MPI Magdeburg, 20 June 2017

UNIVERSITAT JAUME I

Flexible-Size Batched Inversion and Factorization Routines for Block-Jacobi Preconditioning on GPUs

Goran Flegar

Joint work with Hartwig Anzt and Enrique S. Quintana-Ortí.

Scan me for slides!

github.com/gflegar/talks/mpi_magdeburg_2017_06

flegar@uji.es

MPI Magdeburg, 20 June 2017

UNIVERSITAT JAUME I

Overview:

Flexible-Size Batched Inversion and Factorization Routines for Block-Jacobi Preconditioning on GPUs

Goran Flegar

Joint work with Hartwig Anzt and Enrique S. Quintana-Ortí.

Scan me for slides!

github.com/gflegar/talks/mpi_magdeburg_2017_06

flegar@uji.es

Problem setting

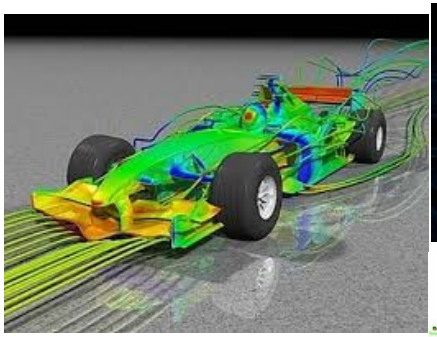
$$Ax = b,\ A \in \mathbb{R}^{n \times n}$$

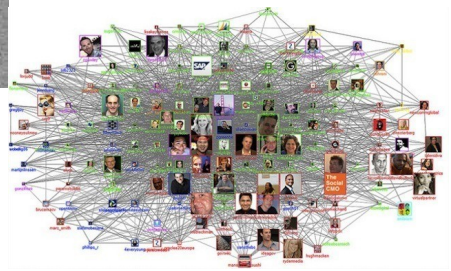
- Sparse linear system
  - The majority of coefficients is 0

Problem setting

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

- Sparse linear system
  - The majority of coefficients is 0
  - Fluid dynamics, circuit simulation, graph analytics

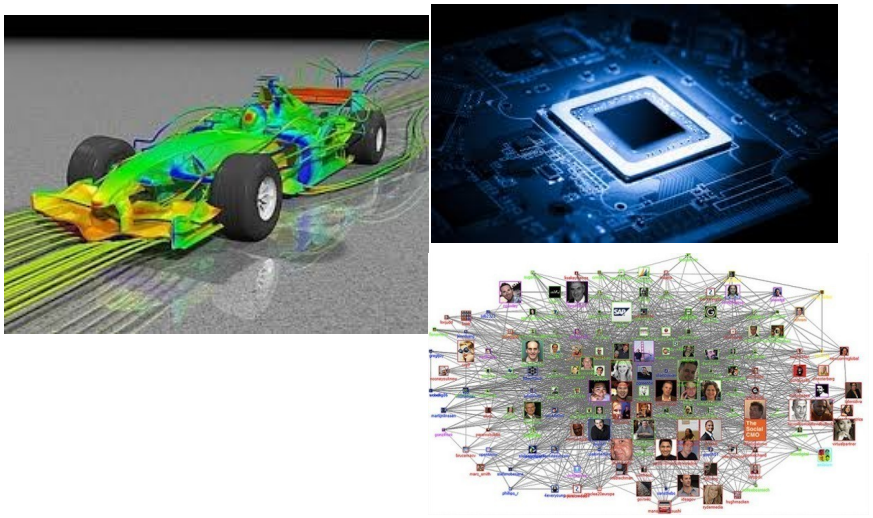




Problem setting

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

- Sparse linear system
  - The majority of coefficients is 0
  - Fluid dynamics, circuit simulation, graph analytics



- Solve it using an iterative Krylov method
  - Vector operations + matrix-vector product
  - Convergence related to spectral properties of coef. matrix

$i \Leftarrow 0$
$r \Leftarrow b - Ax$
$d \Leftarrow r$
$\delta_{new} \Leftarrow r^T r$
$\delta_0 \Leftarrow \delta_{new}$
While $i < i_{max}$ and $\delta_{new} > \varepsilon^2 \delta_0$ do
  $q \Leftarrow Ad$
  $\alpha \Leftarrow \frac{\delta_{new}}{d^T q}$
  $x \Leftarrow x + \alpha d$
  If i is divisible by 50
    $r \Leftarrow b - Ax$
  else
    $r \Leftarrow r - \alpha q$
  $\delta_{old} \Leftarrow \delta_{new}$
  $\delta_{new} \Leftarrow r^T r$
  $\beta \Leftarrow \frac{\delta_{new}}{\delta_{old}}$
  $d \Leftarrow r + \beta d$
  $i \Leftarrow i + 1$

Preconditioning

- Improve convergence by solving a preconditioned system

$$M^{-1}Ax = M^{-1}b$$

Preconditioning

- Improve convergence by solving a preconditioned system
  - Explicitly computing the matrix product causes fill-in

$$M^{-1}Ax = M^{-1}b$$

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

Preconditioning

- Improve convergence by solving a preconditioned system
  - Explicitly computing the matrix product causes fill-in
  - Avoid it by decomposing the application of the product into two steps:
    - Sparse matrix-vector product
    - Preconditioner application

$$M^{-1}Ax = M^{-1}b$$

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

$i \Leftarrow 0$
$r \Leftarrow b - Ax$
$d \Leftarrow M^{-1}r$
$\delta_{new} \Leftarrow r^T d$
$\delta_0 \Leftarrow \delta_{new}$
While $i < i_{max}$ and $\delta_{new} > \varepsilon^2 \delta_0$ do
  $q \Leftarrow Ad$
  $\alpha \Leftarrow \frac{\delta_{new}}{d^T q}$
  $x \Leftarrow x + \alpha d$
  If i is divisible by 50
    $r \Leftarrow b - Ax$
  else
    $r \Leftarrow r - \alpha q$
  $s \Leftarrow M^{-1}r$
  $\delta_{old} \Leftarrow \delta_{new}$
  $\delta_{new} \Leftarrow r^T s$
  $\beta \Leftarrow \frac{\delta_{new}}{\delta_{old}}$
  $d \Leftarrow s + \beta d$
  $i \Leftarrow i + 1$

Preconditioning

- Preconditioning split into two steps
  - Preconditioner setup
  - Preconditioner application

$$A \rightsquigarrow M$$
$$y = M^{-1}x$$

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

- Want preconditioner which is fast to compute, and improves convergence.

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/

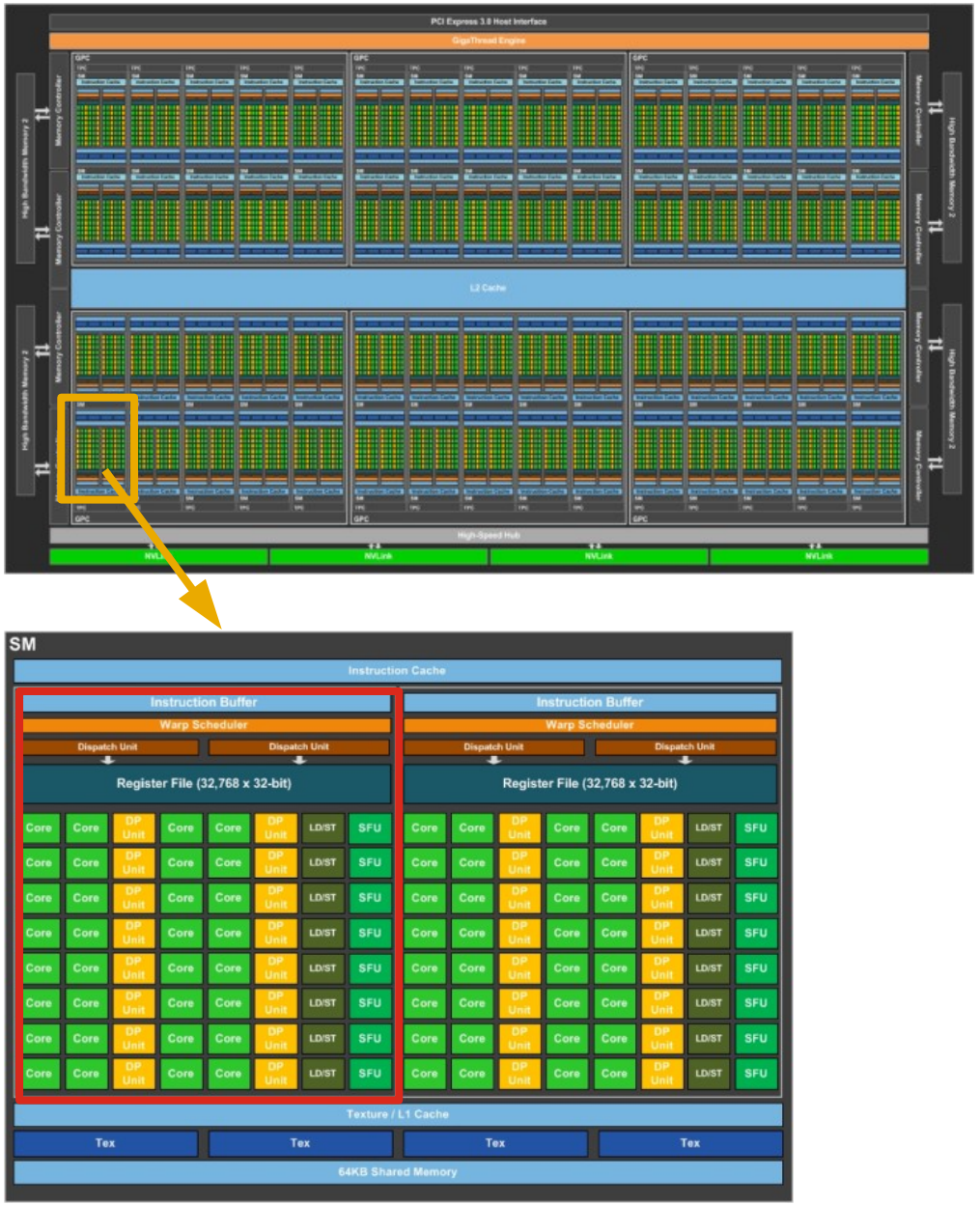
GPU programming 101

- 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

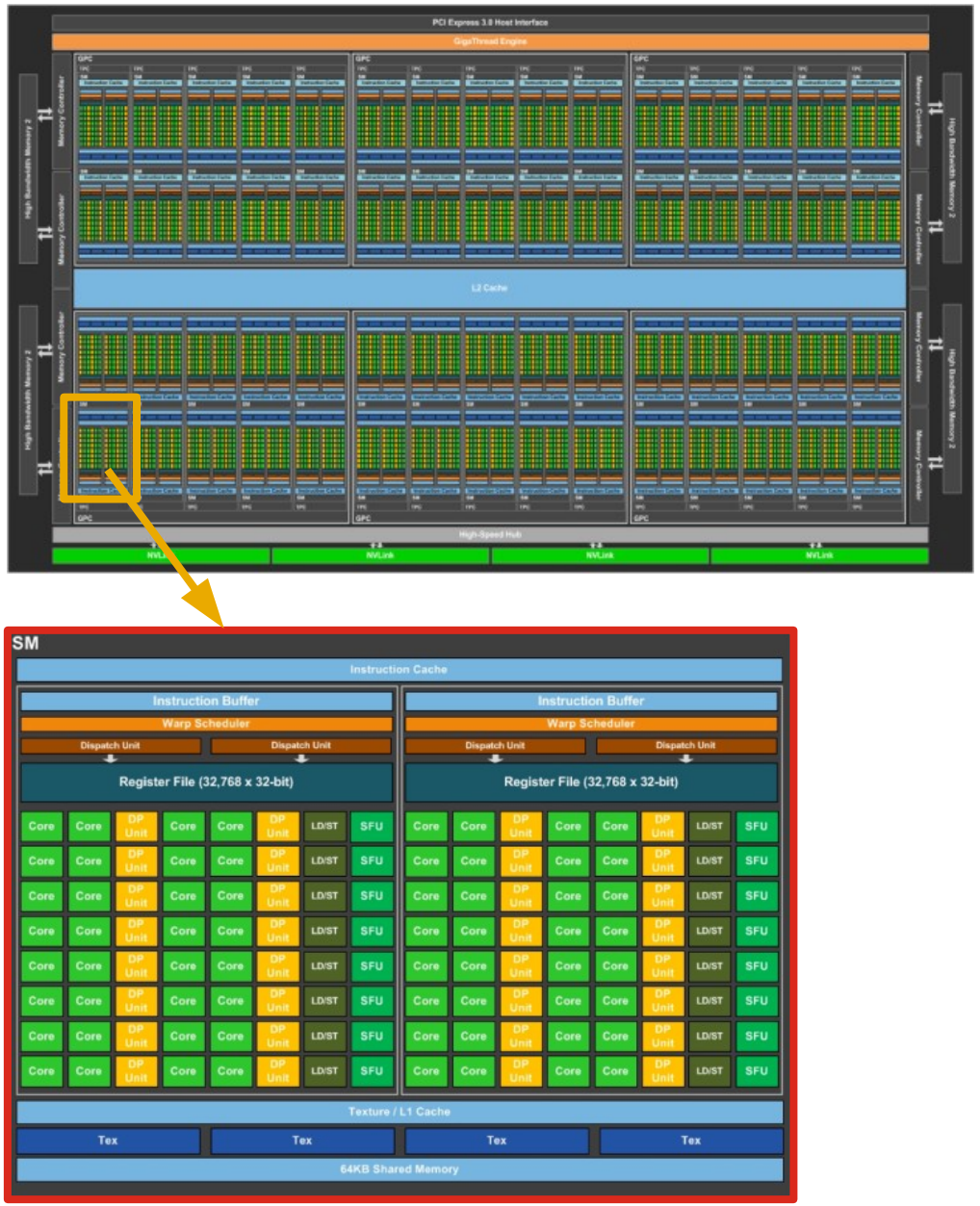
- NVIDIA P100 GPU
  - 4.7 TFLOPs DP performance
  - Up to 740 GB/s (1 : 51)
  - 56 SMs x 64 cores = 3584 cores!
- Hierarchical architecture / programming model:
  - Thread
    - Basic building block, assigned to 1 core
  - Warp
    - Group of 32 threads
    - Perfectly synchronized execution
    - Can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



source: devblogs.nvidia.com/parallelforall/

GPU programming 101

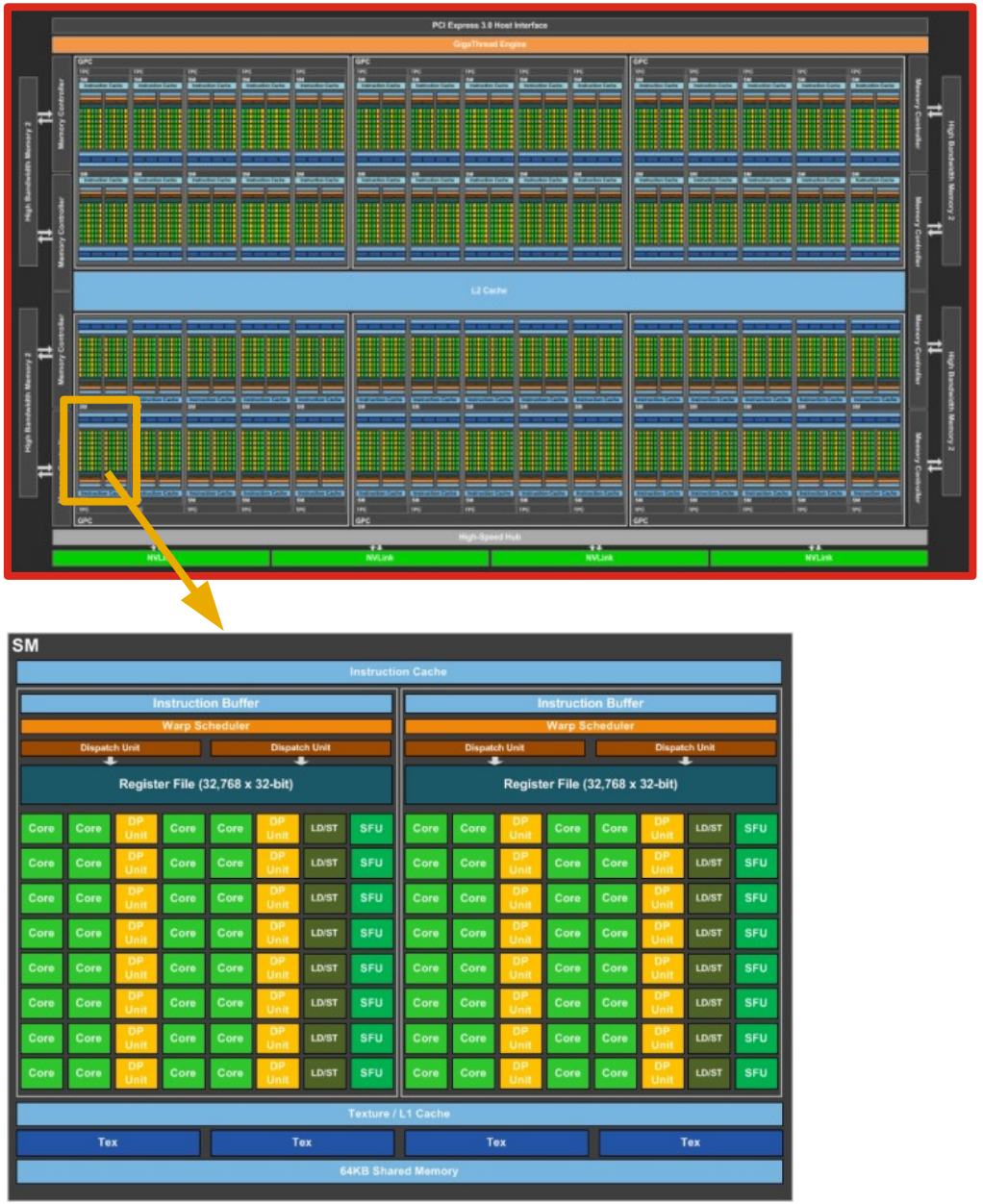
- Block
  - Group of several warps (≤ 64)
  - Can be explicitly synchronized
  - Can share data via shared memory (64KB)



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/

Batched routines

Problem characteristics*

problem size	#concurrent problems
~ 10 - 100	~ 1K - 1M
~ 100 - 10K	~ 1 - 1K
~ 10K - 1M	~ 1 - 10

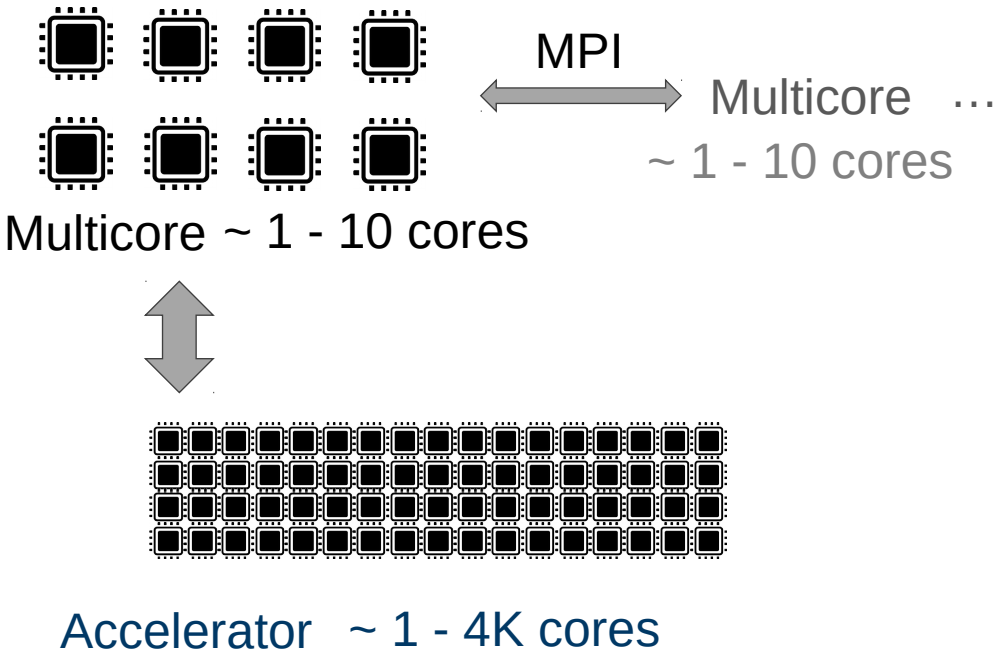
**2016 ICL LA-survey*

Batched routines

Problem characteristics*

problem size	#concurrent problems
~ 10 - 100	~ 1K - 1M
~ 100 - 10K	~ 1 - 1K
~ 10K - 1M	~ 1 - 10

Hardware characteristics



*2016 ICL LA-survey

Batched routines

Problem characteristics*

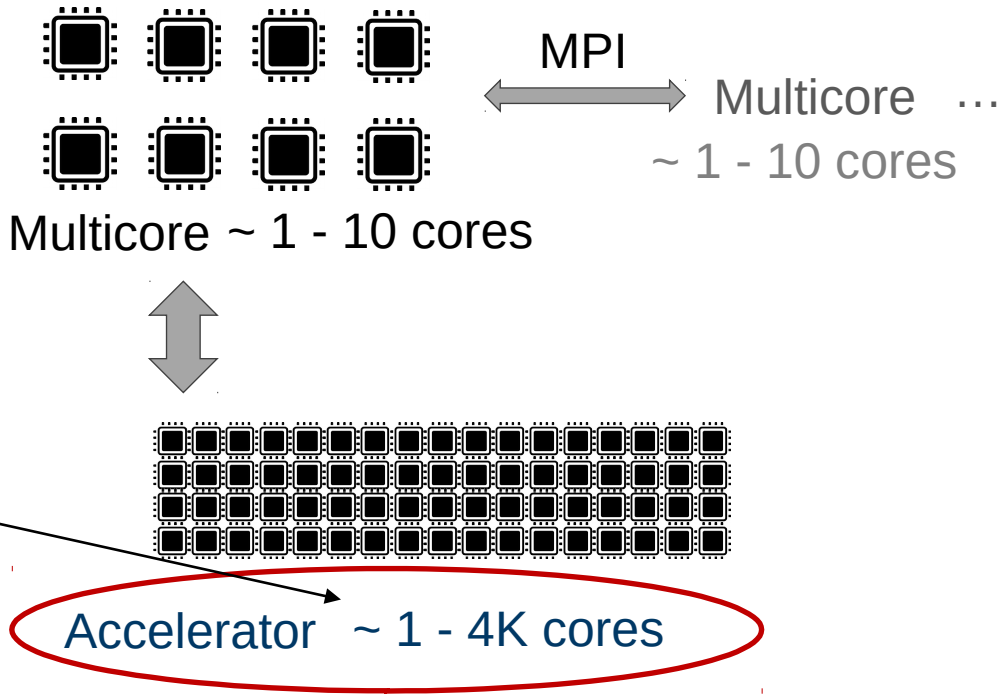
problem size	#concurrent problems
~ 10 - 100	~ 1K - 1M
~ 100 - 10K	~ 1 - 1K
~ 10K - 1M	~ 1 - 10

Solving a 10-100 problem using 1-4K cores?

Loop over items:

- *Hope items are scheduled in parallel*
- *Hope for efficient data access*
- *Hope kernel launch overhead is small*

Hardware characteristics



*2016 ICL LA-survey

Batched routines

Problem characteristics*

problem size	#concurrent problems
~ 10 - 100	~ 1K - 1M
~ 100 - 10K	~ 1 - 1K
~ 10K - 1M	~ 1 - 10

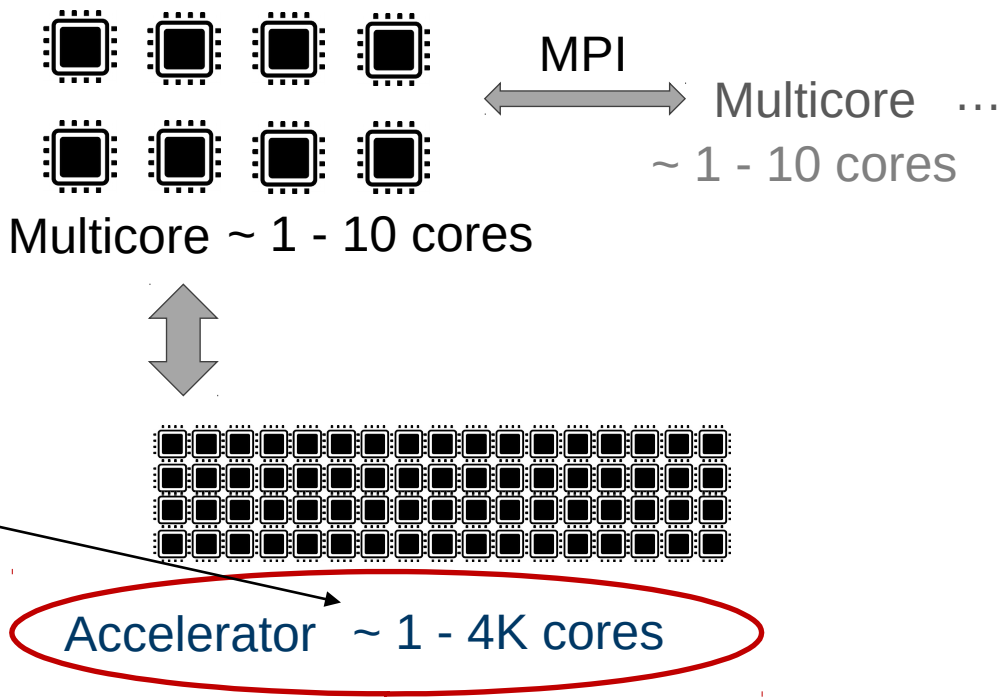
Solving a 10-100 problem using 1-4K cores?

Loop over items:

- *Hope items are scheduled in parallel*
- *Hope for efficient data access*
- *Hope kernel launch overhead is small*

Same computational kernel for *many small independent data items*:
"Batched" routines

Hardware characteristics



- Can achieve good perf. on GPUs
- *Batched for preconditioning?*

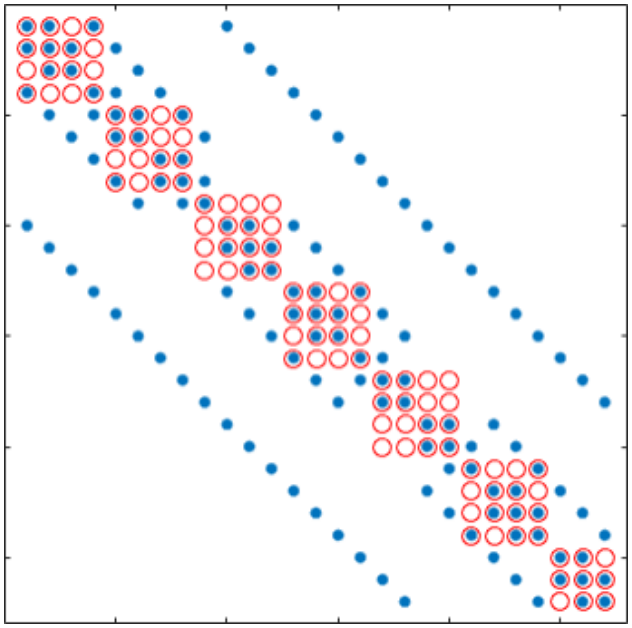
*2016 ICL LA-survey

Block-Jacobi preconditioning

- Scalar Jacobi
  - Scale with inverse of main diagonal

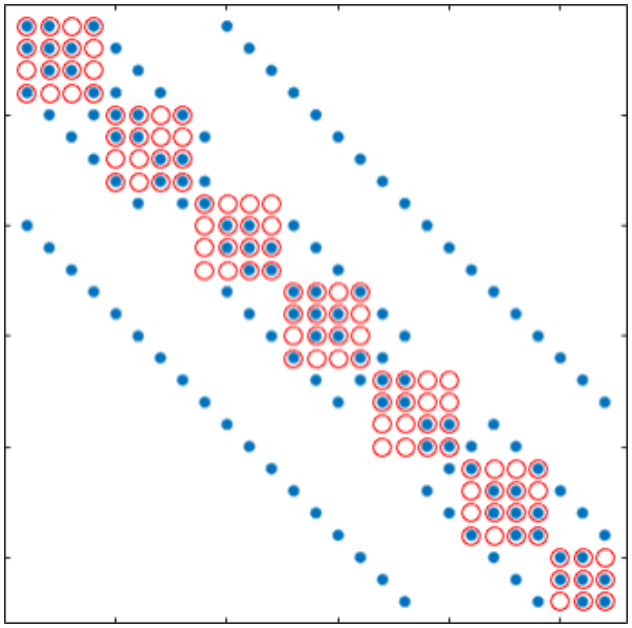
Block-Jacobi preconditioning

- Scalar Jacobi
  - Scale with inverse of main diagonal
- Block-Jacobi
  - Scale with inverses of diagonal blocks (possibly of different sizes!)
  - Can reflect the block structure of the problem
  - Often superior to scalar Jacobi



Block-Jacobi preconditioning

- Scalar Jacobi
  - Scale with inverse of main diagonal
- Block-Jacobi
  - Scale with inverses of diagonal blocks (possibly of different sizes!)
  - Can reflect the block structure of the problem
  - Often superior to scalar Jacobi
- Setup: invert / factorize blocks
- Application: gemv / triangular solve
- Can process each block independently! (Batched routine)

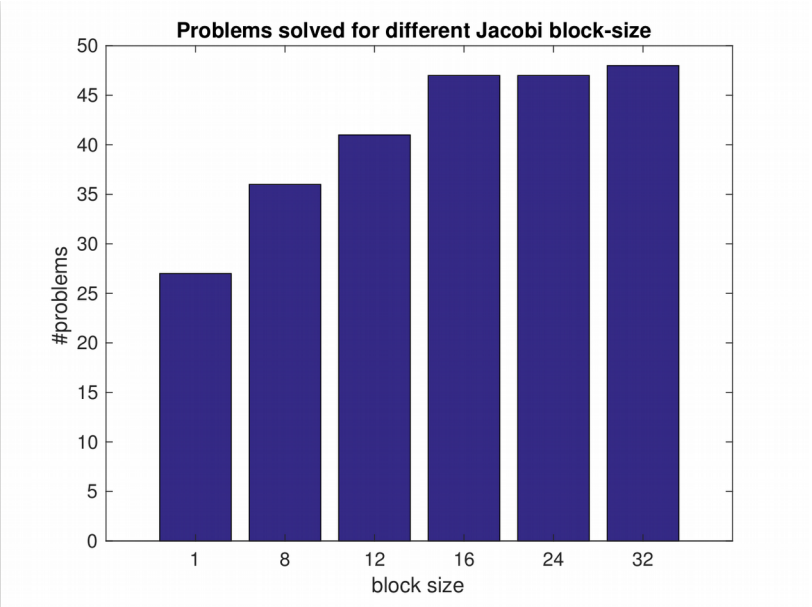


Benefits of block-Jacobi

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

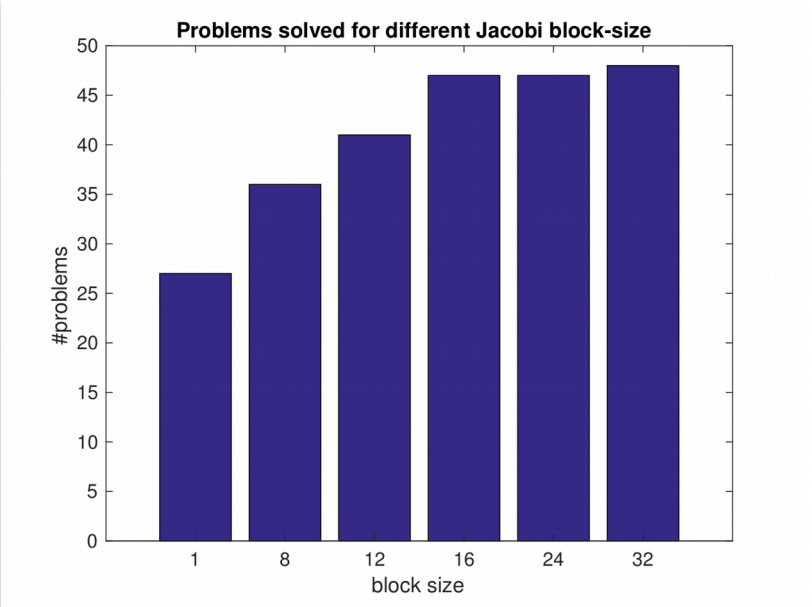
Benefits of block-Jacobi

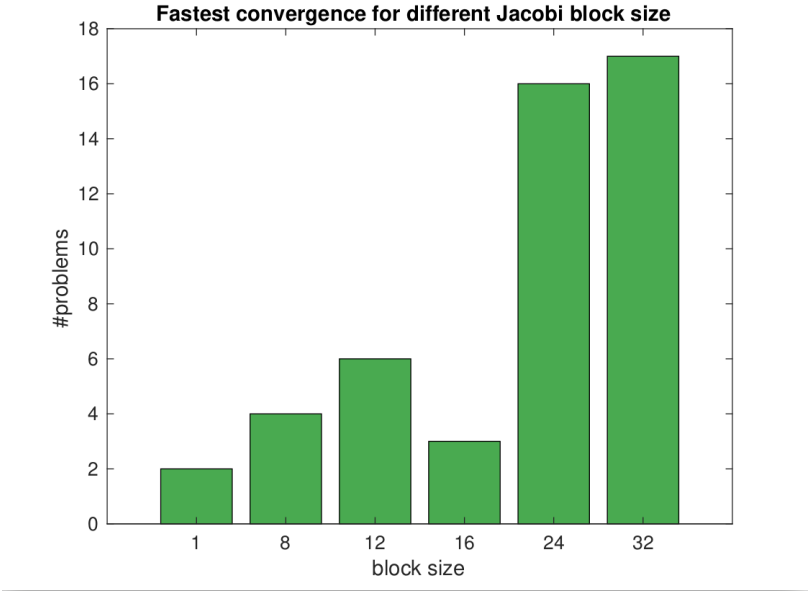
- 56 matrices from SuiteSparse
- MAGMA-sparse open source library
  - IDR solver
  - Scalar Jacobi preconditioner
  - Supervariable blocking
    - Detects block structure of the matrix
- Improves the robustness of the solver



Benefits of block-Jacobi

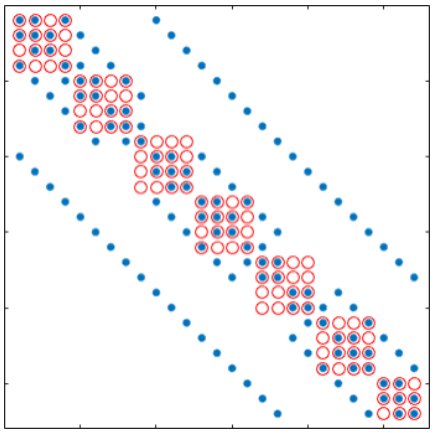
- 56 matrices from SuiteSparse
- MAGMA-sparse open source library
  - IDR solver
  - Scalar Jacobi preconditioner
  - Supervariable blocking
    - Detects block structure of the matrix
- Improves the robustness of the solver
- Decreases time-to-solution





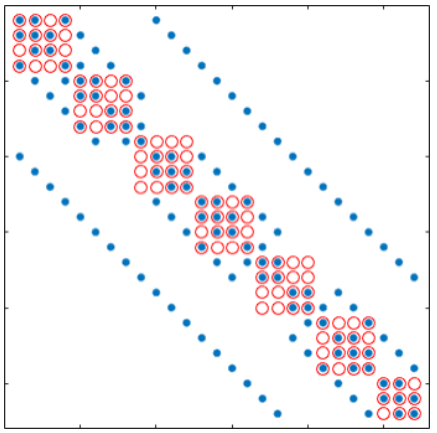
General Ideas

- Restrict block size to 32x32
  - Large block sizes require more memory to store the preconditioner matrix



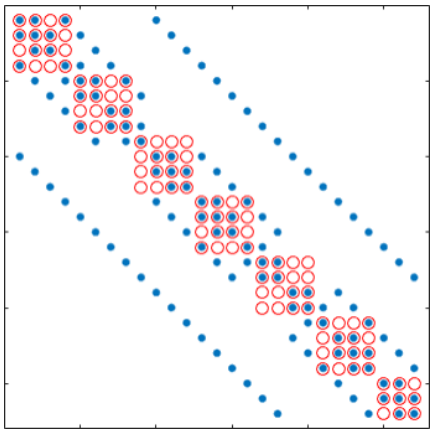
General Ideas

- Restrict block size to 32x32
  - Large block sizes require more memory to store the preconditioner matrix
- Use a single warp to process the whole block (one thread per row / column)
  - No need for explicit synchronization



General Ideas

- Restrict block size to 32x32
  - Large block sizes require more memory to store the preconditioner matrix
- Use a single warp to process the whole block (one thread per row / column)
  - No need for explicit synchronization
- Use the large register file to store the entire block
  - Read/write from mem. once
  - Comm. via warp shuffles
  - Avoids load/store instructions
- **Do pivoting implicitly (without swapping the rows)**



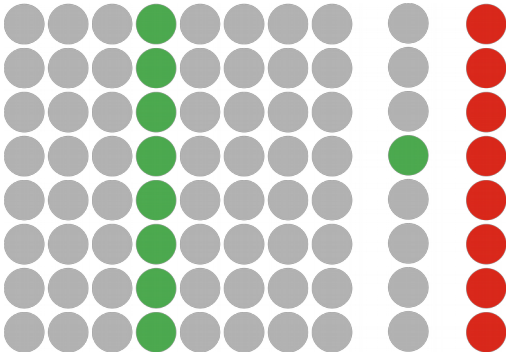
Block-Jacobi setup & application ecosystem

Setup

Application

Gauss-Jordan inversion

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



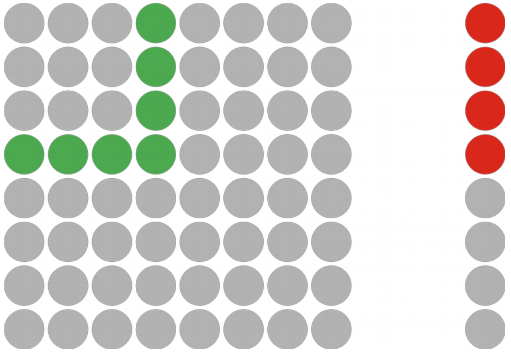
H. Anzt et al. “Batched Gauss-Jordan Elimination for Block-Jacobi Preconditioner Generation on GPUs”, PMAM’17

Gauss-Huard decomposition

LU factorization

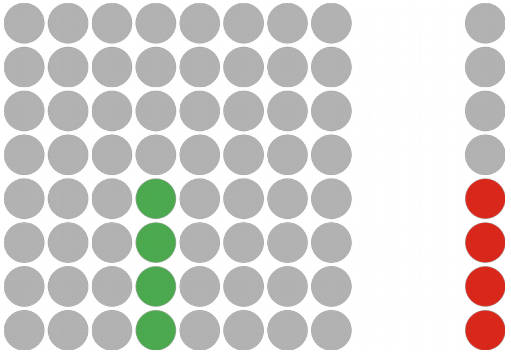
Decomposition-based
($2/3n^3 + 2n^2$ FLOPS)

Gauss-Huard solve



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

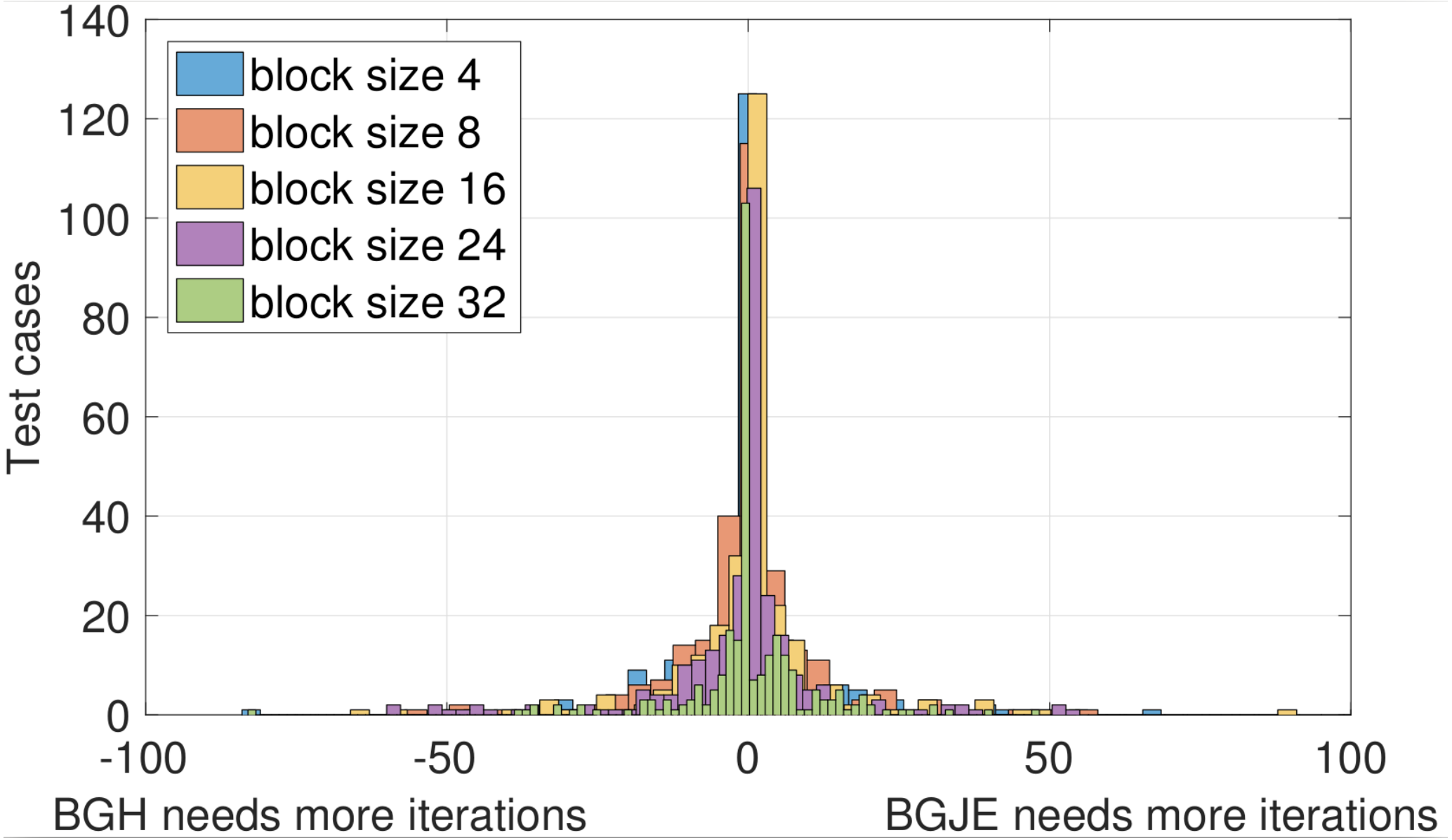
2x triangular solve



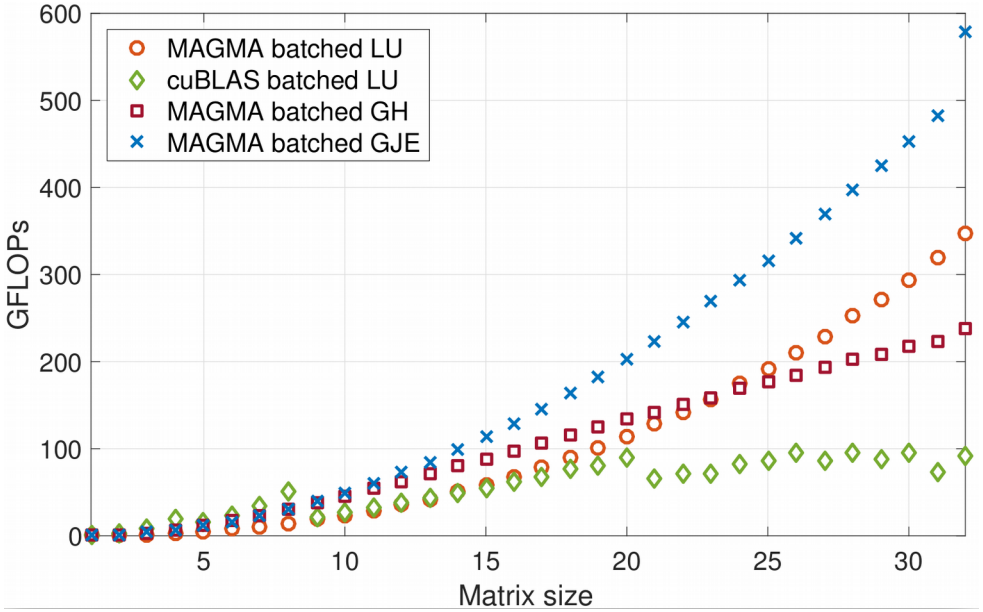
H. Anzt et al. “Flexible-Size Batched LU for Small Matrices and its Integration into Block-Jacobi Preconditioning”, ICPP’17 (to appear)

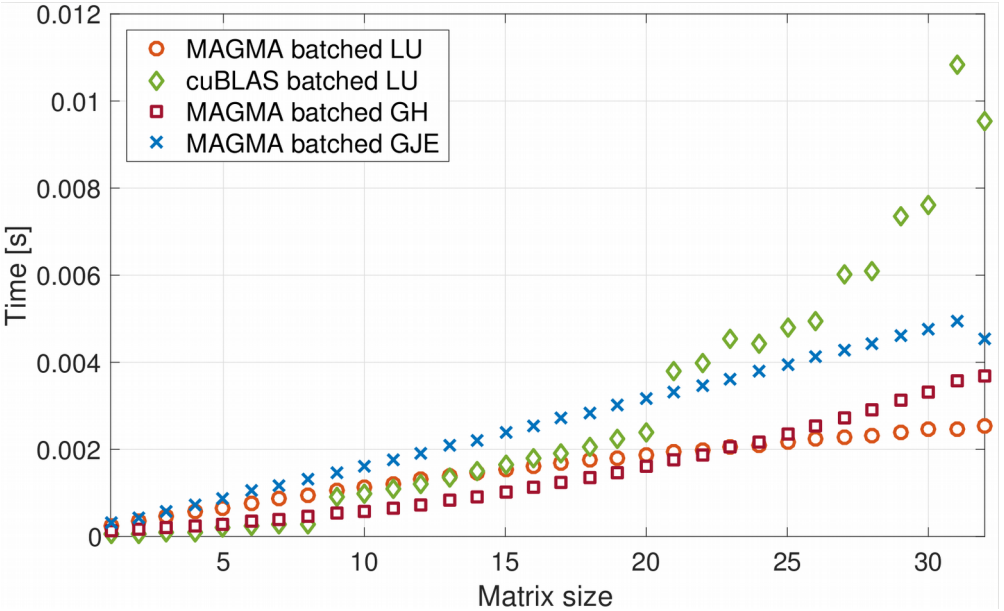
- read - write

Inversion?!

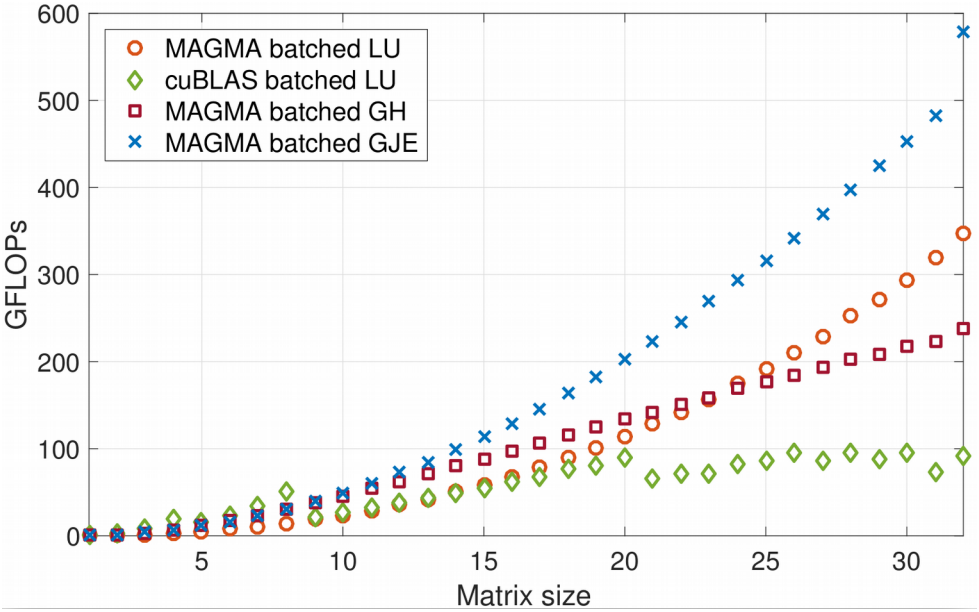

block size 4
block size 8
block size 16
block size 24
block size 32
Test cases
140
120
100
80
60
40
20
0
-100
-50
0
50
100
BGH needs more iterations
BGJE needs more iterations

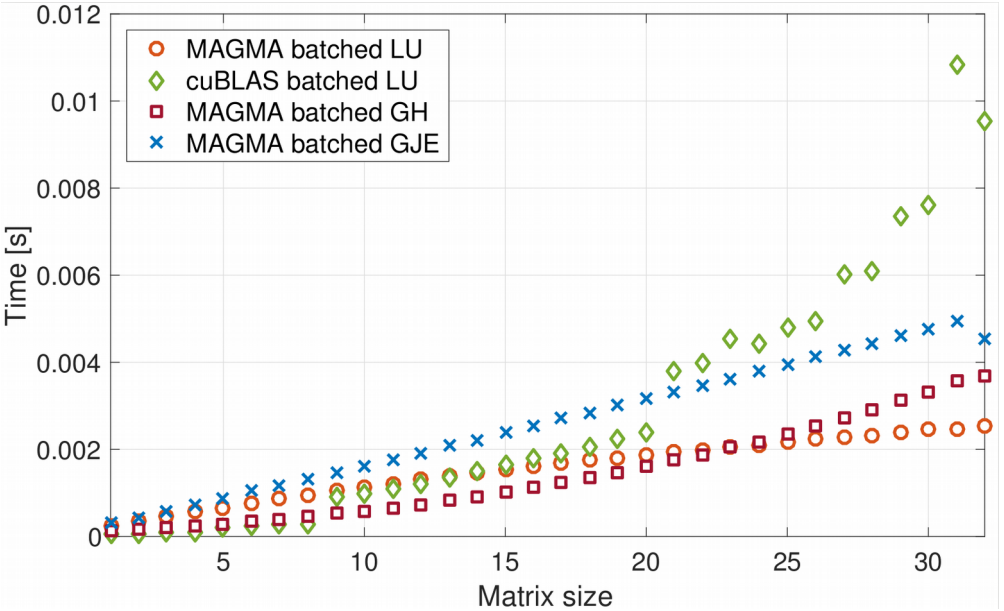
Batched routines performance

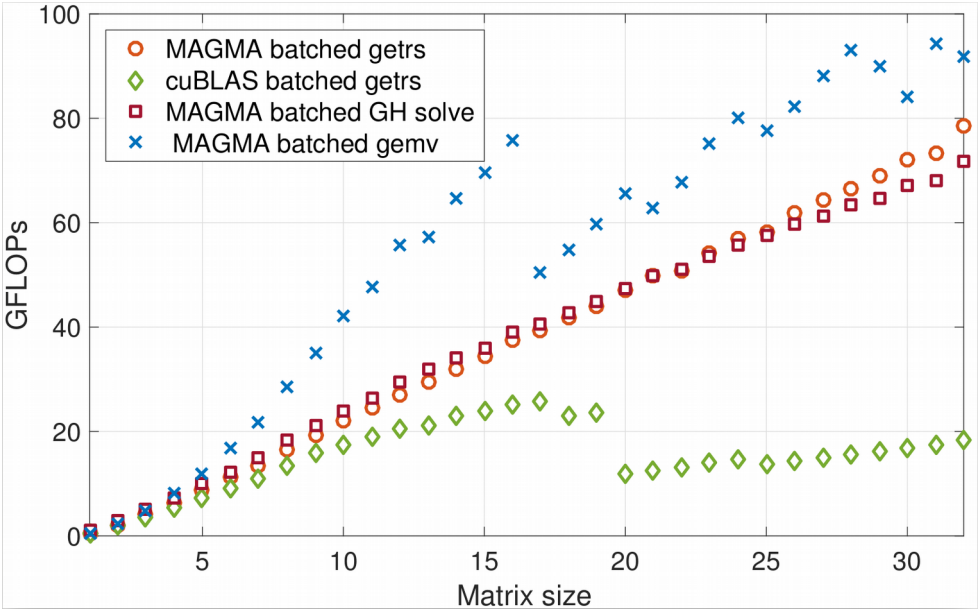


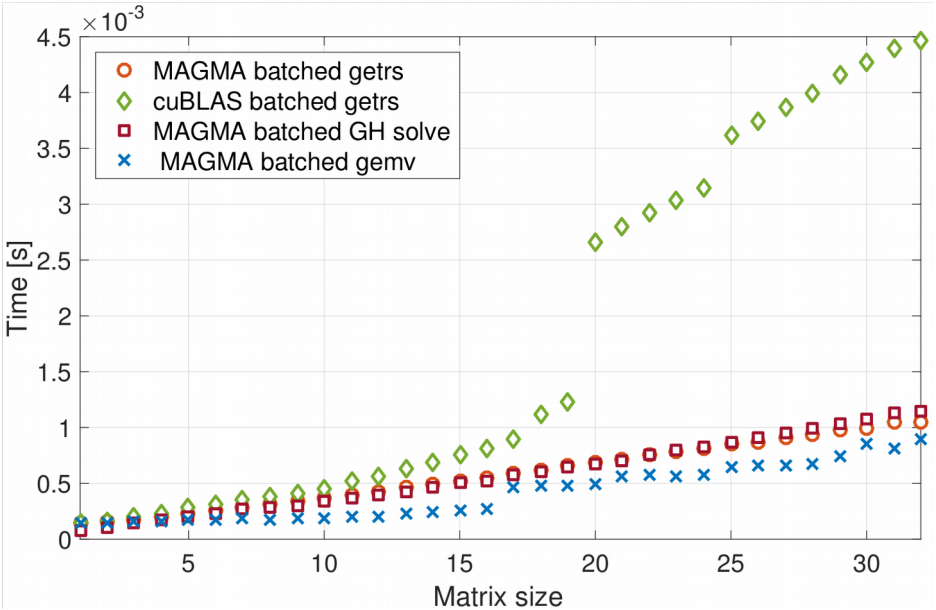


Batched routines performance



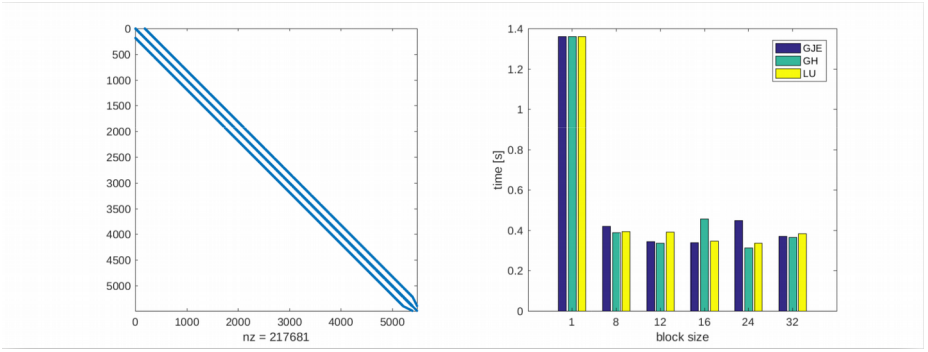




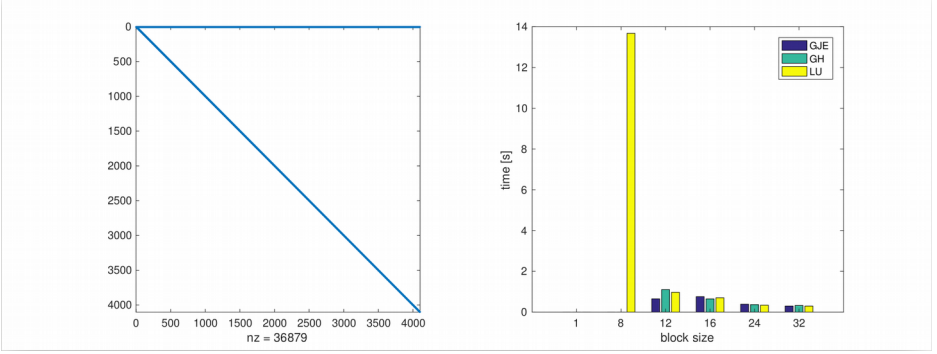


Complete solver runtime

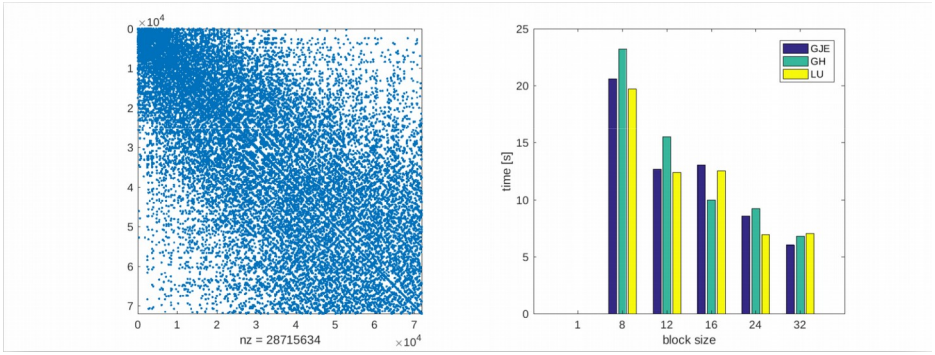
s2rmt3m1



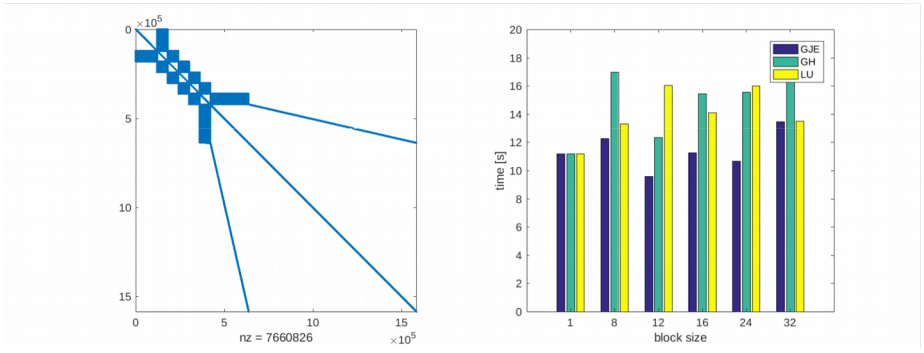
Chebyshev3



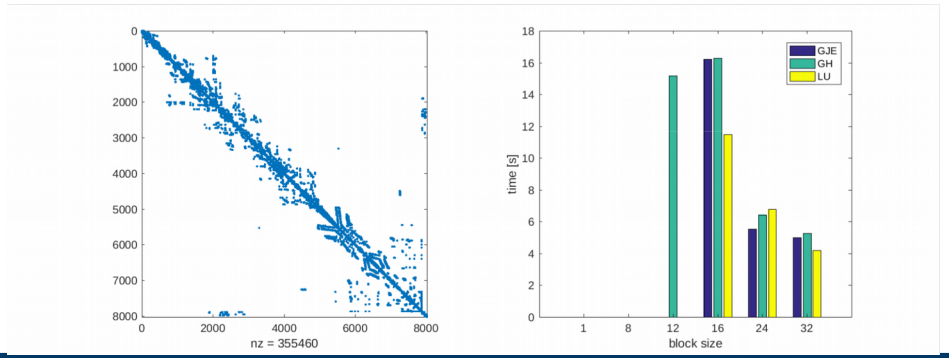
nd24k



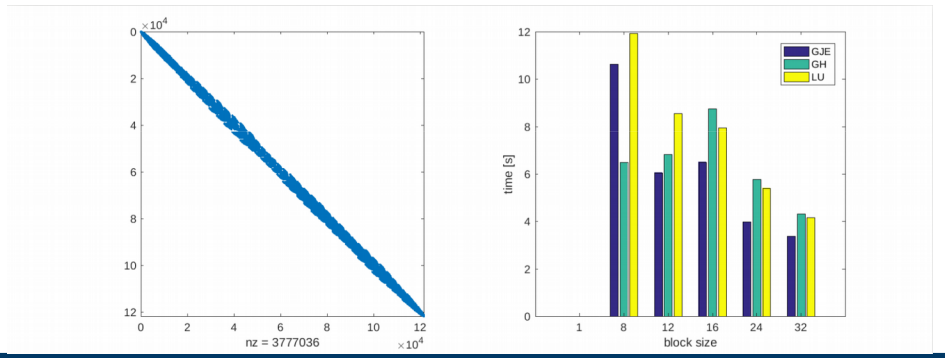
G3_circuit



bcsstk38

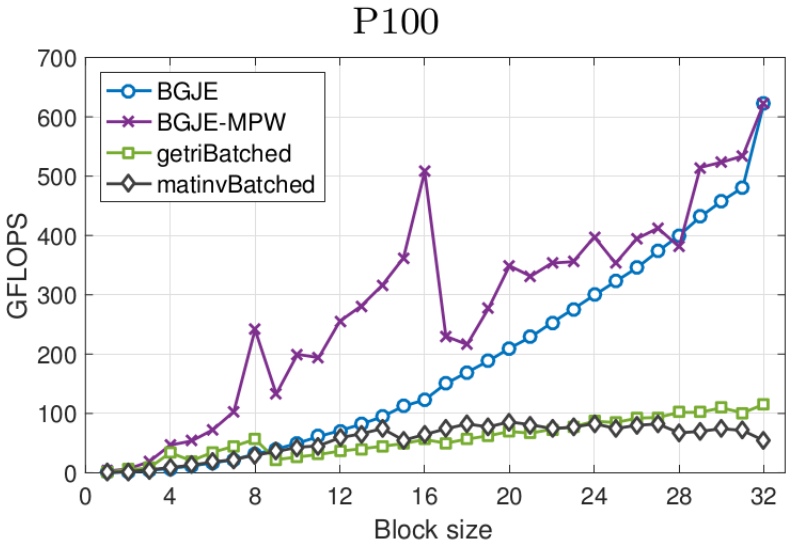


ship_003



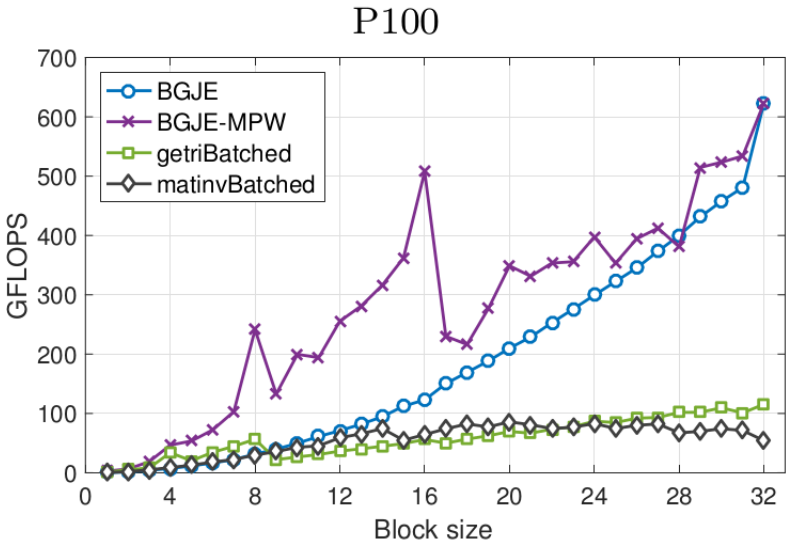
Flexible-size batched routines & future research

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

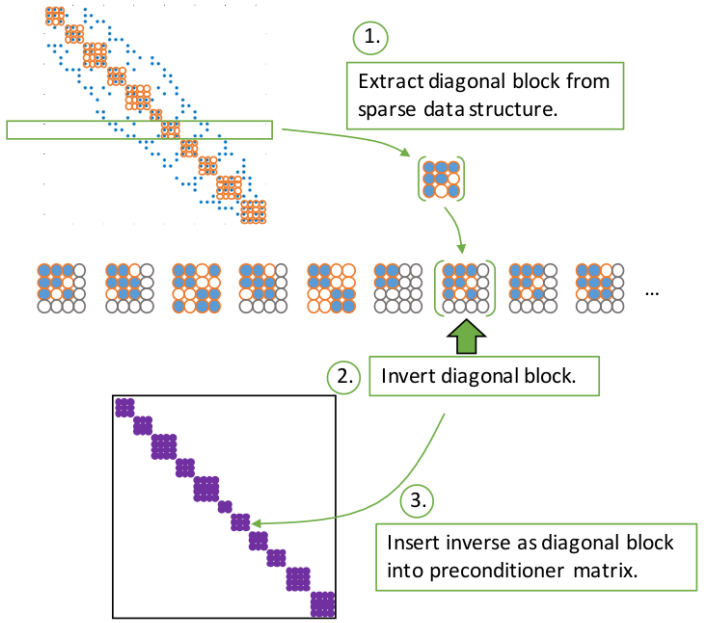


Flexible-size batched routines & future research

- 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)
  - Currently supported, but not yet optimized
  - How to combine this with multiple problems per warp?
    - Remember: entire warp executes the same instruction!
    - Current solution: padding



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/mpi_magdeburg_2017_06

https://github.com/gflegar/talks/raw/master/mpi_magdeburg_2017_06/slides.pdf

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

THE UNIVERSITY OF TENNESSEE KNOXVILLE

UNIVERSITAT JAUME I