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:

6 4 5

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

3 1 2

Goran Flegar

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

Scan me for slides!

github.com/gflegar/talks/mpi_magdeburg_2017_06

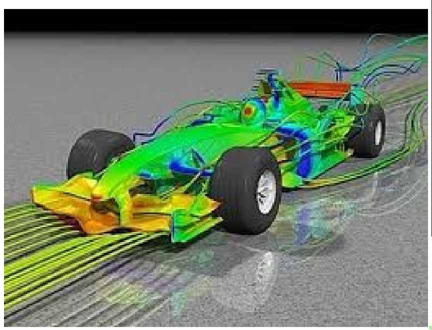
flegar@uji.es

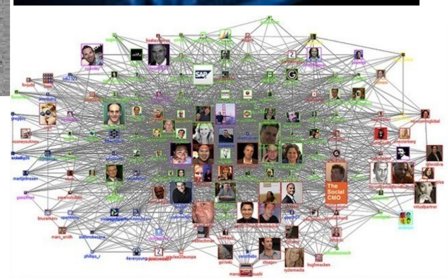
Problem setting

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

- Solve it using an iterative Krylov method

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





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

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

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

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

Preconditioning

- Preconditioning split into two steps
  - Preconditioner setup
  - Preconditioner application
- Trade-off: faster convergence, but more work per iteration

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

GPU programming 101

- NVIDIA P100 GPU
  - 4.7 TFLOPs DP performance
  - Up to 740 GB/s (1 : 51)
  - s56 SMs x 64 cores = 3584 cores!
- 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

- NVIDIA P100 GPU
  - 4.7 TFLOPs DP performance
  - Up to 740 GB/s (1 : 51)
  - s56 SMs x 64 cores = 3584 cores!
- 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

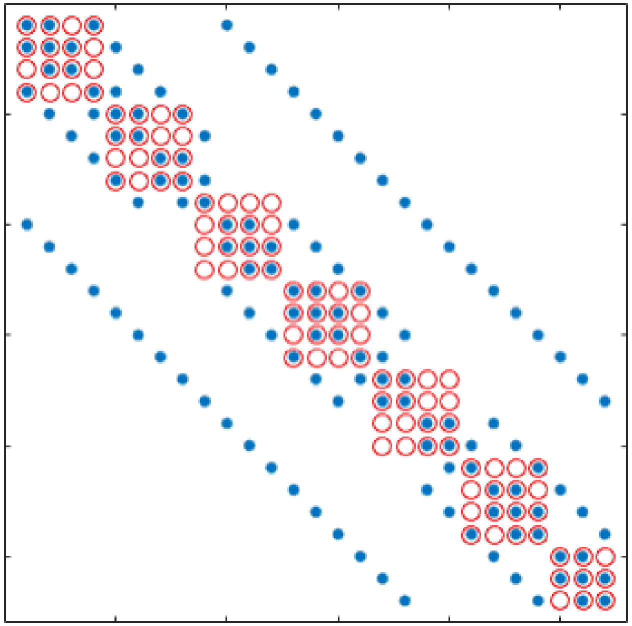
- Programming model:
  - Block
    - Group of several warps (≤ 64)
    - Can be explicitly synchronized
    - Can share data via shared memory (64KB)
  - Grid
    - Group of blocks
    - Cannot synchronize!
    - Global memory (12 or 16GB)
    - Simple caches
    - Atomics

source: devblogs.nvidia.com/parallelforall/

Batched routines

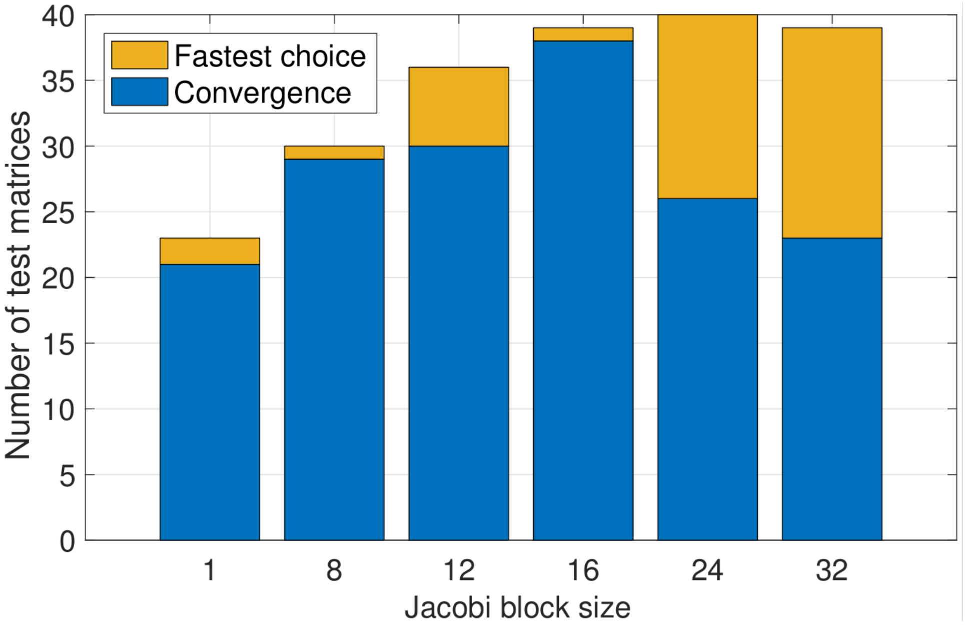
Block-Jacobi preconditioning

- 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: GEMM / triangular solve
- Can process each block independently! (Batched routine)



Benefits of block-Jacobi

- 40 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
  - More problems converge
- 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

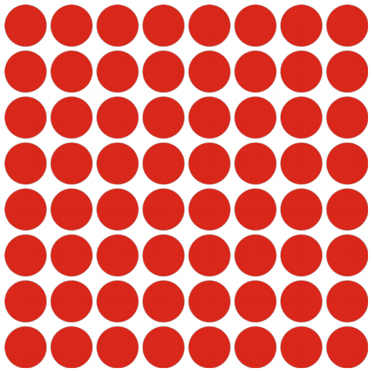
General Ideas

- 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)

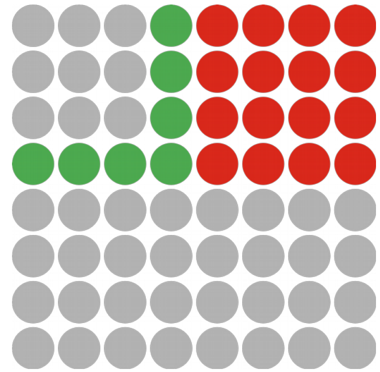
Block-Jacobi setup & application ecosystem

Setup

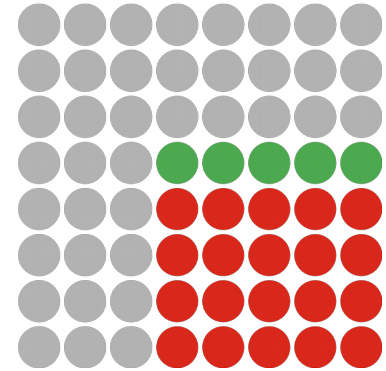
Gauss-Jordan inversion



Gauss-Huard decomposition



LU factorization

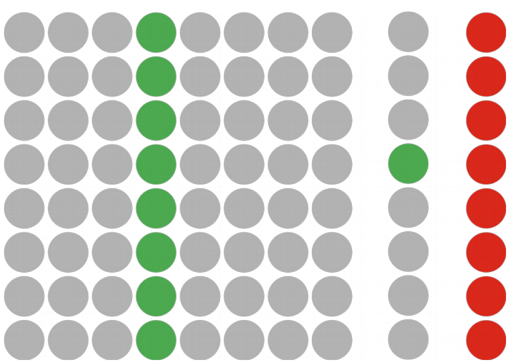


Inversion-based
($2n^3 + 2n^2$ FLOPS)

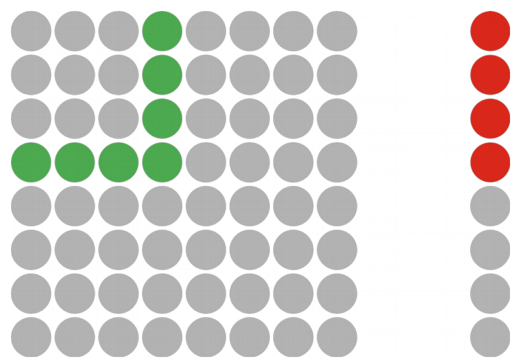
Decomposition-based
$2/3n^3 + 2n^2$ FLOPS

Application

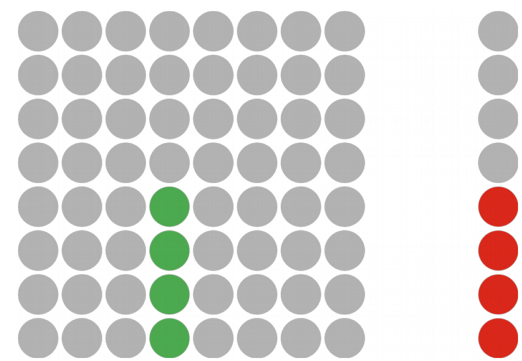
matrix-vector multiply



Gauss-Huard solve



2x triangular solve



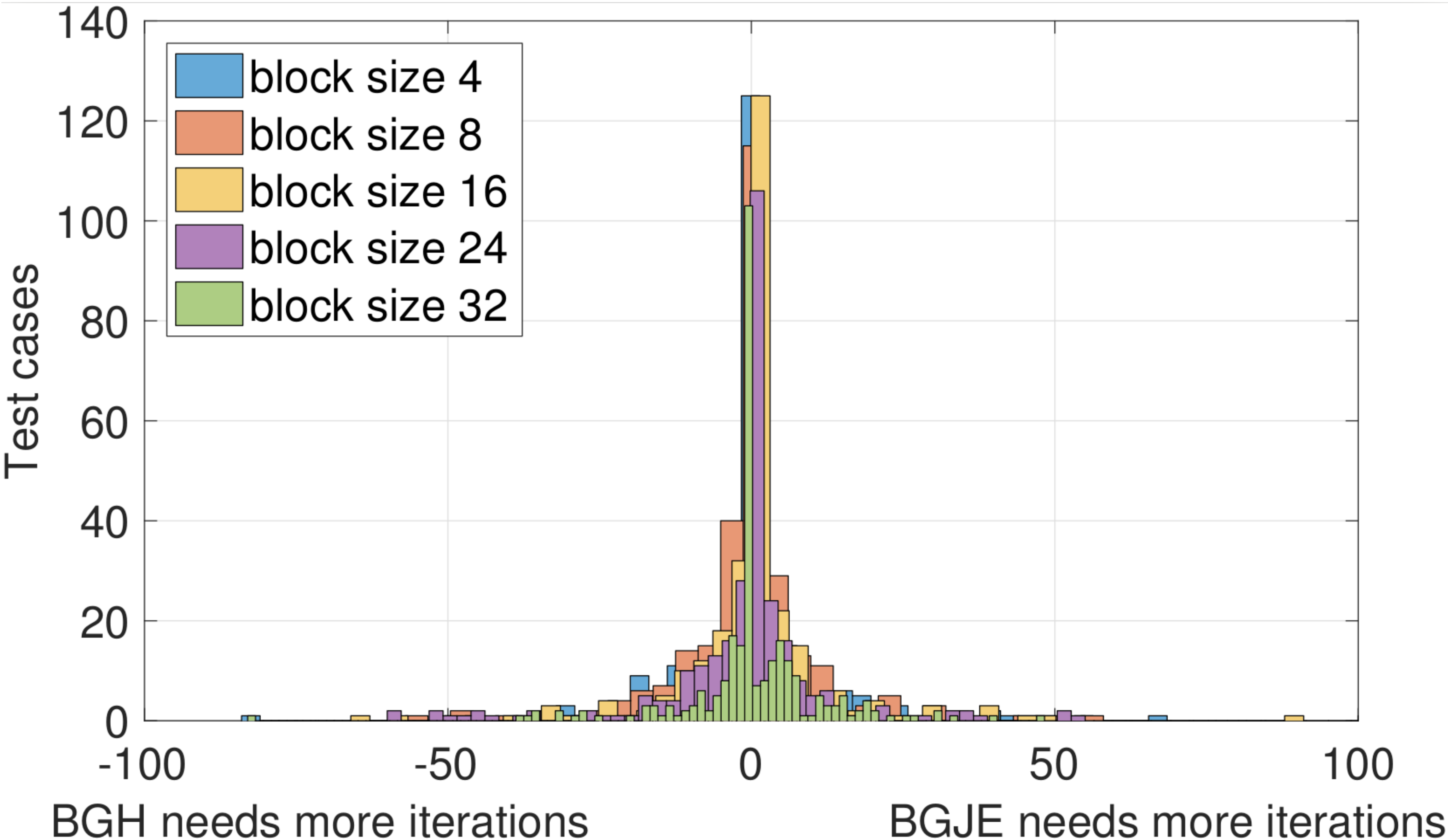
H. Anzt et al. “Batched Gauss-Jordan Elimination for Block-Jacobi Preconditioner Generation on GPUs”, PMAM’17

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

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

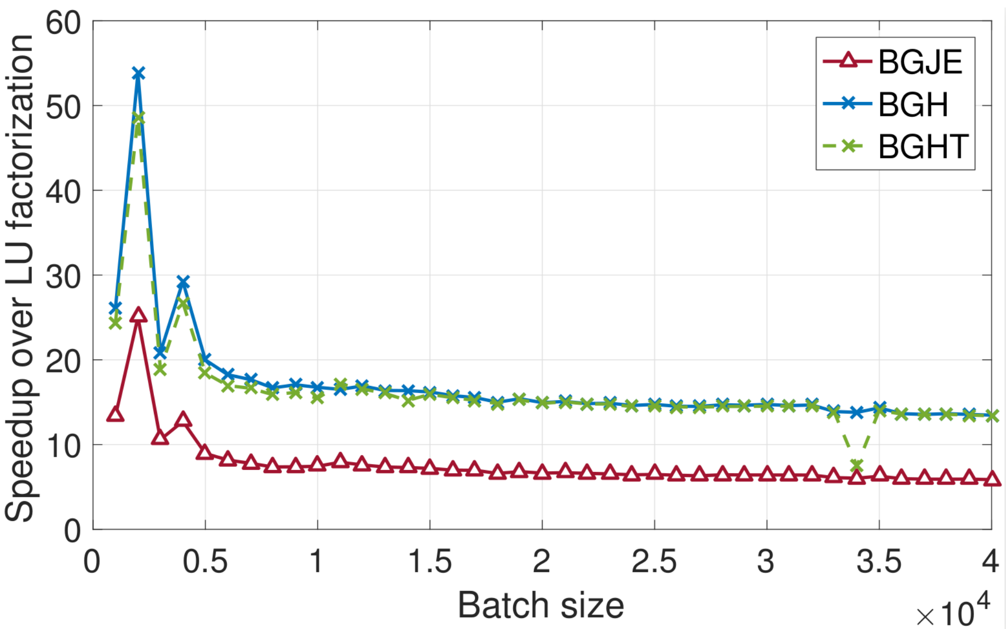
- read   - write

Inversion?!

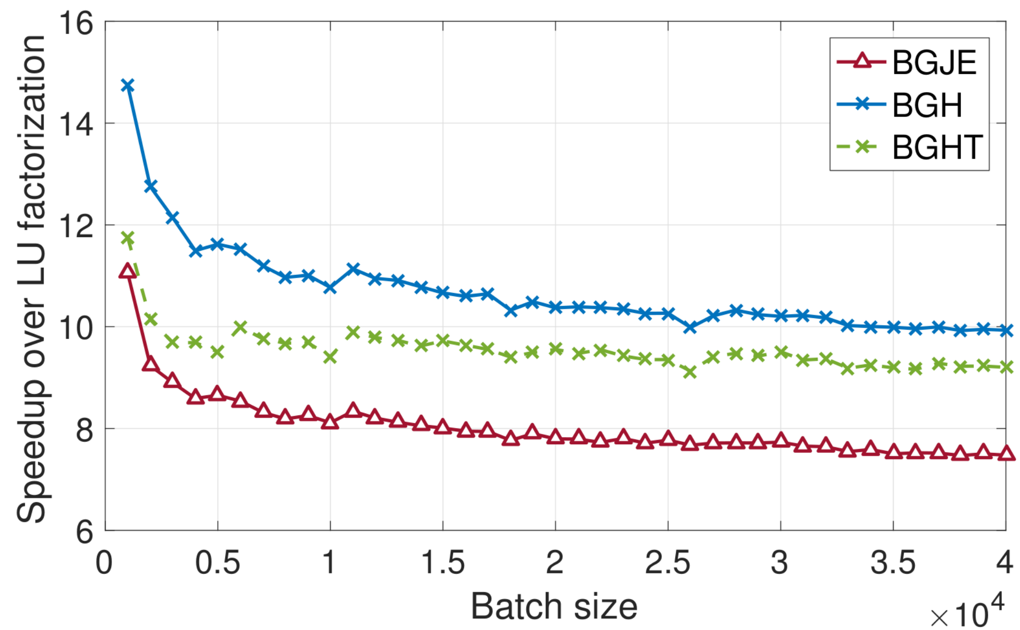

block size 4
block size 8
block size 16
block size 24
block size 32
Test cases
140
120
100
80
60
40
20
0
-100
-50
0
50
100
BGH needs more iterations
BGJE needs more iterations

Factorization & inversion performance

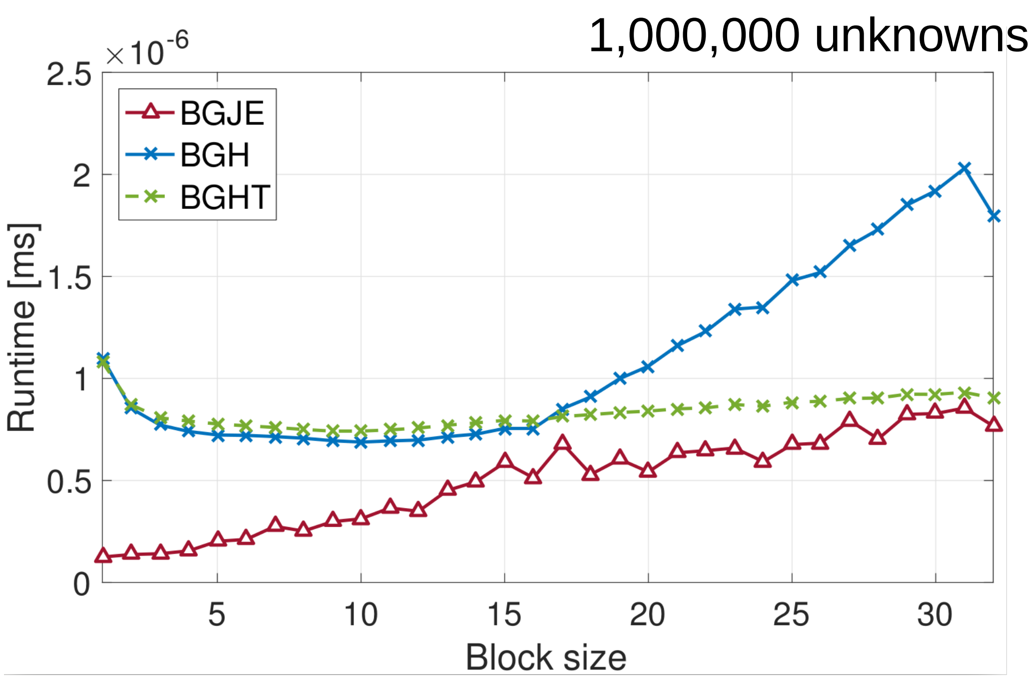
Block size 16



Block size 32

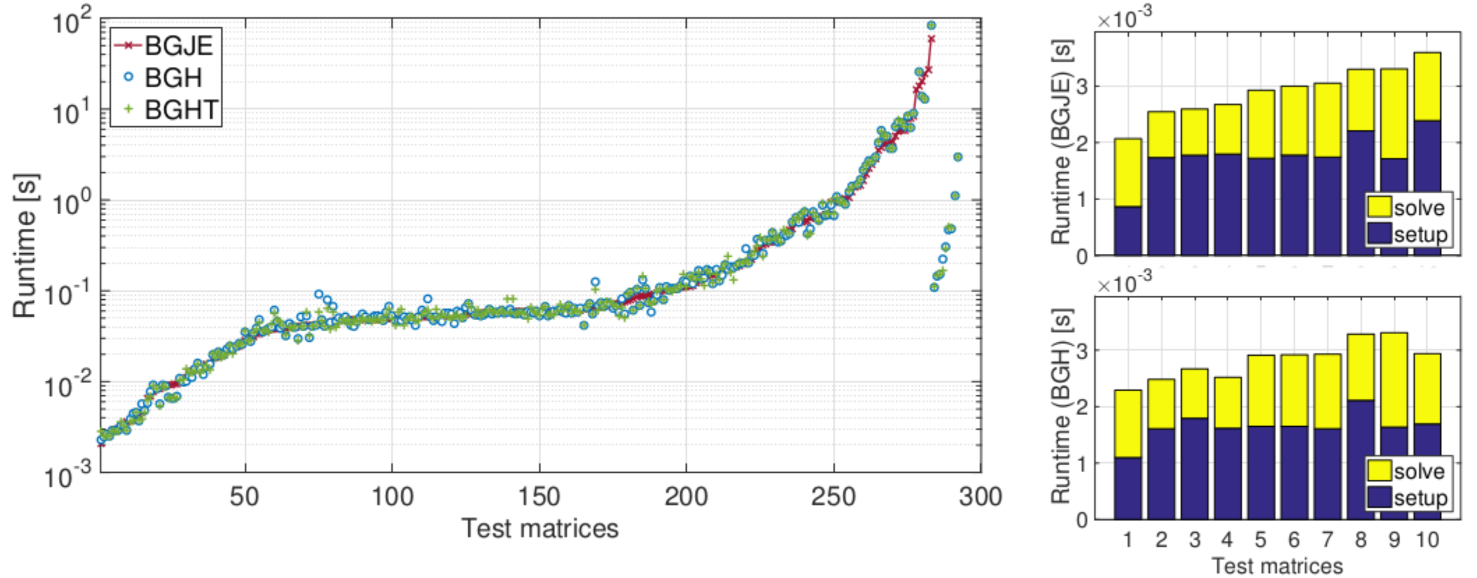


Application (GEMV / solve) performance

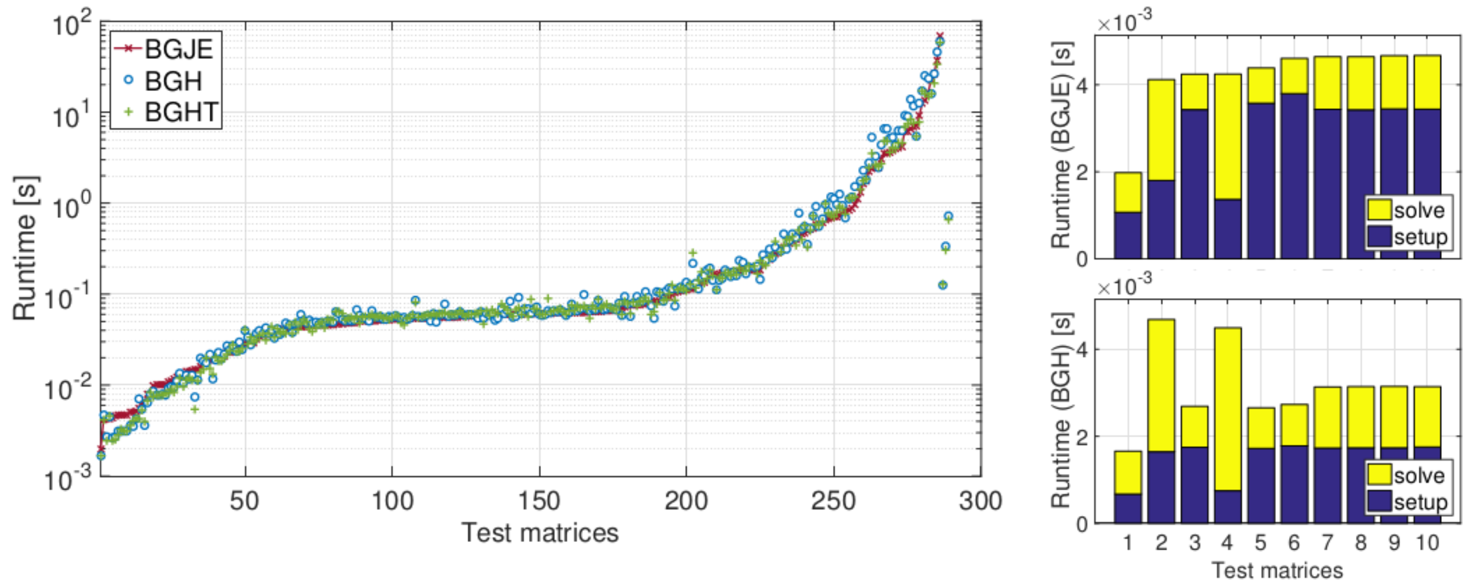


Complete solver runtime

Block size 16



Block size 32



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

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