Workshop on Batched, Reproducible, and Reduced Precision BLAS
Atlanta, GA 02/25/2017

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

Hartwig Anzt
Joint work with Goran Flegar, Enrique Quintana-Orti

http://www.icl.utk.edu/~hanzt/talks/batched_bjac.pdf

ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

Workshop on Batched, Reproducible, and Reduced Precision BLAS
Atlanta, GA 02/25/2017

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

Hartwig Anzt
Joint work with Goran Flegar, Enrique Quintana-Orti

ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

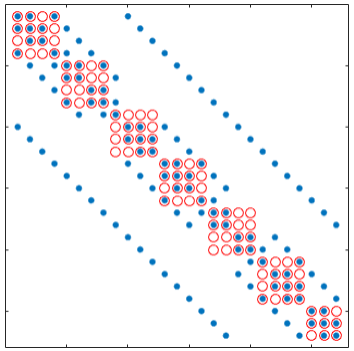
Pitch from the sparse community:

- Small size (<30), "somewhat" flexible
- Add inversion routine to Batched-Blas
- Go for Gauss-Jordan Elimination instead of LU

http://www.icl.utk.edu/~hanzt/talks/batched_bjac.pdf

Iterative linear system solvers

- Goal: solve linear system $Ax = b, A \in \mathbb{R}^{n \times n}$
- Use iterative **Krylov method** for generating solution approximation
- Convergence typically benefits from using a **preconditioner**
- We focus on **Jacobi preconditioners** based on **diagonal scaling**
  - $P = (diag(A))^{-1}$
  - Provide a high level of parallelism
  - Attractive for many-core accelerators like GPUs
  - Scalar Jacobi acts on the main diagonal
  - block-Jacobi acts on the block-diagonal



Block-Jacobi preconditioner

- Can reflect the block-structure of sparse matrices (e.g. Higher-Order FEM)
- Often superior to scalar Jacobi preconditioner
- Light-weight preconditioner (compared to ILU)
- Parallel preconditioner application (batched trsv / SpMV + vector scaling)
  - Inversion of diag. blocks in prec. setup + SpMV in prec. application
  - Factorization of diag. blocks in prec. setup + trsv in prec. application

Block-Jacobi preconditioner

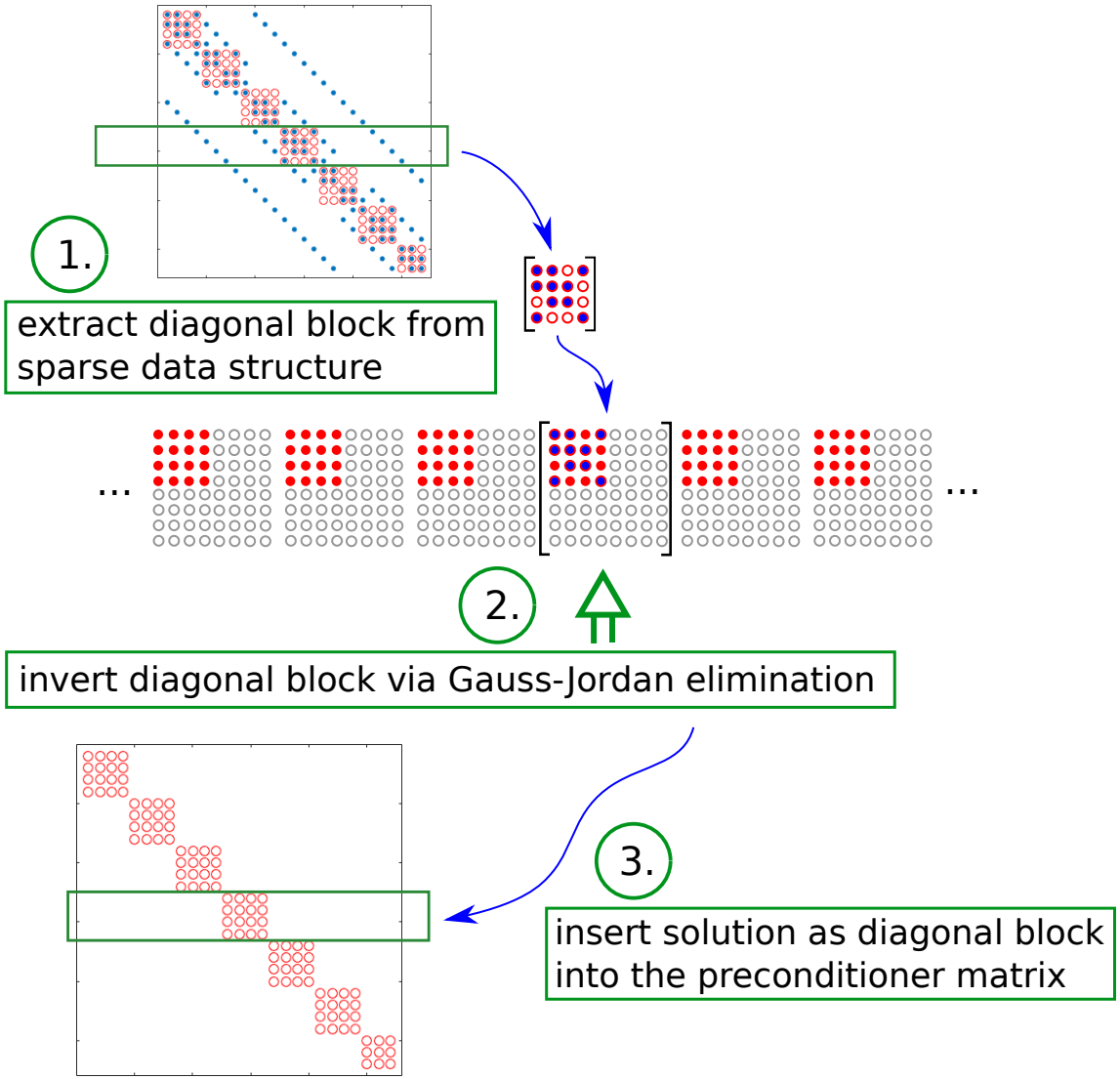
- Can reflect the block-structure of sparse matrices (e.g. Higher-Order FEM)
- Often superior to scalar Jacobi preconditioner
- Light-weight preconditioner (compared to ILU)
- Parallel preconditioner application (batched trsv / SpMV + vector scaling)
  - Inversion of diag. blocks in prec. setup + SpMV in prec. application
  - Factorization of diag. blocks in prec. setup + trsv in prec. application

Block-Jacobi preconditioner

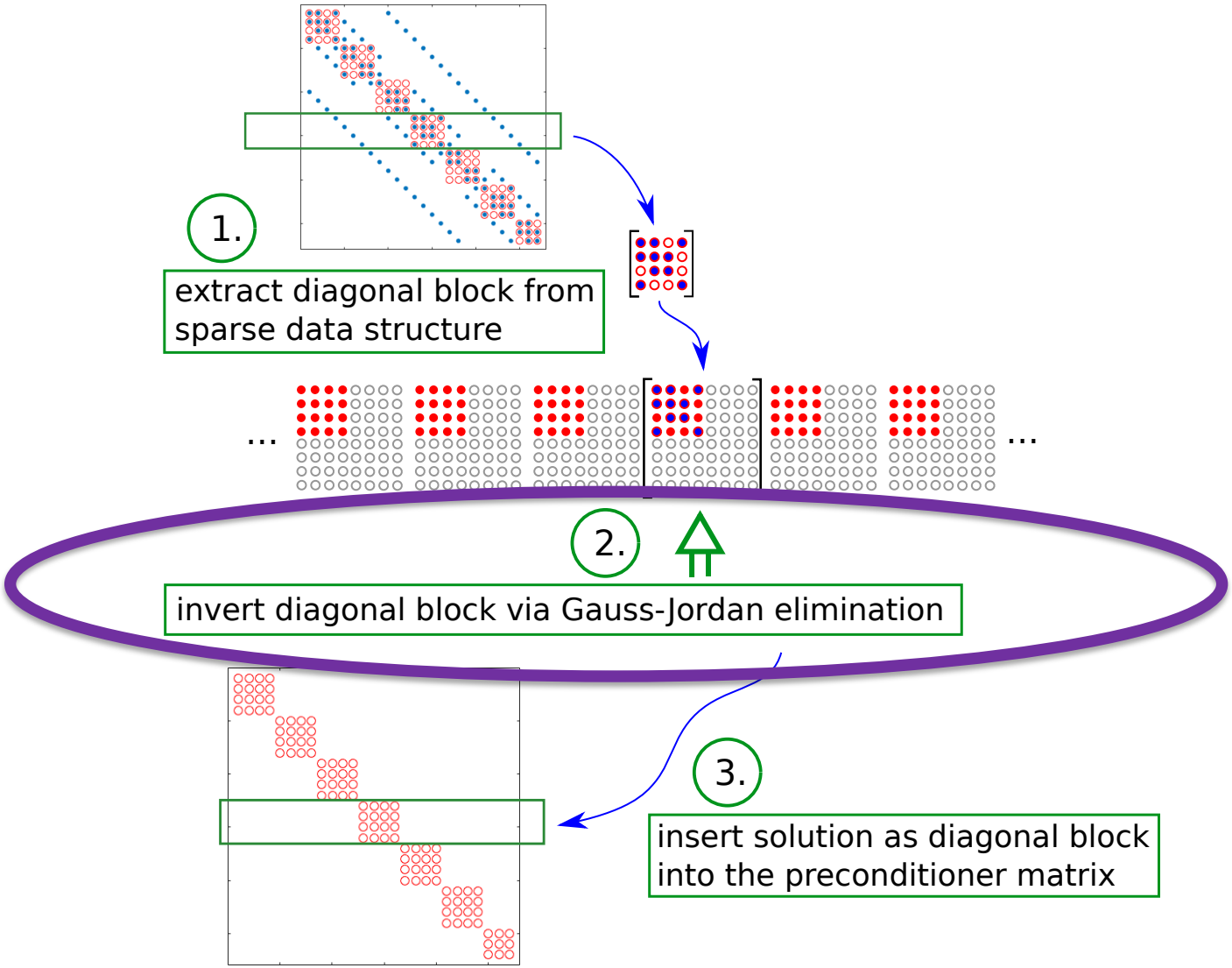
- Can reflect the block-structure of sparse matrices (e.g. Higher-Order FEM)
- Often superior to scalar Jacobi preconditioner
- Light-weight preconditioner (compared to ILU)
- Parallel preconditioner application (batched trsv / SpMV + vector scaling)
  - Inversion of diag. blocks in prec. setup + SpMV in prec. application
  - Factorization of diag. blocks in prec. setup + trsv in prec. application

- Preconditioner setup (inversion of diagonal blocks) can become a challenge
  - Batched inversion of many small systems
  - Diagonal blocks are typically of small size

Block-Jacobi preconditioner generation



Block-Jacobi preconditioner generation



Batched Gauss-Jordan elimination (BGJE) on GPUs

- Flexible size up to 32 x 32
- One warp per system
- Inversion in registers
- Communication via __shfl()
- Implicit pivoting:
  - accumulate row/col swaps
  - realize them when writing results

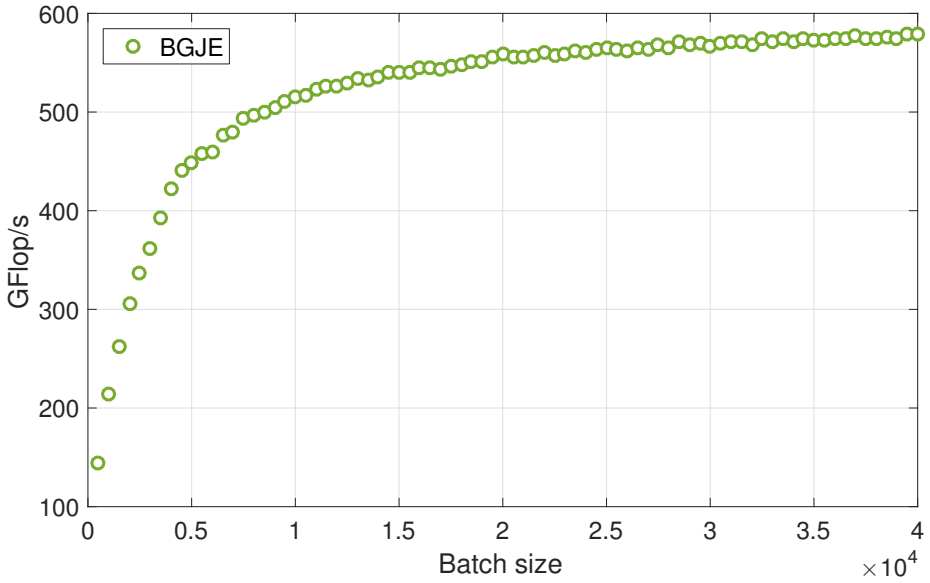
```
% Input  : m x m nonsingular matrix block Di.
% Output : Matrix block Di overwritten by its inverse
p = [1:m];
for k = 1 : m
   % pivoting
    [abs_ipiv, ipiv]      = max(abs(Di(k:m,k)));
    ipiv                  = ipiv+k-1;
    [Di(k,:), Di(ipiv,:)] = swap(Di(ipiv,:), Di(k,:));
    [p(k),   p(ipiv)]     = swap(p(ipiv), p(k));

   % Jordan transformation
    d        =  Di(k,k);
    Di(:,k) =-[Di(1:k-1,k); 0; Di(k+1:m,k)] / d;% SCAL
    Di       =  Di + Di(:,k) * Di(k,:);          % GER
    Di(k,:) = [Di(k,1:k-1), 1, Di(k,k+1:m)] / d;% SCAL
end
% Undo permutations
Di(:,p) = Di;
```

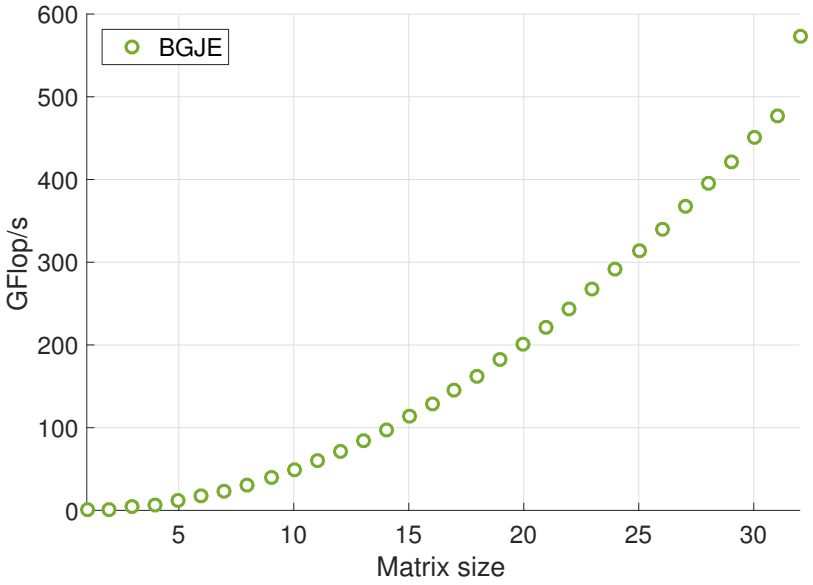
Advantages over LU-based approach:

- No operations on triangular systems
- Implicit pivoting
- Implementation can use static system size

NVIDIA Pascal architecture (P100)

- NVIDIA whitepaper: 56 SMX, 5.3 TF DP, 768 GB/s (**6 : 1**)
- $2n^3$ operations for inversion
- Flexible-size inversion up to size 32x32, one warp per system.

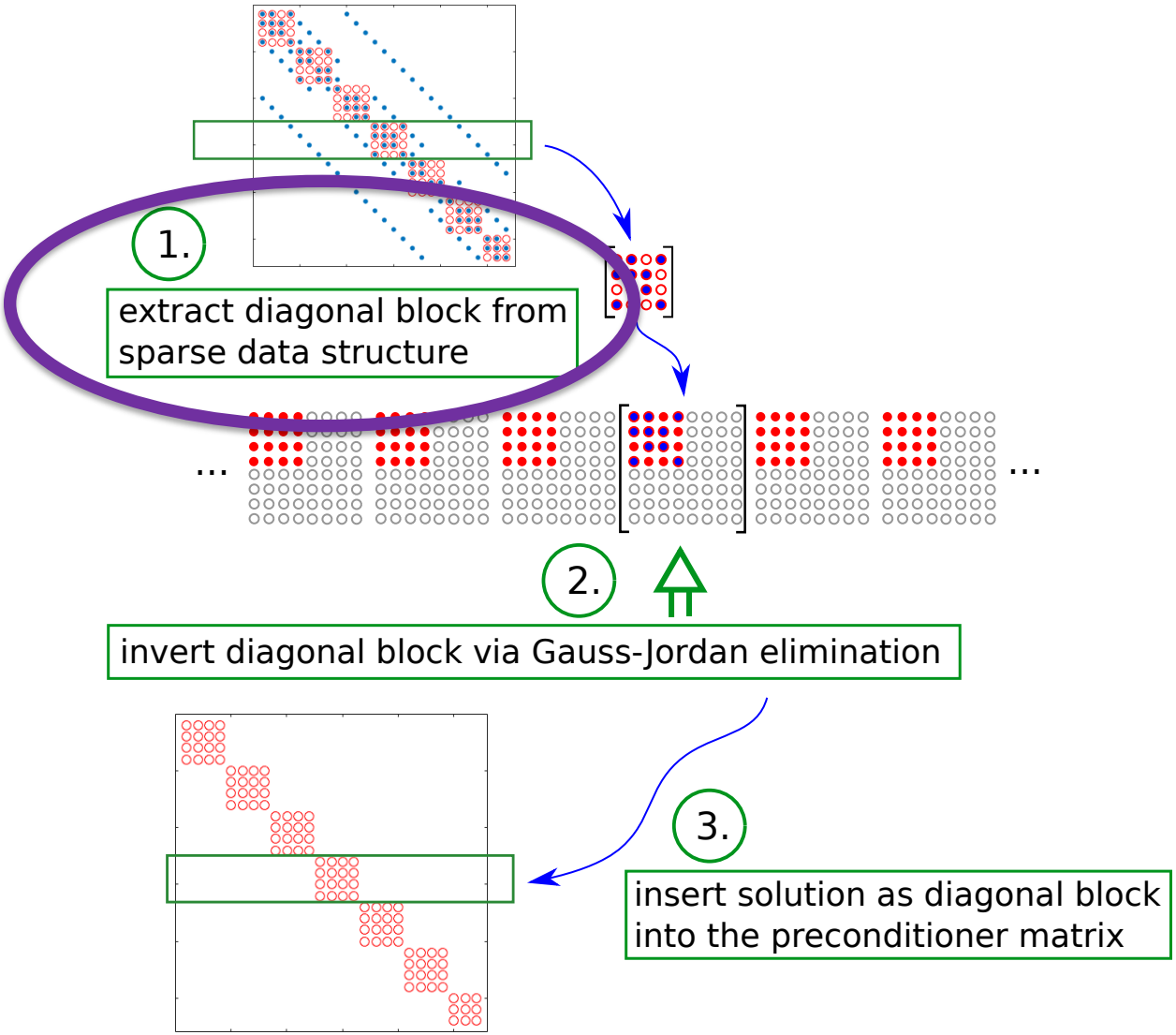




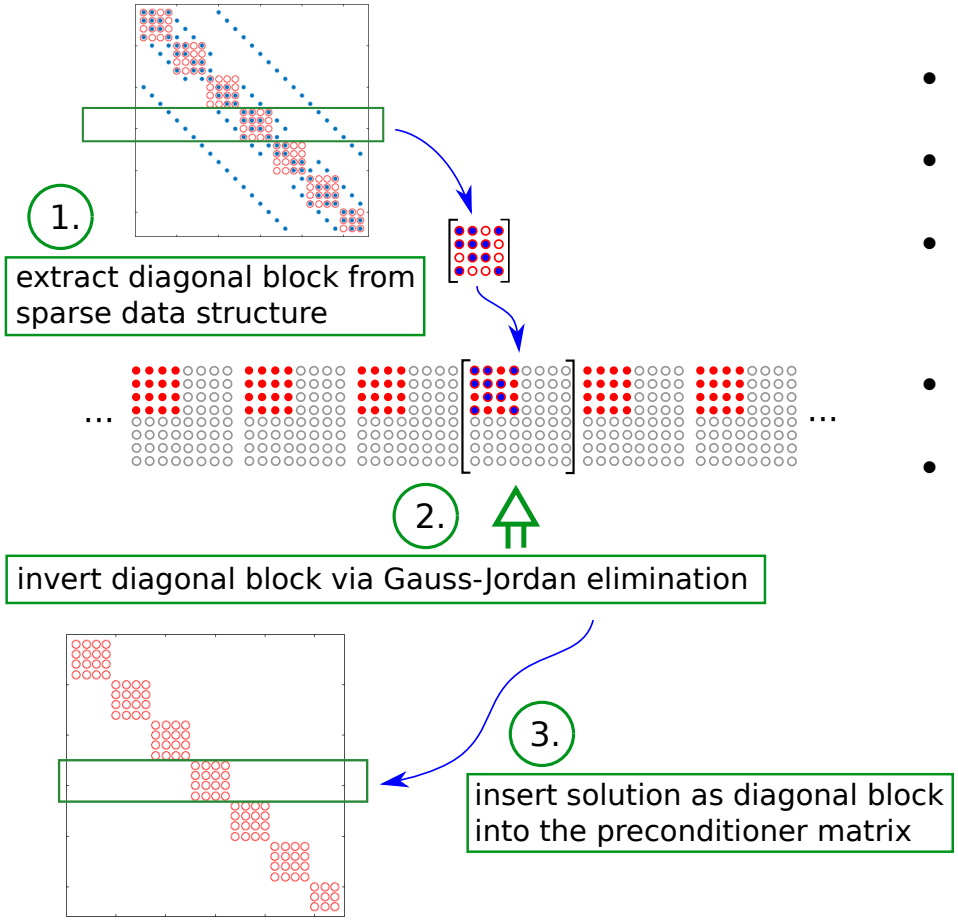
Batched Gauss-Jordan Elimination (BGJE) achieves >10% of DP peak!

Anzt et al. **"Batched Gauss-Jordan Elimination for Block-Jacobi Preconditioner Generation on GPUs".** *PMAM'17, 8th International Workshop on Programming Models and Applications for Multicores and Manycores, pp 1-10, 2017.*

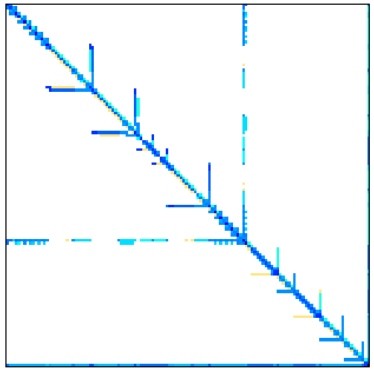
Block-Jacobi preconditioner generation



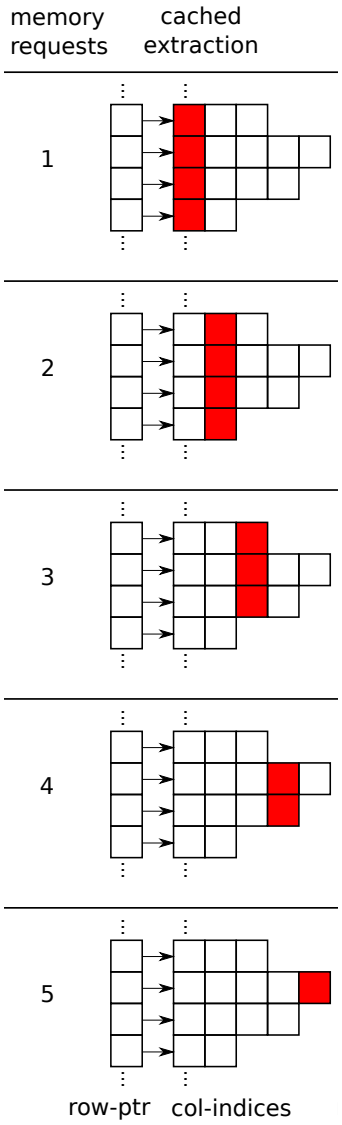
Challenge: diagonal block extraction from sparse data structures



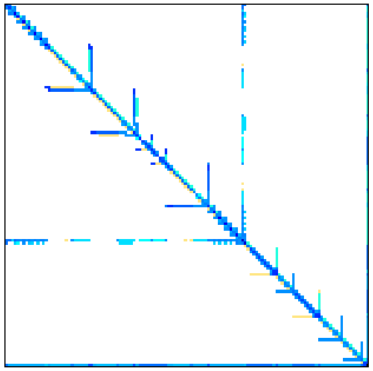
- Matrix in CSR format
- Diagonal block extraction costly
- Assign one thread per row
- Unbalanced nonzero distribution results in load imbalance
- “Dense” rows become bottleneck
- Non-coalescent memory access



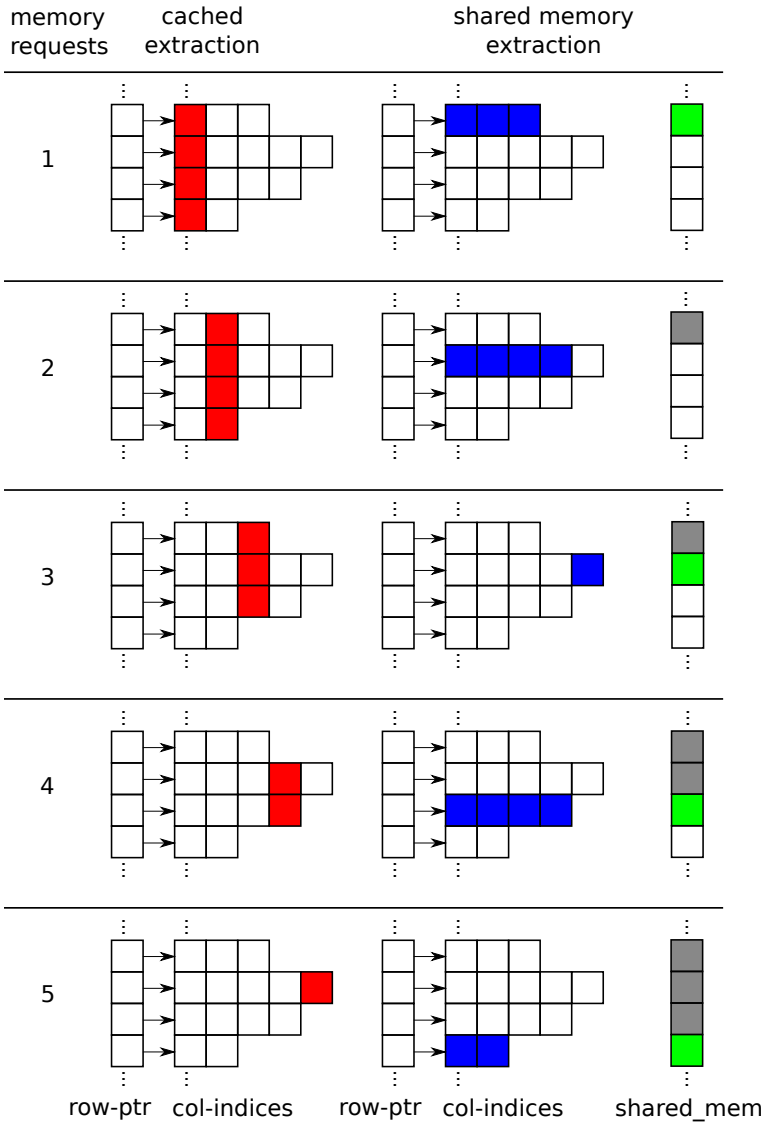
Challenge: diagonal block extraction from sparse data structures



- Matrix in CSR format
- Diagonal block extraction costly
- Assign one thread per row
- Unbalanced nonzero distribution results in load imbalance
- “Dense” rows become bottleneck
- Non-coalescent memory access

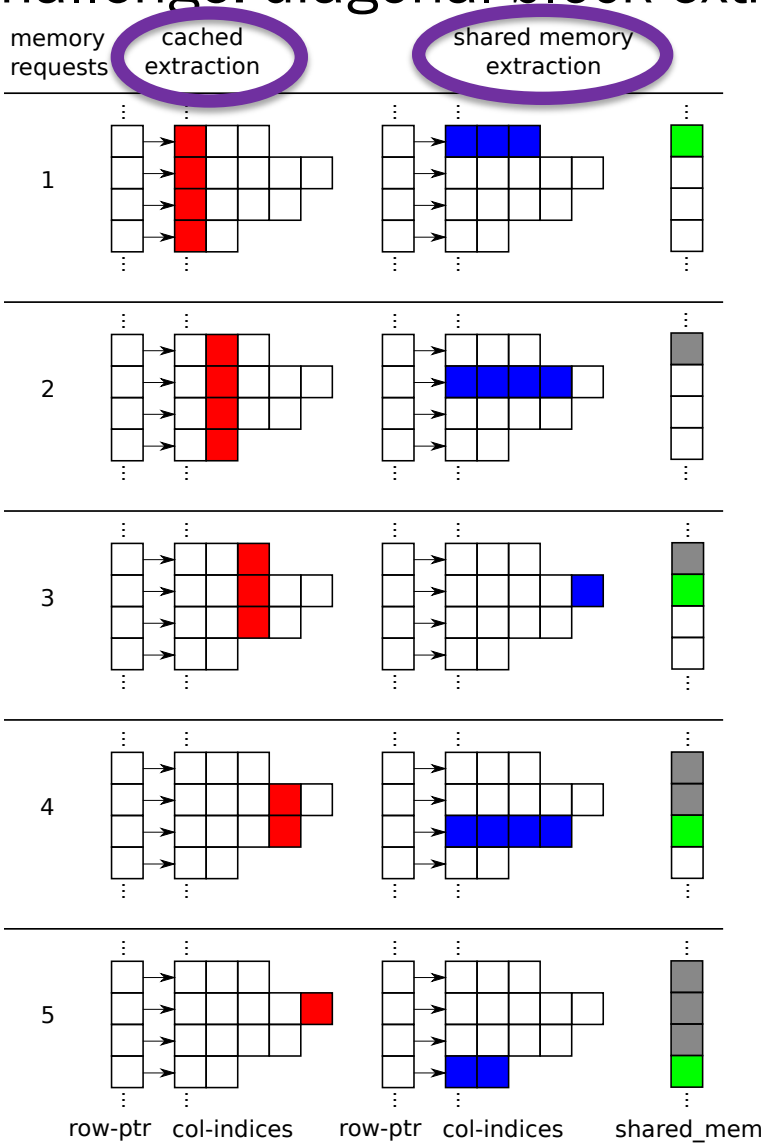


Challenge: diagonal block extraction from sparse data structures



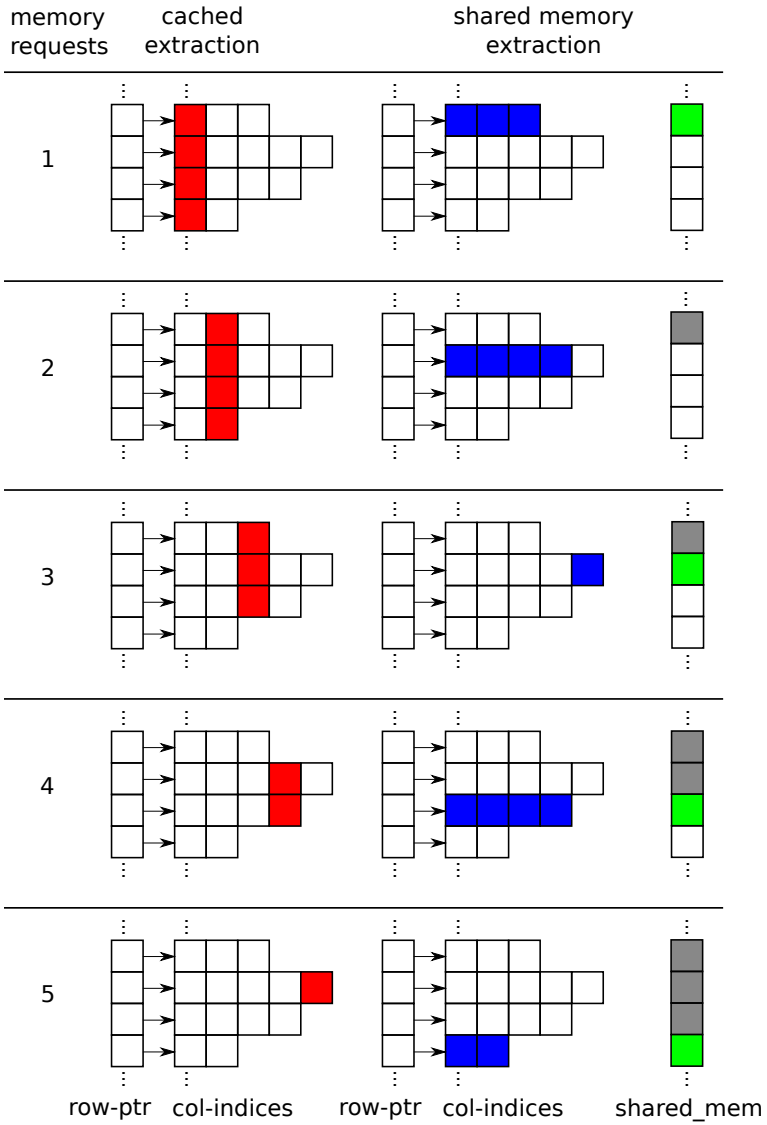
- Matrix in CSR format
- Diagonal block extraction costly
- Assign one thread per row
- Unbalanced nonzero distribution results in load imbalance
- "Dense" rows become bottleneck
- Non-coalescent memory access
- Add intermediate step:
  - Extract diag. blocks to shared mem.
  - Multiple threads for each row
  - Coalescent access to data in CSR
  - Col-major storage

Challenge: diagonal block extraction from sparse data structures



- Matrix in CSR format
- Diagonal block extraction costly
- Assign one thread per row
- Unbalanced nonzero distribution results in load imbalance
- "Dense" rows become bottleneck
- Non-coalescent memory access
- Add intermediate step:
  - Extract diag. blocks to shared mem.
  - Multiple threads for each row
  - Coalescent access to data in CSR
  - Col-major storage

Challenge: diagonal block extraction from sparse data structures



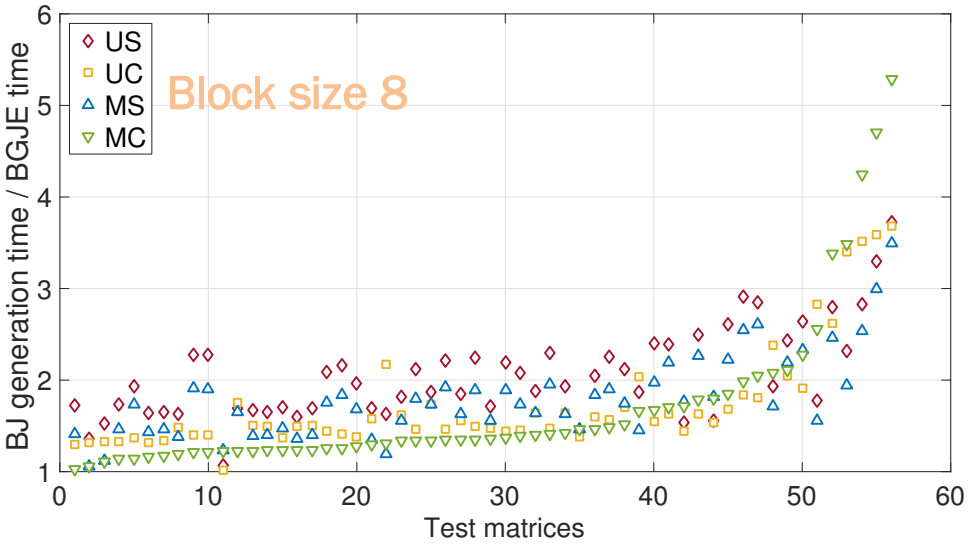
- Matrix in CSR format
- Diagonal block extraction costly
- Assign one thread per row
- Unbalanced nonzero distribution results in load imbalance
- "Dense" rows become bottleneck
- Non-coalescent memory access
- Add intermediate step:
  - Extract diag. blocks to shared mem.
  - Multiple threads for each row
  - Coalescent access to data in CSR
  - Col-major storage
- Merge kernels vs. 3 separate kernels:
  [extraction + inversion + insertion]
  [extraction] + [inversion] + [insertion]

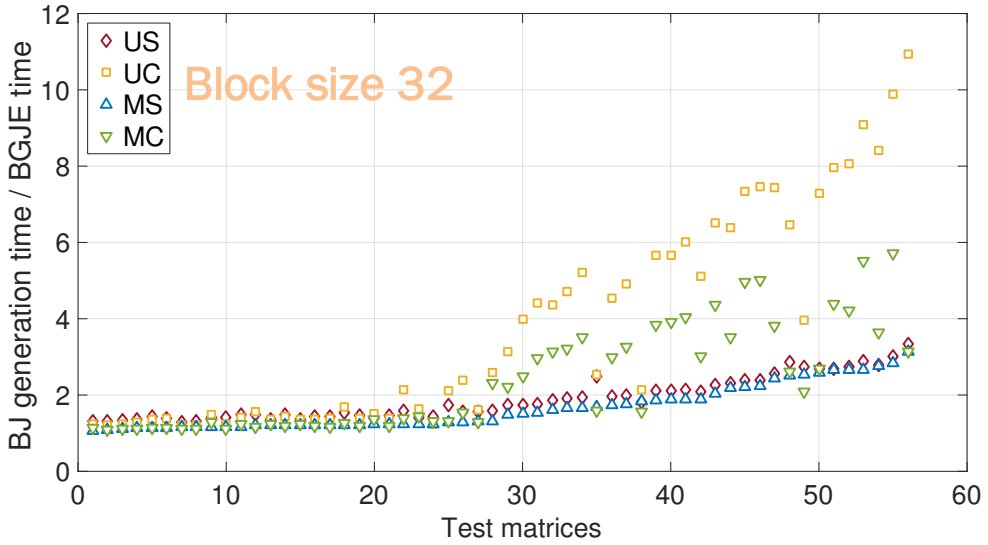
Diagonal block extraction from sparse data structures

US: unmerged + shared extraction
UC: unmerged + cached extraction
MS: merged + shared extraction
MC: merged + cached extraction

Results show:

- Runtime block-Jacobi vs. BGJE
- 56 test matrices (SuiteSparse)
- Data arranged w.r.t. performance of (avg.) best strategy





Diagonal block extraction from sparse data structures

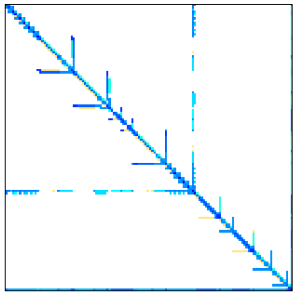
US: unmerged + shared extraction
UC: unmerged + cached extraction
MS: merged + shared extraction
MC: merged + cached extraction

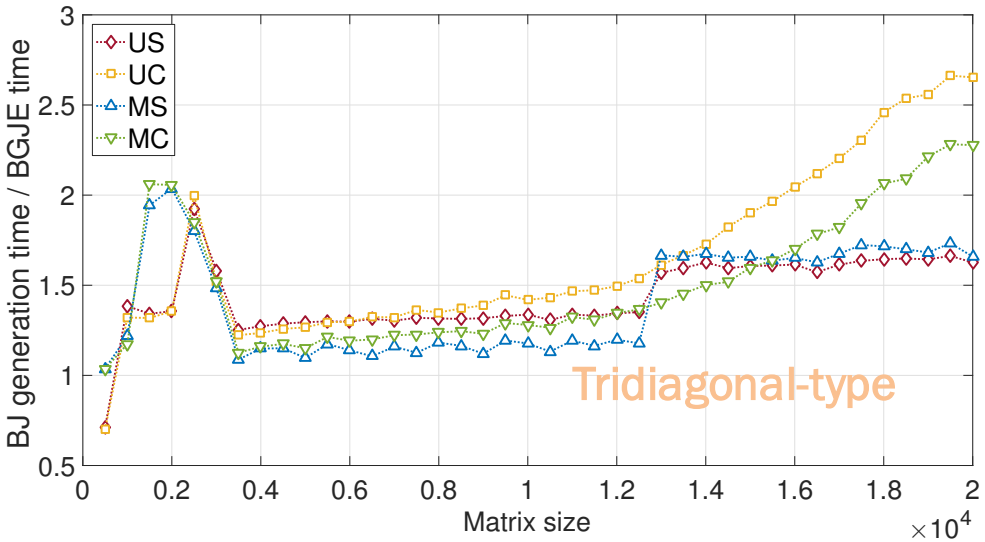
Tridiagonal-type matrix structure:

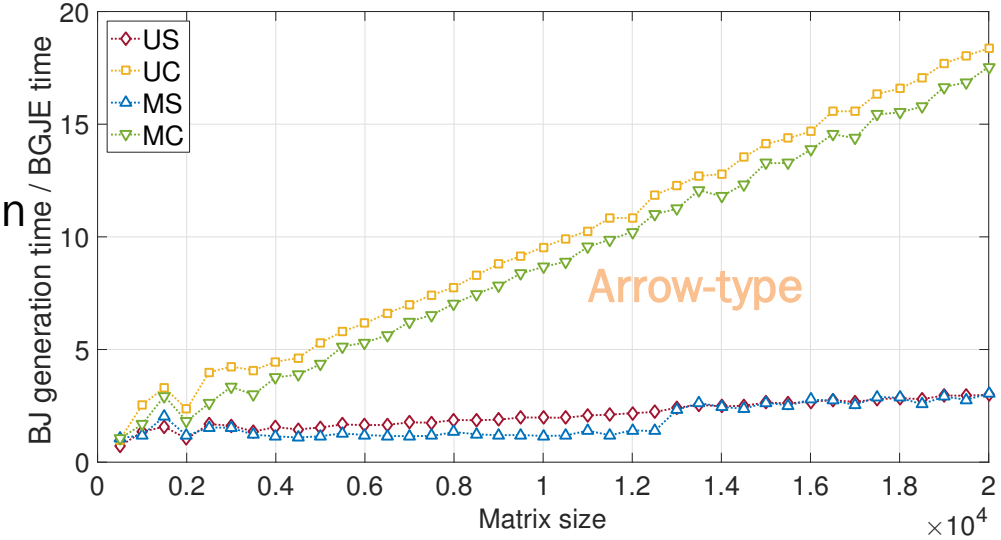
- balanced nnz/row

Arrow-type matrix structure:

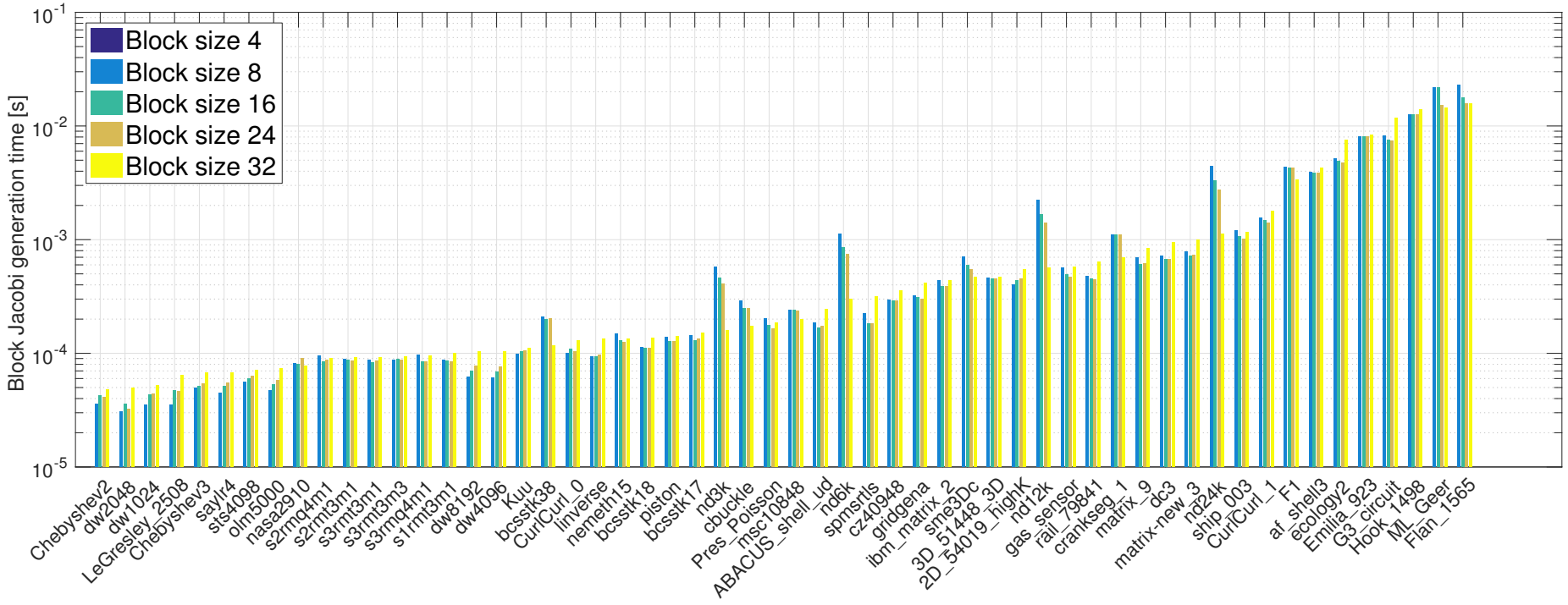
- few rows with many elements
- corresponding threads are bottleneck for cached extraction







Block-Jacobi preconditioner setup cost

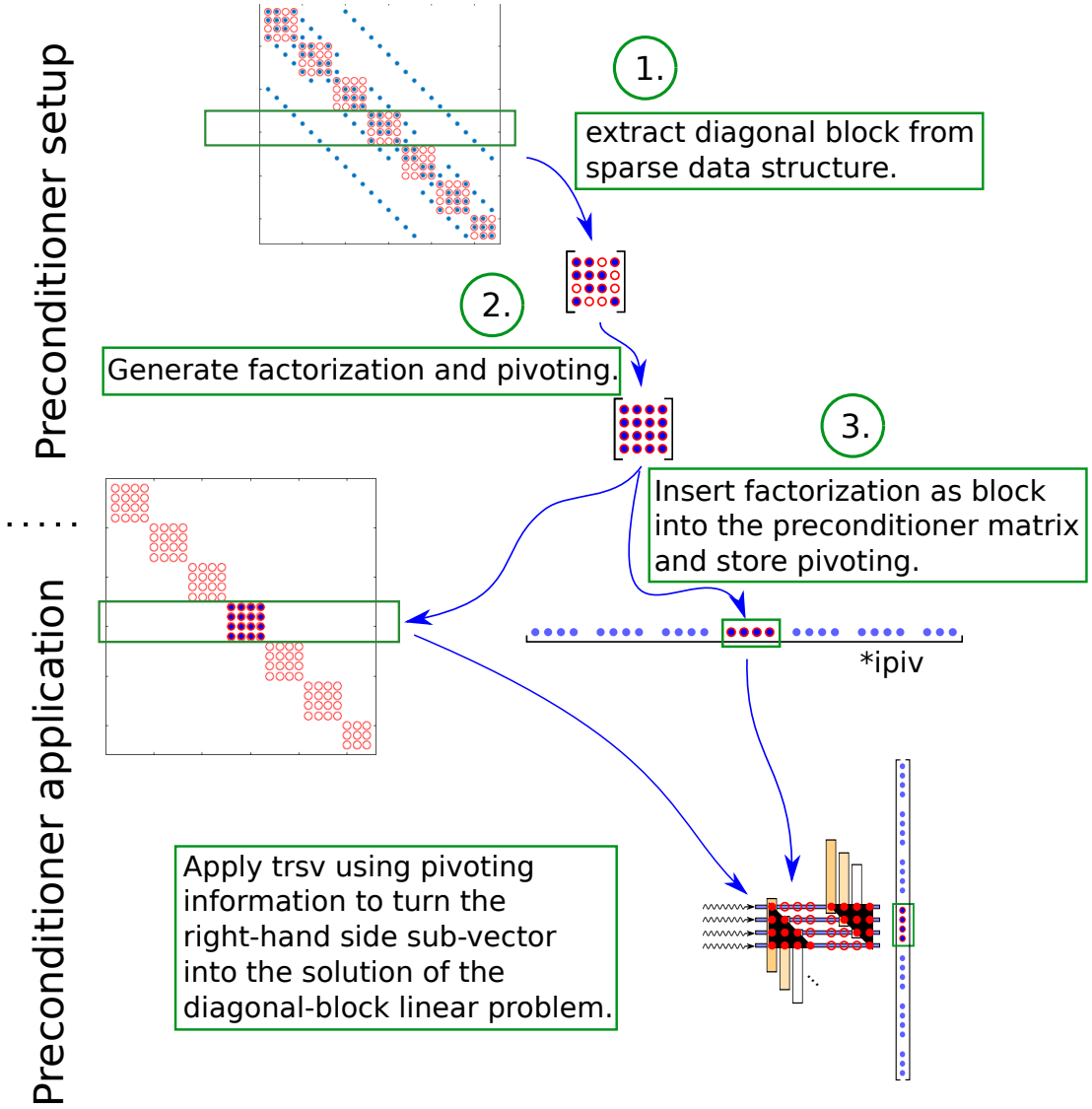


- Preconditioner setup needs up to 0.01 s on P100 GPU.
- Irrelevant for overall Iterative solver execution time.

Block-Jacobi preconditioner

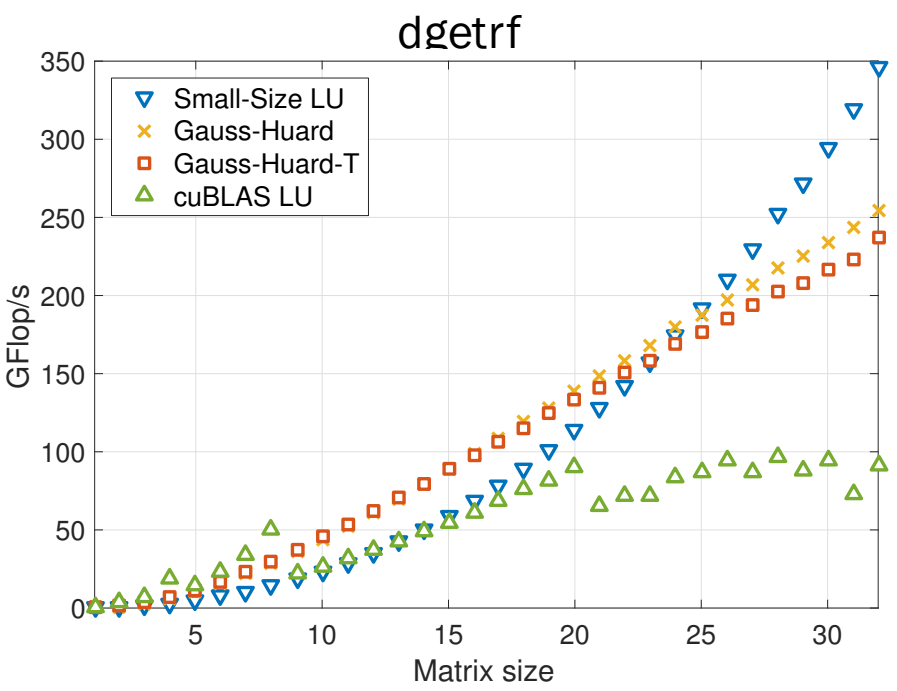
- Can reflect the block-structure of sparse matrices (e.g. Higher-Order FEM)
- Often superior to scalar Jacobi preconditioner
- Light-weight preconditioner (compared to ILU)
- Parallel preconditioner application (batched trsv / SpMV + vector scaling)
  - Inversion of diag. blocks in prec. setup + SpMV in prec. application
  - Factorization of diag. blocks in prec. setup + trsv in prec. application

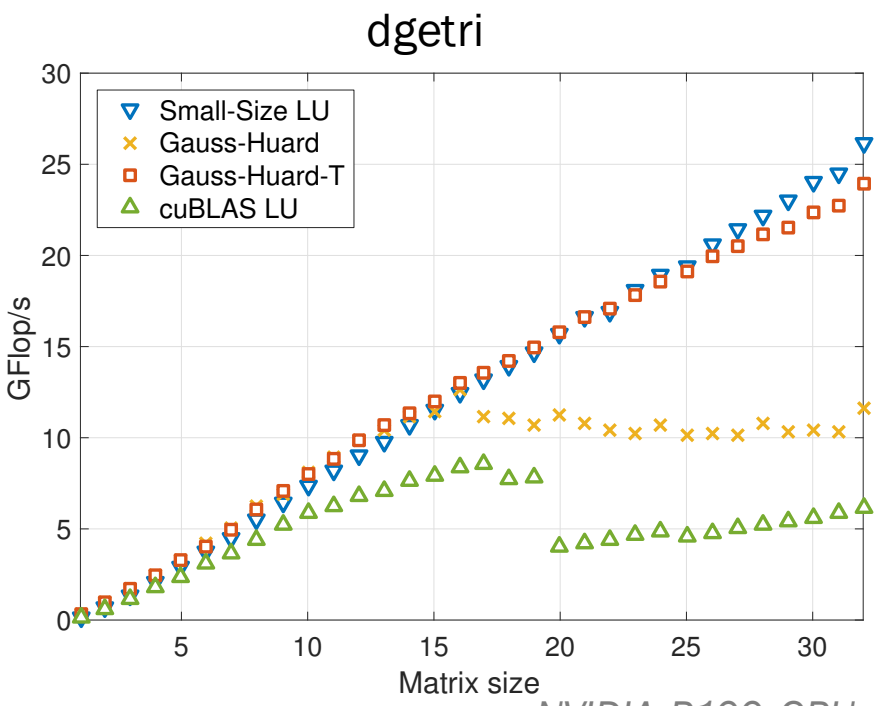
Factorization-based block-Jacobi



Factorization-based block-Jacobi

- Batched routines: Small-Size LU, Gauss-Huard, Gauss-Huard-T.
- Gauss-Huard-T writes the factors in "access-friendly" fashion.
- Flexible size up to limit of 32 x 32.
- Use one warp per system, factorization in registers.



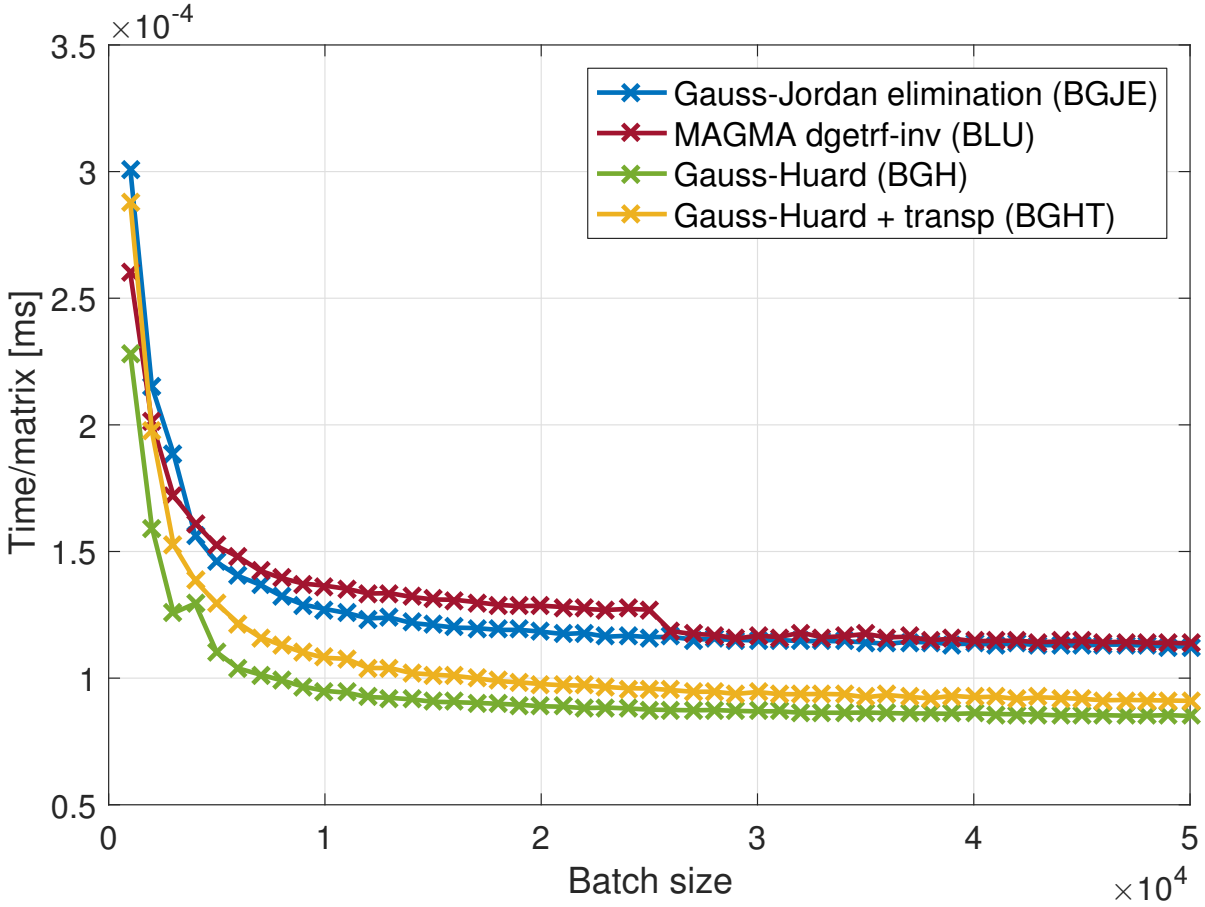


NVIDIA P100 GPU

ICL

THE UNIVERSITY OF TENNESSEE KNOXVILLE

NVIDIA Pascal architecture (P100)

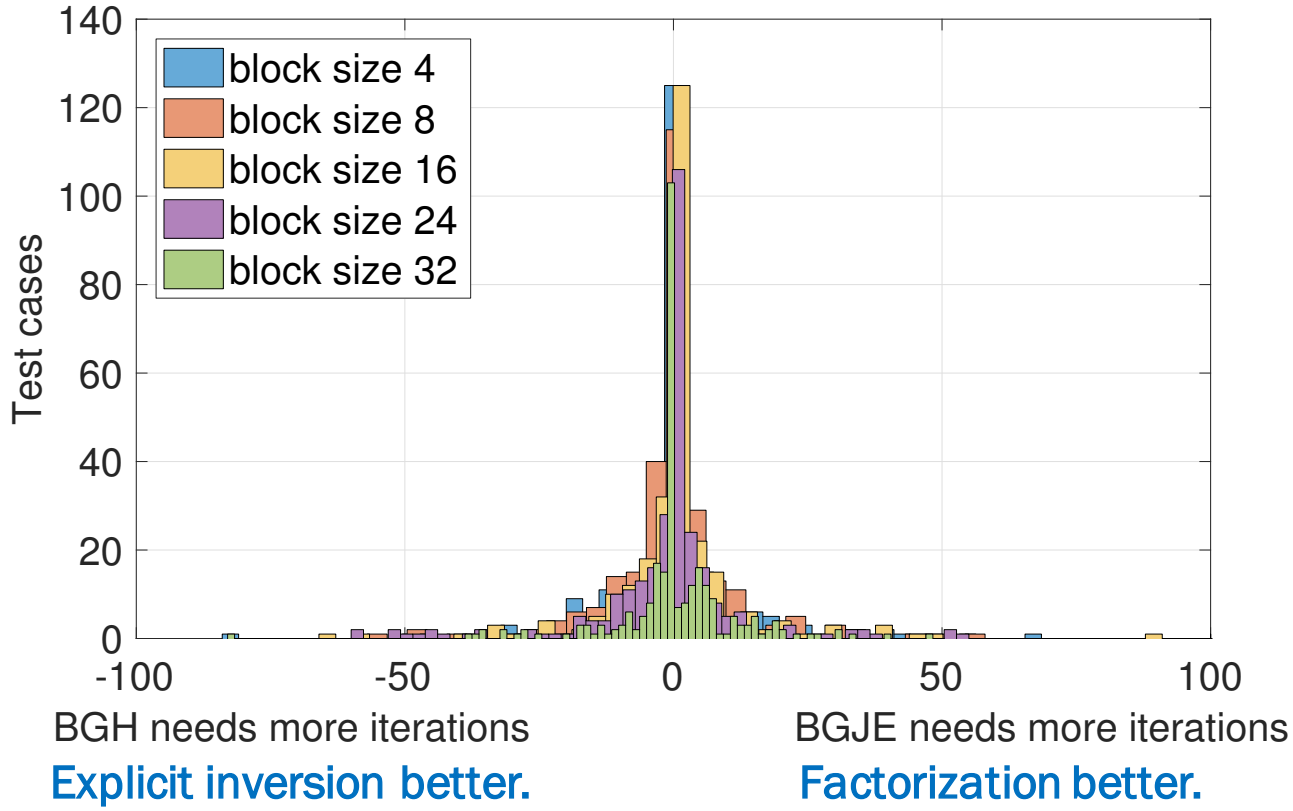


- BGJE and BLU invert diagonal blocks vs. BGH / BGHT factorizing only.
- BGHT writes factors in non-coalescent fashion for faster preconditioner application.

NVIDIA P100 GPU

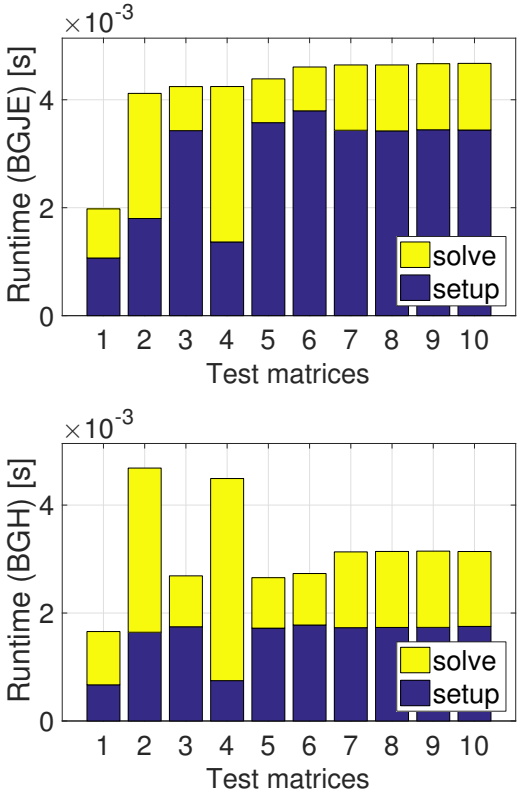
Inversion-based block-Jacobi vs. factorization-based block-Jacobi

Convergence of BiCGSTAB with block-Jacobi preconditioner for 300 matrices

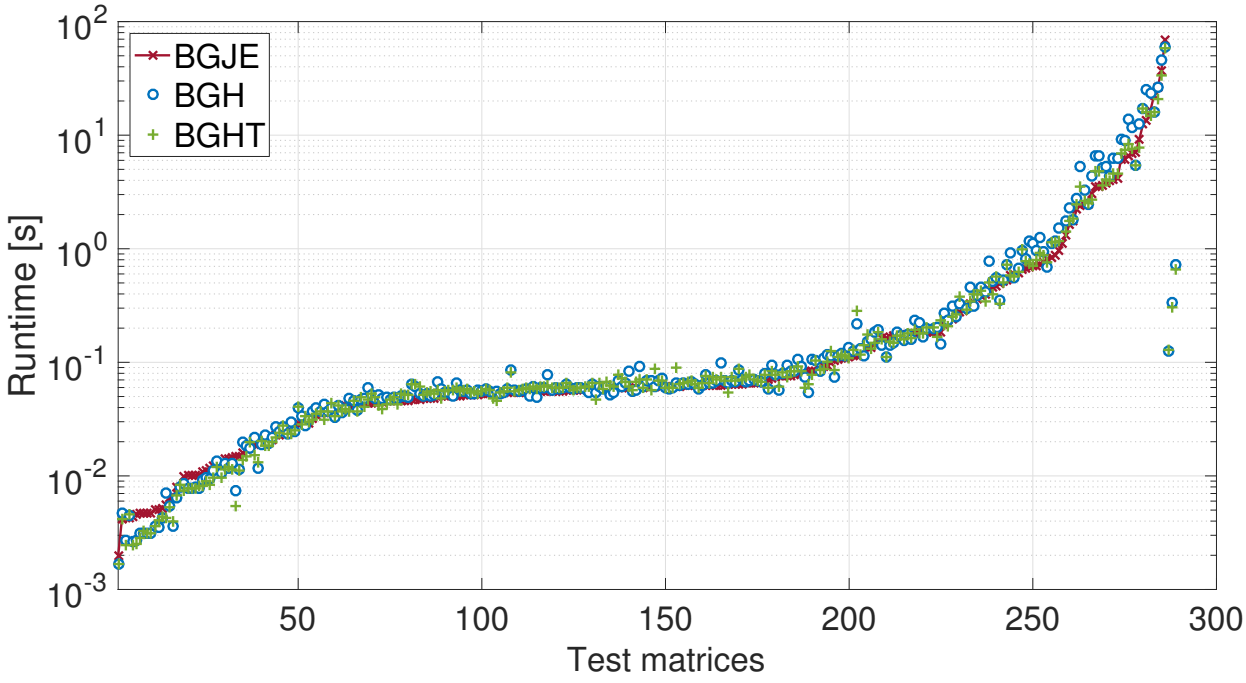


NVIDIA P100 GPU

Block-Jacobi preconditioning: Performance goal



Execution time of BiCGSTAB with Jacobi(24) preconditioner



- Almost no difference in total execution time.
- BGH better if setup is significant to iterative solver time.
- BGJE better for long solver runs.

NVIDIA P100 GPU

MAGMA

EXASCALE COMPUTING 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/

Pitch from the sparse community:

- Small size (<30) "somewhat" flexible
- Add inversion routine to Batched-Blas
- Go for Gauss-Jordan Elimination instead of LU

This research is based on a cooperation between Hartwig Anzt, Jack Dongarra (University of Tennessee), Goran Flegar and Enrique Quintana-Orti (University of Jaume I), and partly funded by the Department of Energy.

THE UNIVERSITY OF TENNESSEE KNOXVILLE

UNIVERSITAT JAUME·I

http://www.icl.utk.edu/~hanzt/talks/batched_bjac.pdf