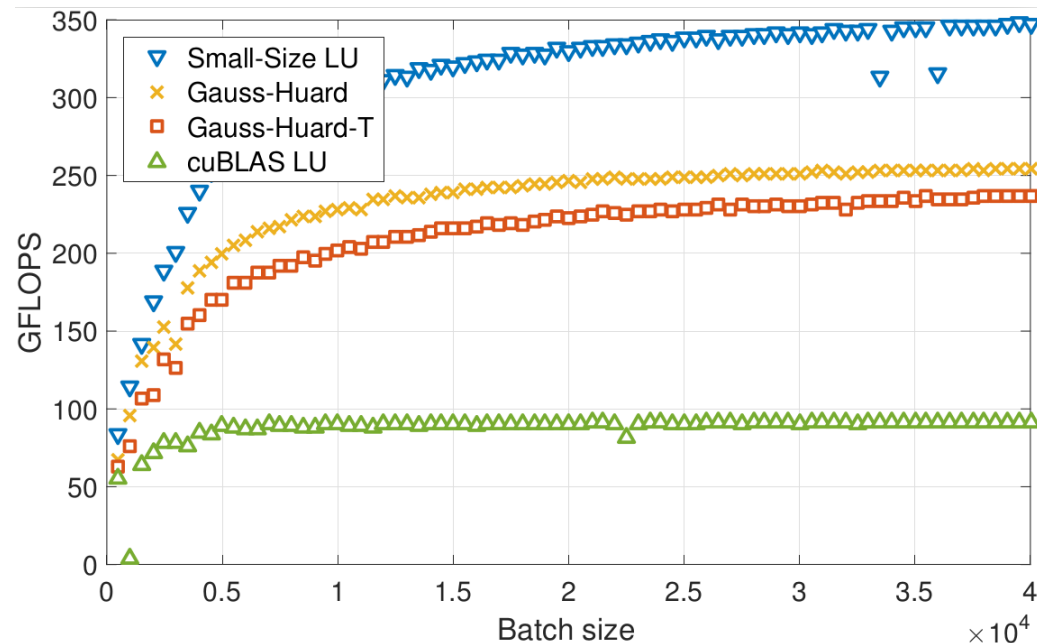


* Anzt et al., Variable-Size Batched Gauss-Huard for Block-Jacobi Preconditioning, ICCS'17

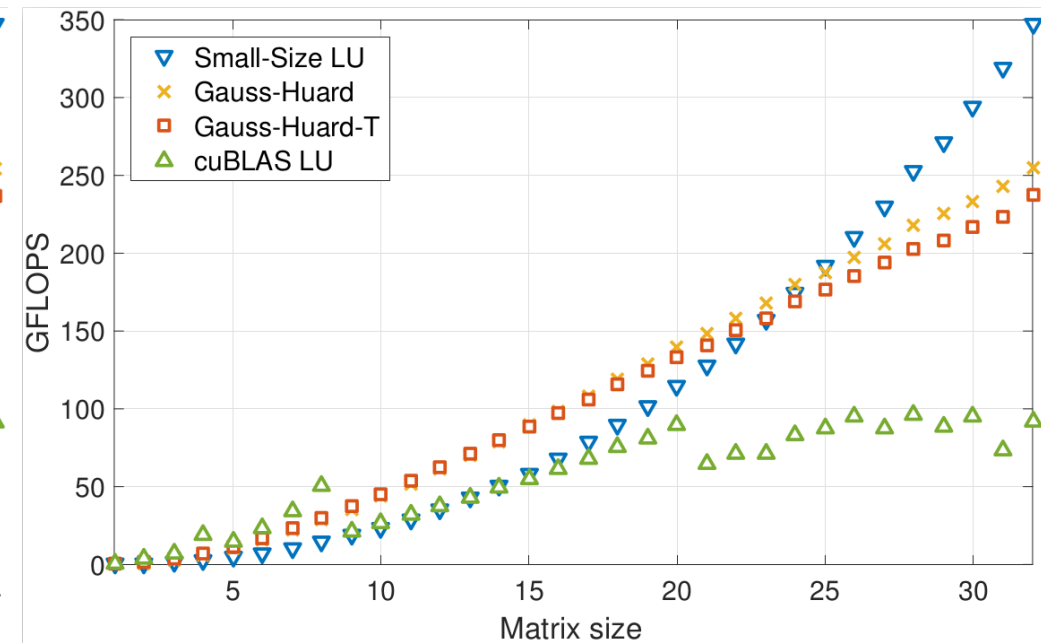
LU decomposition performance

- Comparison of MAGMA-sparse vs cuBLAS batched LU decomposition
- Gauss-Huard(-T) is a similar approach, using a different algorithm for decomposition/solves*

Fixed problem size (32)



Fixed batch size (40,000)



MAGMA-sparse LU can also:

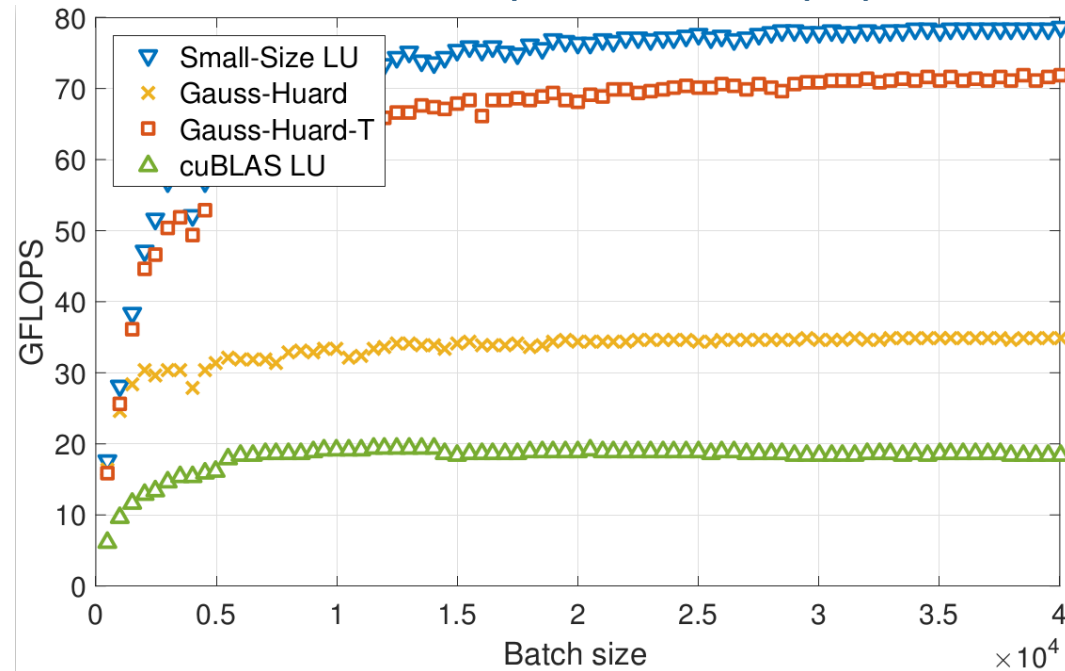
- Handle problems of different sizes
- Integrate diagonal block extraction and diagonal block decomposition into a single kernel

* Anzt et al., Variable-Size Batched Gauss-Huard for Block-Jacobi Preconditioning, ICCS'17

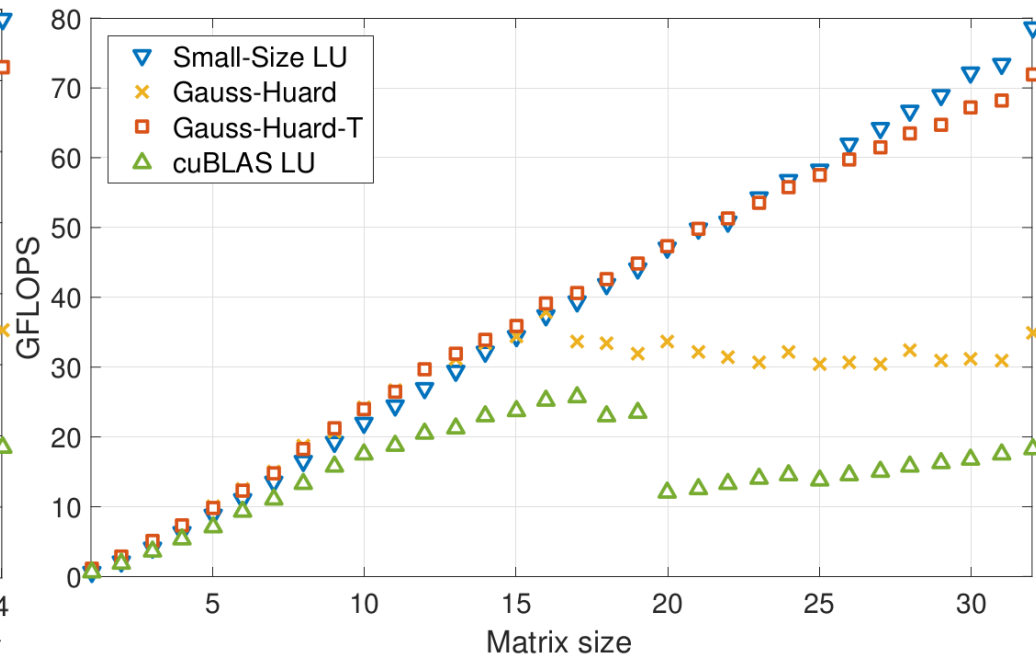
Triangular solve performance

- Comparisson of MAGMA-sparse vs cuBLAS batched triangular solves
- Gauss-Huard(-T) is a similar approach, using a different algorithm for decomposition/solves*

Fixed problem size (32)

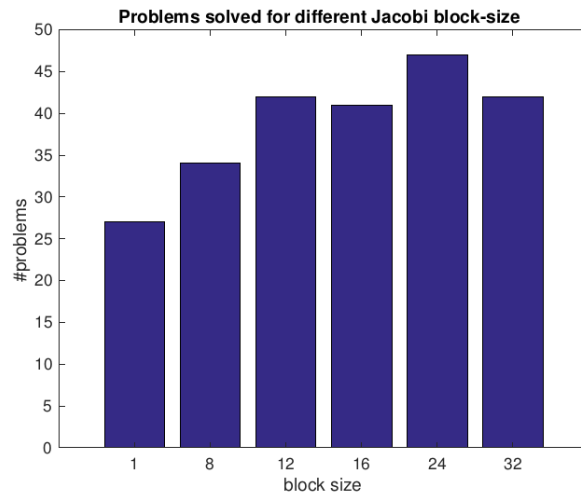


Fixed batch size (40,000)



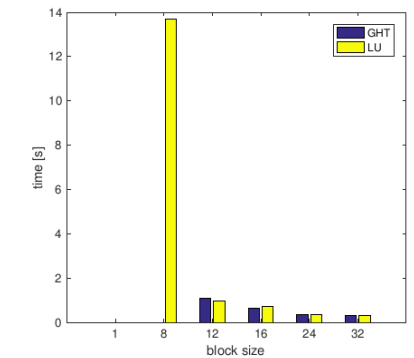
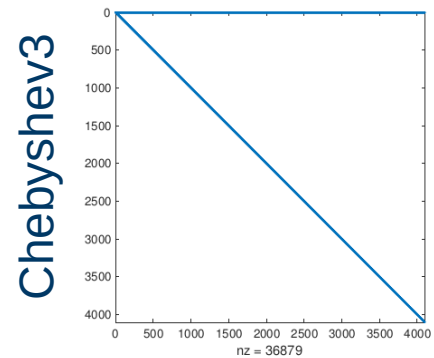
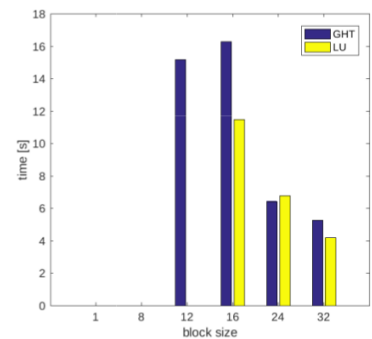
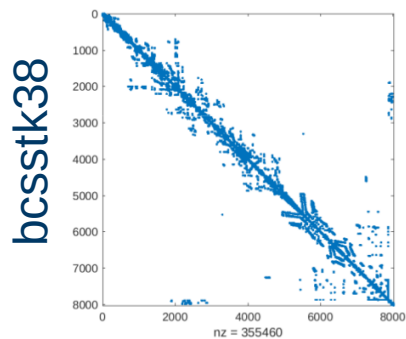
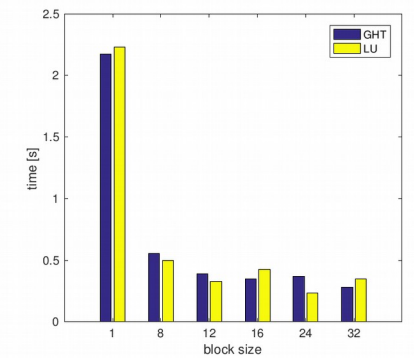
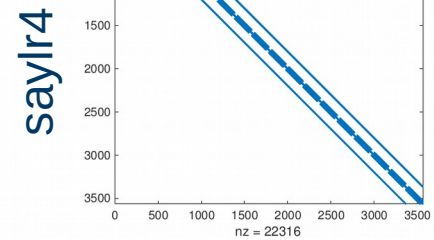
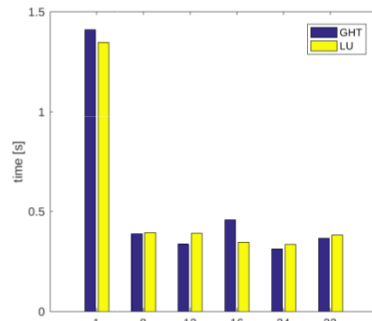
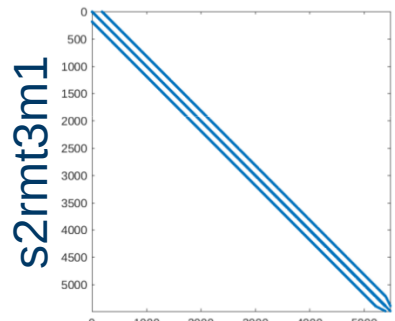
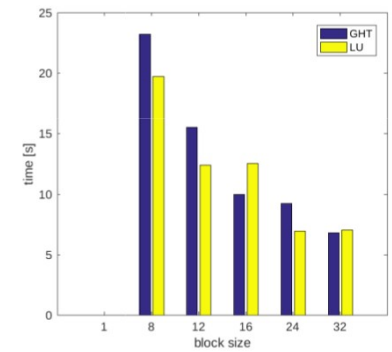
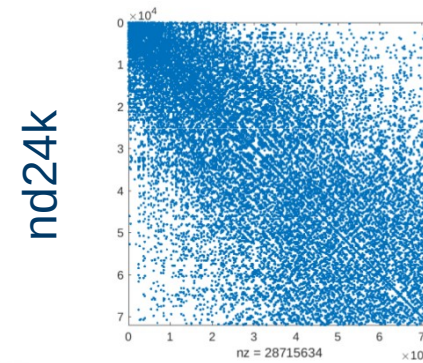
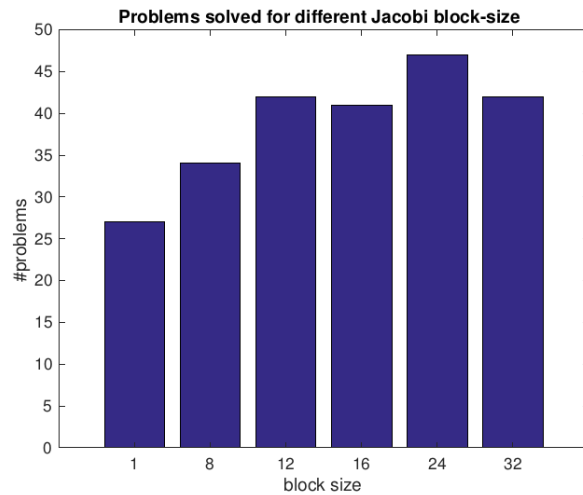
Complete solver runtime

56 problems from SuitSparse



Complete solver runtime

56 problems from SuitSparse



Conclusion

“Solving many **small** problems in sequence on manycore hardware wastes computational resources! We can design **batched routines** which apply the same operation on a set of independent data entities.”

Workshop on batched BLAS*

- Effort to **standardize** the batched BLAS interface

* <http://bit.ly/Batch-BLAS-2017>

Conclusion

“Solving many **small** problems in sequence on manycore hardware wastes computational resources! We can design **batched routines** which apply the same operation on a set of independent data entities.”

Workshop on batched BLAS*

- Effort to **standardize** the batched BLAS interface

What is **small**?

- Can we design a single routine that can handle both 8-by-8 and 500-by-500 matrices?
- Or should we have more routines?

* <http://bit.ly/Batch-BLAS-2017>

Conclusion

“Solving many **small** problems in sequence on manycore hardware wastes computational resources! We can design **batched routines** which apply the same operation on a set of independent data entities.”

Workshop on batched BLAS*

- Effort to **standardize** the batched BLAS interface

What is **small**?

- Can we design a single routine that can handle both 8-by-8 and 500-by-500 matrices?
- Or should we have more routines?

Does batched BLAS / LAPACK solve the problem?

- Even with efficient implementation, **cannot “merge”** with diagonal block extraction.
- Batched gemm proposal has **16 input arguments!**
- **Global synchronization** between two batched calls.

* <http://bit.ly/Batch-BLAS-2017>

Conclusion

“Solving many **small** problems in sequence on manycore hardware wastes computational resources! We can design **batched routines** which apply the same operation on a set of independent data entities.”

Workshop on batched BLAS*

- Effort to **standardize** the batched BLAS interface

What is **small**?

- Can we design a single routine that can handle both 8-by-8 and 500-by-500 matrices?
- Or should we have more routines?

Does batched BLAS / LAPACK solve the problem?

- Even with efficient implementation, **cannot “merge”** with diagonal block extraction.
- Batched gemm proposal has **16 input arguments!**
- **Global synchronization** between two batched calls.

Instead provide BLAS which operates on a part of memory/core hierarchy?

- E.g. block, warp, thread level BLAS for CUDA.
- Let users build their own batched routines from these building blocks.

* <http://bit.ly/Batch-BLAS-2017>

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/icpp_2017

This research is based on a cooperation between Hartwig Anzt, Jack Dongarra (University of Tennessee), Goran Flegar and Enrique S. Quintana-Ortí (Universitat Jaume I).



THE UNIVERSITY OF
TENNESSEE
KNOXVILLE



UNIVERSITAT
JAUME I