Technische Universität Braunschweig

Institute for Computational Mathematics

High Performance Block Incomplete LU Factorization

Matthias Bollhöfer (TU Braunschweig)
Sparse Days @ Cerfacs 2017, Toulouse, September 8, 2017

Outline

- Motivation
- Block ILUs
- Setting Up and Improving the Block Structures
- Numerical Results
- Numerical Results — Symmetric Indefinite Case
- Conclusions

Outline

- **Motivation**
- Block ILUs
- Setting Up and Improving the Block Structures
- Numerical Results
- Numerical Results — Symmetric Indefinite Case
- Conclusions

Motivation — Fast Approximate *LU* factorization

- Objectives:
  - solving large–scale sparse linear systems

$$Ax \overset{!}{=} b$$

Motivation — Fast Approximate *LU* factorization

- Objectives:
  - solving large–scale sparse linear systems

$$Ax \stackrel{!}{=} b$$

  - Computing (e.g.) the diagonal part of the matrix inverse of large–scale sparse matrices

$$D \approx \mathcal{D}(A^{-1})$$

Motivation — Fast Approximate *LU* factorization

- Objectives:
  - solving large–scale sparse linear systems

$$Ax \stackrel{!}{=} b$$

  - Computing (e.g.) the diagonal part of the matrix inverse of large–scale sparse matrices

$$D \approx \mathcal{D}(A^{-1})$$

- Common numerical approach: approximate Gaussian elimination (incomplete *LU* decomposition — ILU) by dropping entries of small size

$$A \approx LDU$$

where L, U^T unit lower triangular, D diagonal

Motivation — Fast Approximate *LU* factorization

- Objectives:
  - solving large–scale sparse linear systems

$$Ax \stackrel{!}{=} b$$

  - Computing (e.g.) the diagonal part of the matrix inverse of large–scale sparse matrices

$$D \approx \mathcal{D}(A^{-1})$$

- Common numerical approach: approximate Gaussian elimination (incomplete *LU* decomposition — ILU) by dropping entries of small size

$$A \approx LDU$$

  where L, U^T unit lower triangular, D diagonal
- $Ax = b$: Iterative solution (preconditioned Krylov subspace method, e.g. PCG, GMRES,...)

Motivation — Fast Approximate *LU* factorization

- Objectives:
  - solving large–scale sparse linear systems

$$Ax \stackrel{!}{=} b$$

  - Computing (e.g.) the diagonal part of the matrix inverse of large–scale sparse matrices

$$D \approx \mathcal{D}(A^{-1})$$

- Common numerical approach: approximate Gaussian elimination (incomplete *LU* decomposition — ILU) by dropping entries of small size

$$A \approx LDU$$

  where L, U^T unit lower triangular, D diagonal
- $Ax = b$: Iterative solution (preconditioned Krylov subspace method, e.g. PCG, GMRES,...)
- $\mathcal{D}(A^{-1})$: selectively invert LDU, supplemented with low-rank deflation using singular vectors

Motivation — Fast (approximate) *LU* Factorization

Modern sparse direct solvers (e.g. PARDISO, MA48, UMFPACK, MUMPS, SuperLU, ...) are hard to beat, they

- (may) improve diagonal dominance and reduce pivoting in advance
- use fill-reducing orderings
- apply combinatorial techniques to detect dense blocks (supernodes/multifrontal) [Duff,Erisman,Reid'17]
- make heavily use of dense matrix kernels (level–3 BLAS, LAPACK)

Motivation — Fast (approximate) *LU* Factorization

Modern sparse direct solvers (e.g. PARDISO, MA48, UMFPACK, MUMPS, SuperLU, ...) are hard to beat, they

- (may) improve diagonal dominance and reduce pivoting in advance
- use fill-reducing orderings
- apply combinatorial techniques to detect dense blocks (supernodes/multifrontal) [Duff,Erisman,Reid'17]
- make heavily use of dense matrix kernels (level–3 BLAS, LAPACK)

To compete with direct methods, ILUs should

- produce drastically less fill–in (ILUs in general with modest drop tolerance)
- compute robust factorizations (e.g. ILUPACK)
- make use of dense matrix kernels, if possible (that is what this talk is about)

Outline

- Motivation
- **Block ILUs**
- Setting Up and Improving the Block Structures
- Numerical Results
- Numerical Results — Symmetric Indefinite Case
- Conclusions

A Block Incomplete *LU* Factorization method

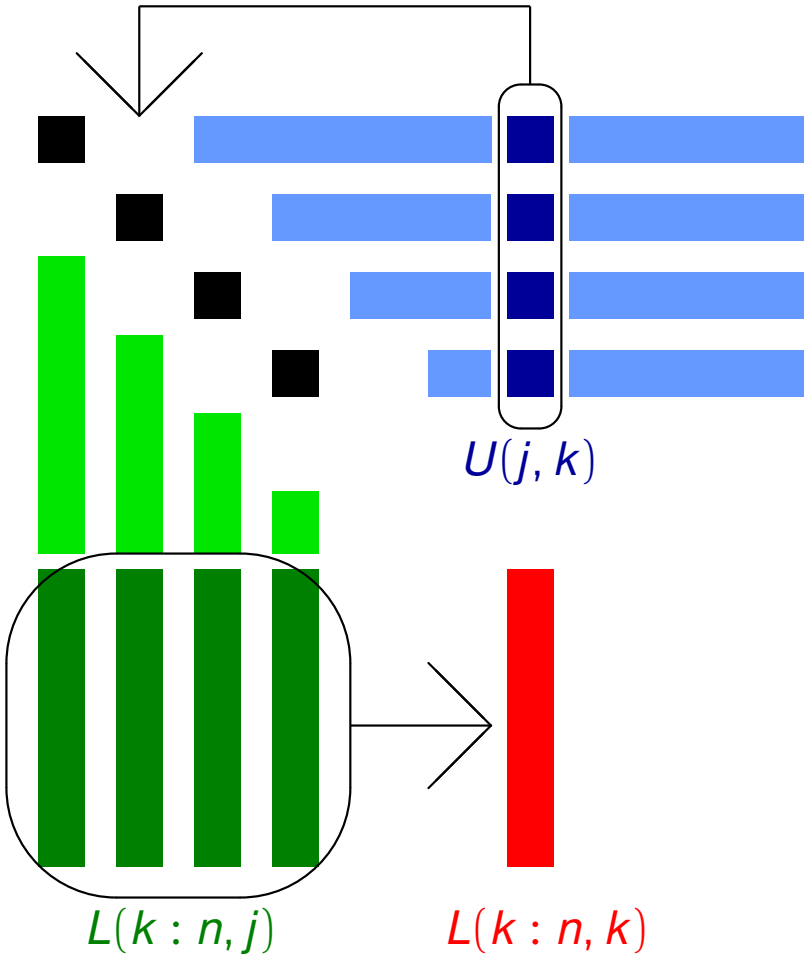
A generic meta block ILU approach

Suppose that some preprocessing has been done in advance:

- improve diagonal dominance by scaling and permuting the system matrix A
- reduce fill–in by using a fill–reducing method
- suppose that the (block) ILU will not break down
- suppose that an initial block structure is provided

$\longrightarrow$ (block) ILU with no worries

Basics: the Crout ILU



Basics: the Crout ILU for Sparse Matrices

Suppose that A is sparse.

- initially, $L(k:n,k)$ is sparse
- downdating $L(k:n,k)$ can be implemented efficiently:
  - use a linked list to quickly discover those j such that $U(j,k) \neq 0$ are required
  - `AXPY`-like operation in sparse mode
- dropping and scaling is cheap
- hopefully, $L(k:n,k)$ remains sparse after all
- analogous arguments apply to U

The Crout Block ILU

- replace scalars by blocks in the Crout ILU
  $\longrightarrow$ analogous algorithm!

The Crout Block ILU

- replace scalars by blocks in the Crout ILU
  $\longrightarrow$ analogous algorithm!
- diagonal blocks stored as dense matrices

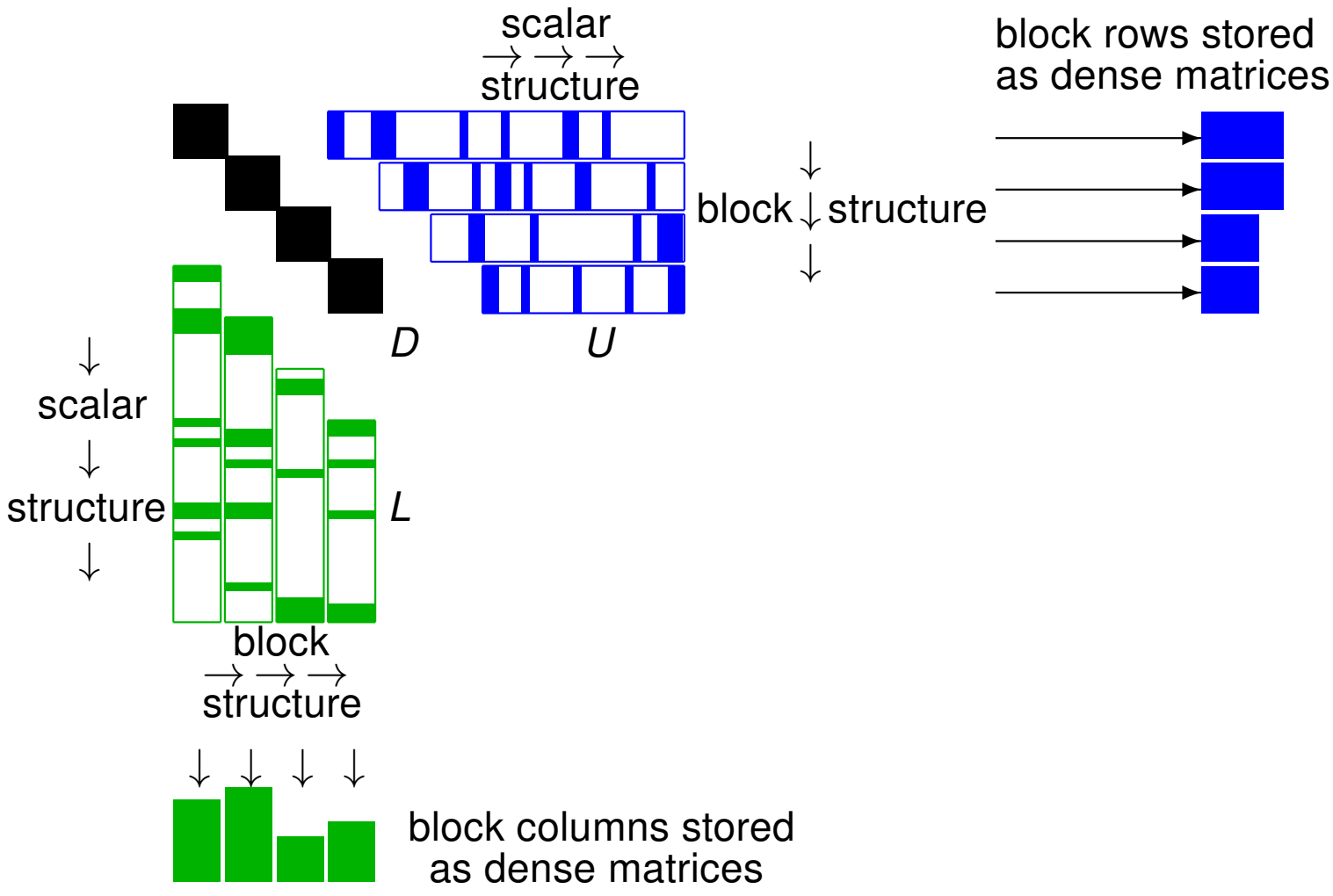
The Crout Block ILU

- replace scalars by blocks in the Crout ILU
  $\longrightarrow$ analogous algorithm!
- diagonal blocks stored as dense matrices
- off-diagonal blocks use hybrid structure

The Crout Block ILU

- replace scalars by blocks in the Crout ILU
  $\longrightarrow$ analogous algorithm!
- diagonal blocks stored as dense matrices
- off-diagonal blocks use hybrid structure
  - L: $\rightarrow$ block columns, $\downarrow$ scalar rows
  - U: $\downarrow$ block rows, $\rightarrow$ scalar columns

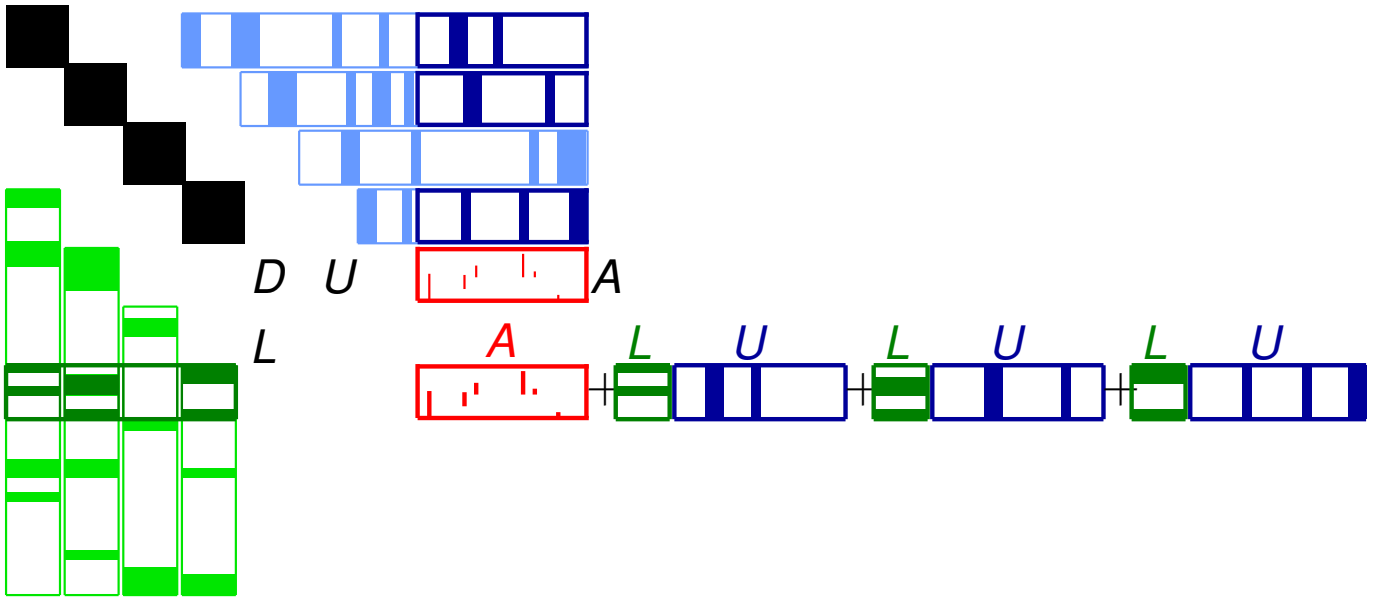
The Crout Block ILU



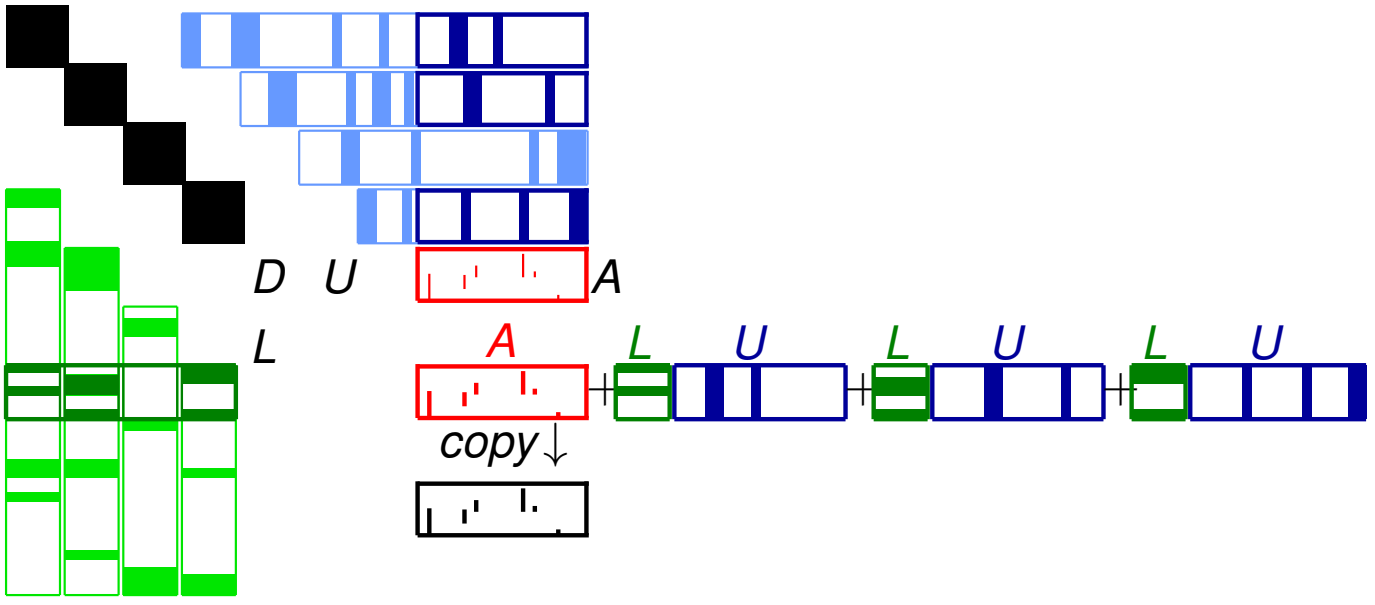
The Crout Block ILU

- dropping: $\|L(i,k) \cdot L(k,k)^{-1}\| \leqslant \tau$
  $\|U(k,k)^{-1} \cdot U(k,i)\| \leqslant \tau$
- downdating: $L(k:n,k) \leftarrow L(k:n,k) - L(k:n,j) \cdot U(j,k)$
  1. `gather` from $L(k:n,k)$
  2. use dense matrix kernel (level–3 BLAS `GEMM`)
  3. `scatter` back to $L(k:n,k)$

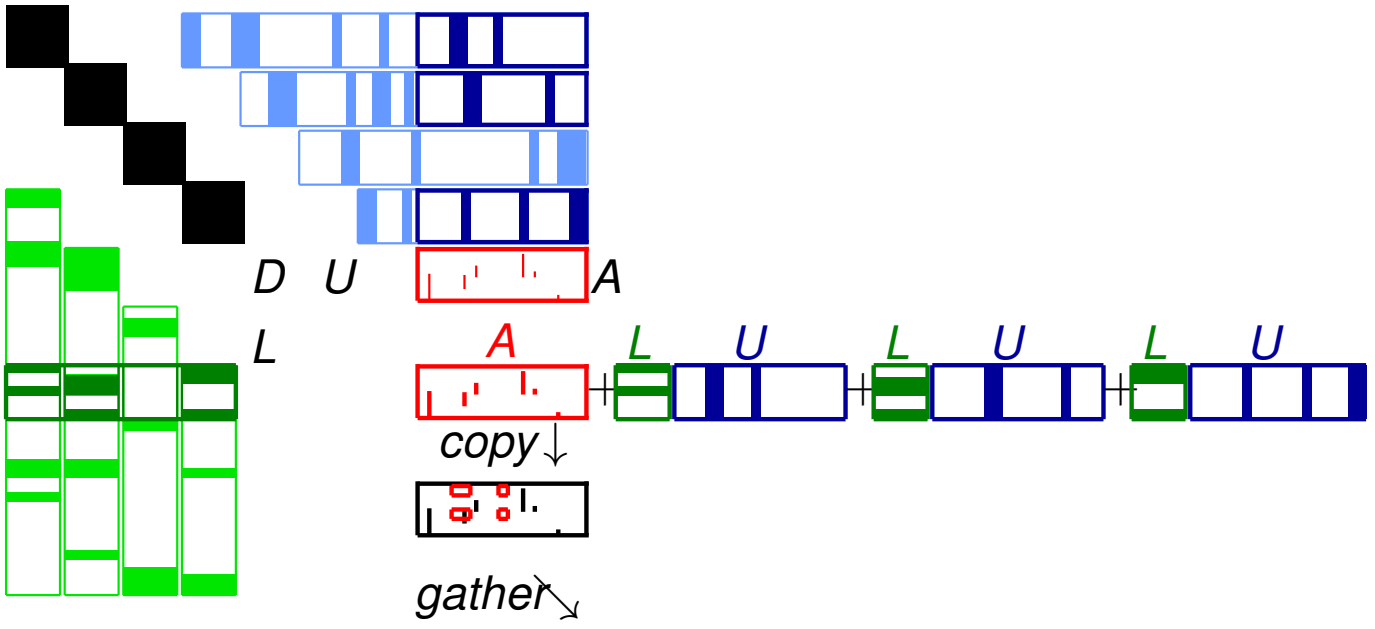
The Crout Block ILU



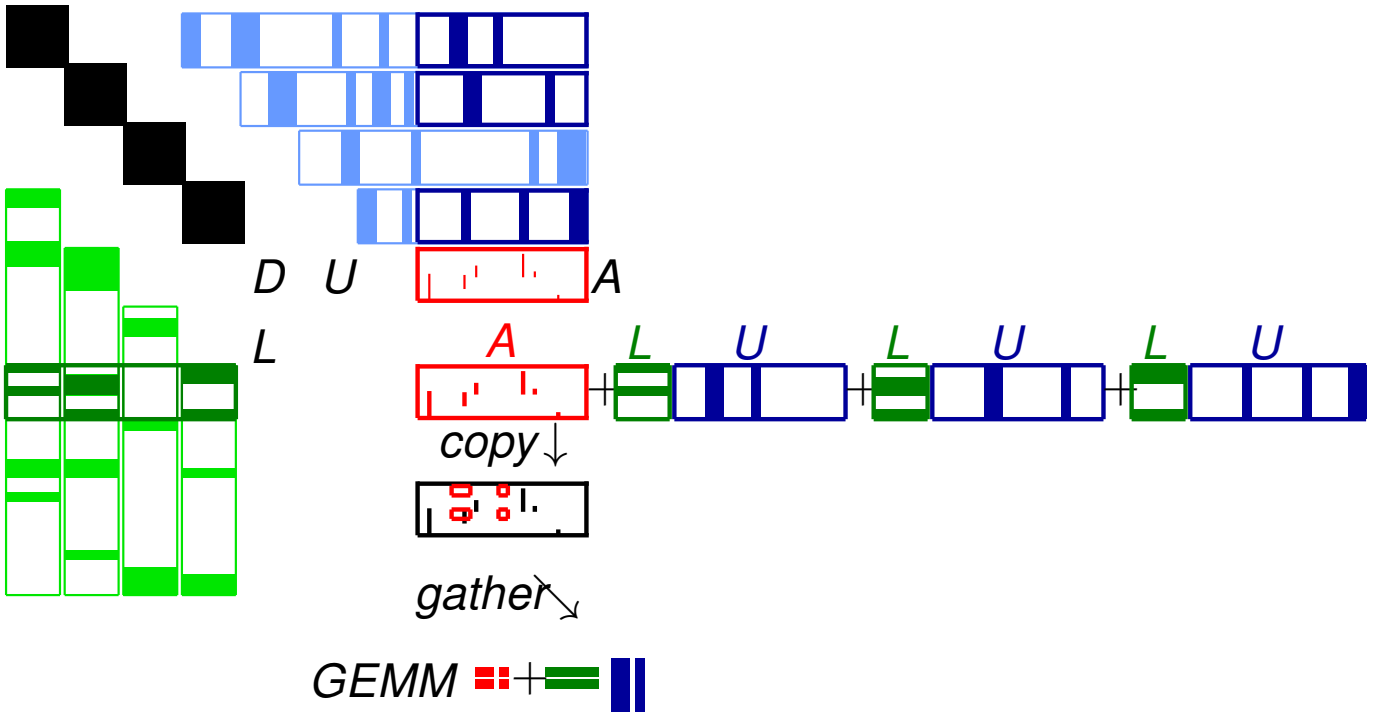
The Crout Block ILU



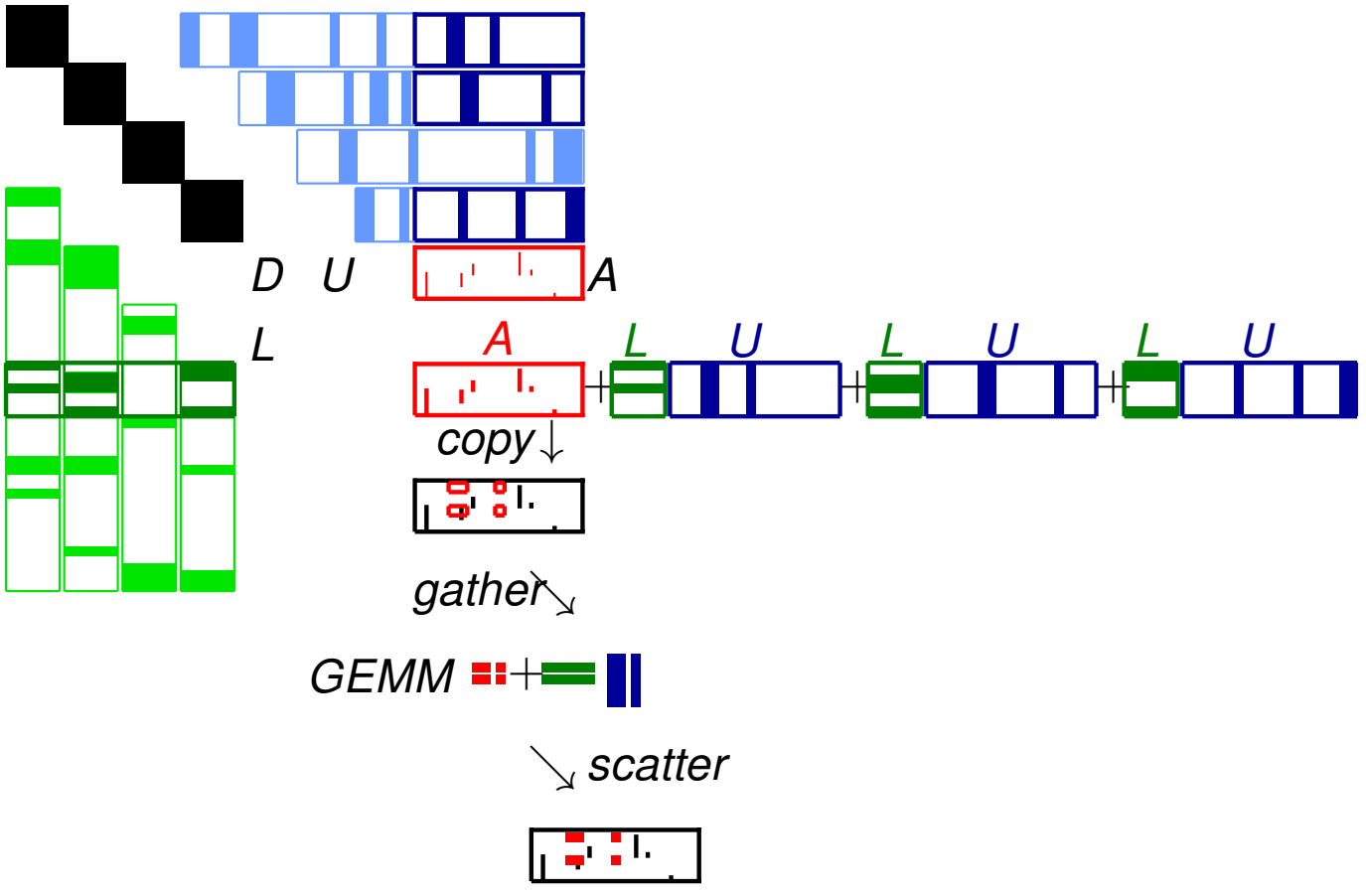
The Crout Block ILU



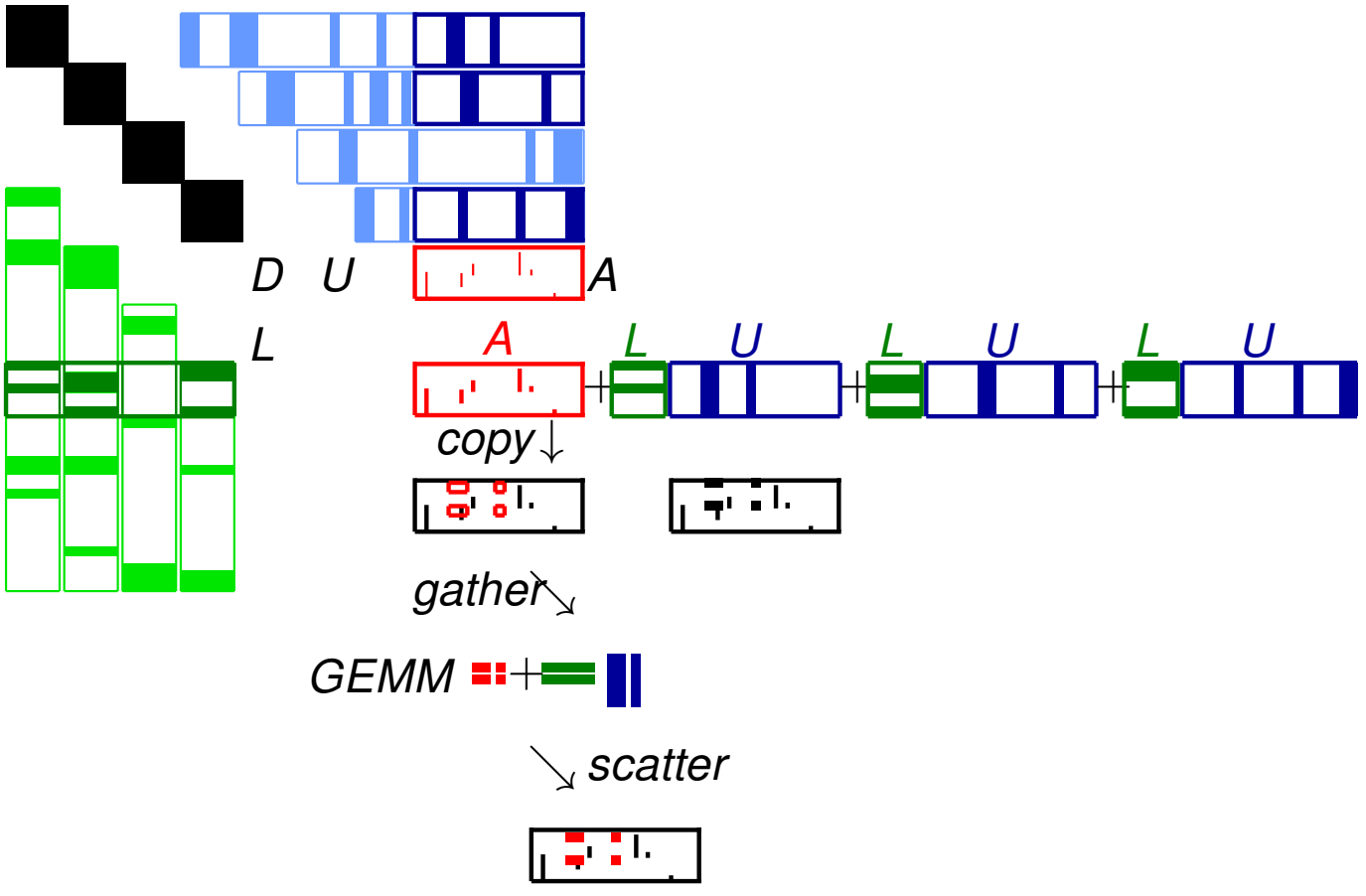
The Crout Block ILU



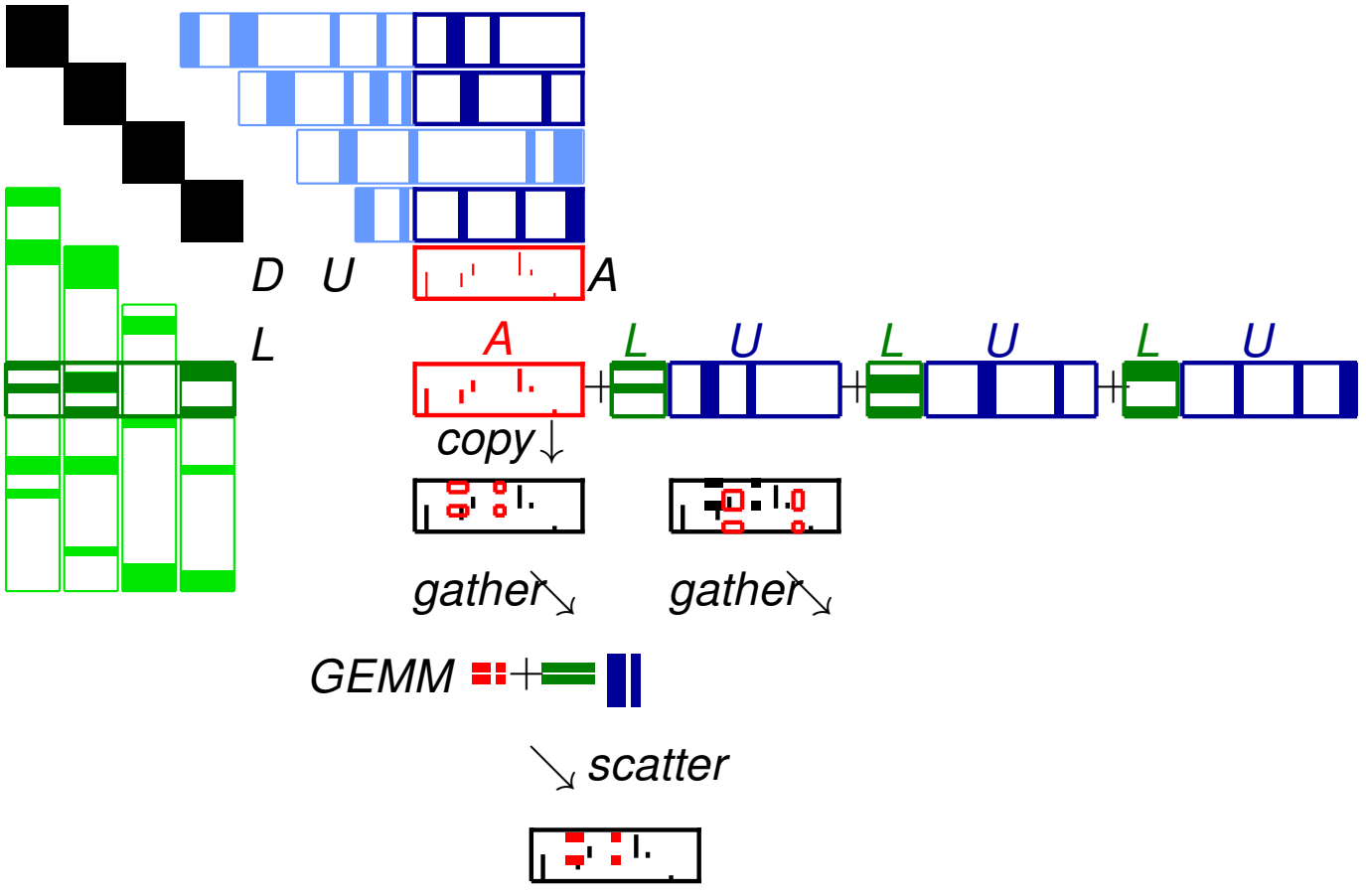
The Crout Block ILU



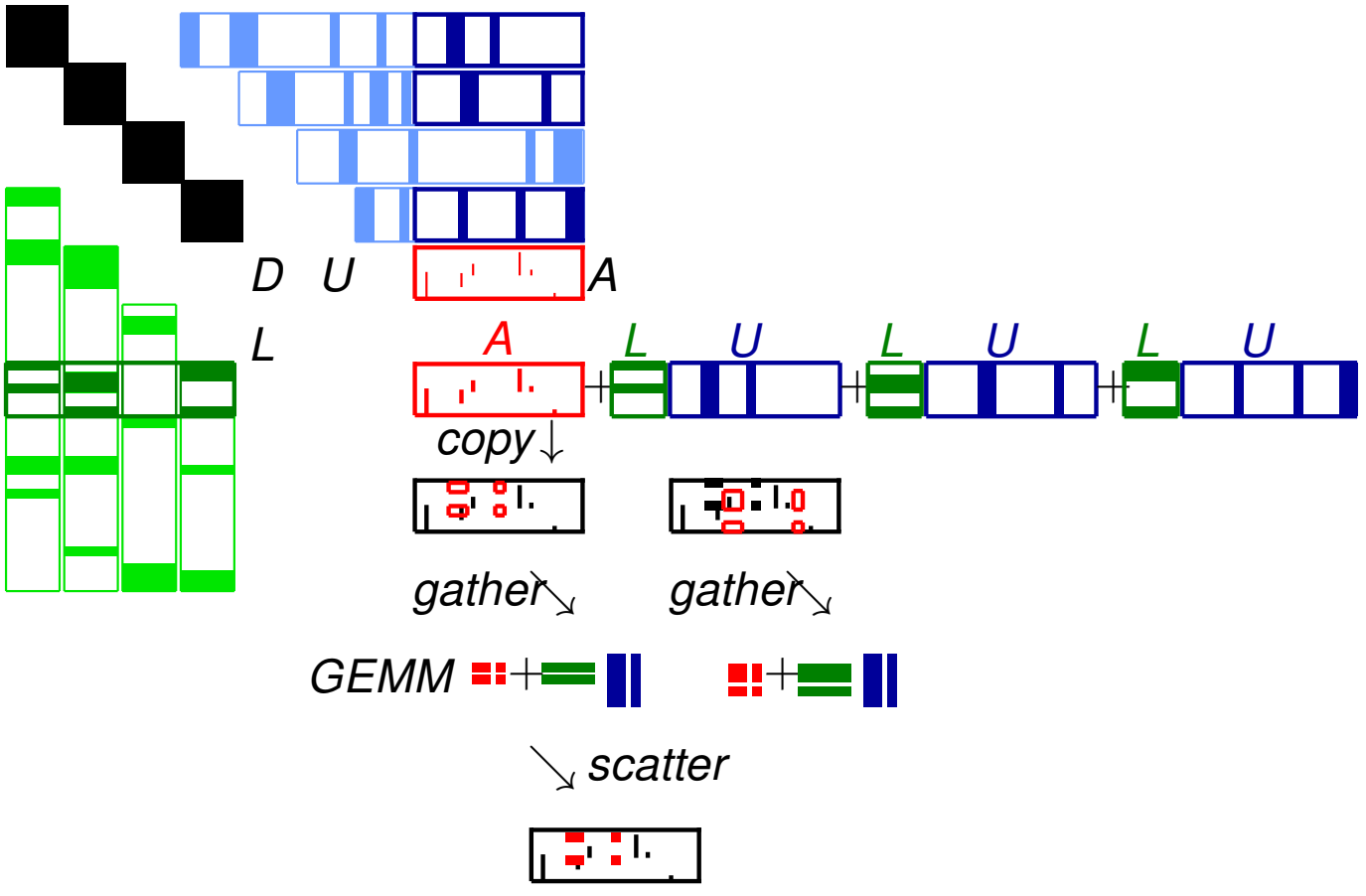
The Crout Block ILU



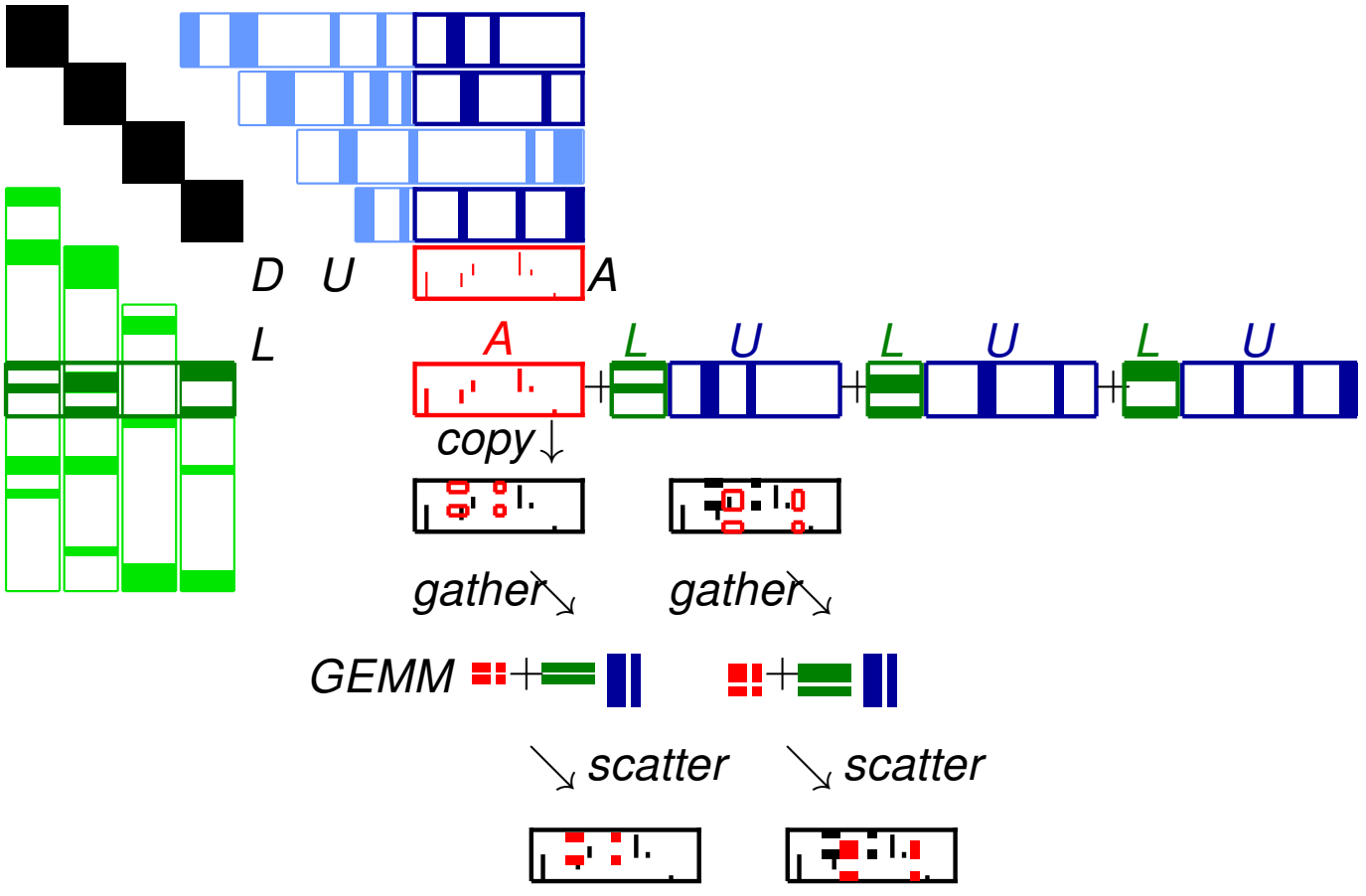
The Crout Block ILU



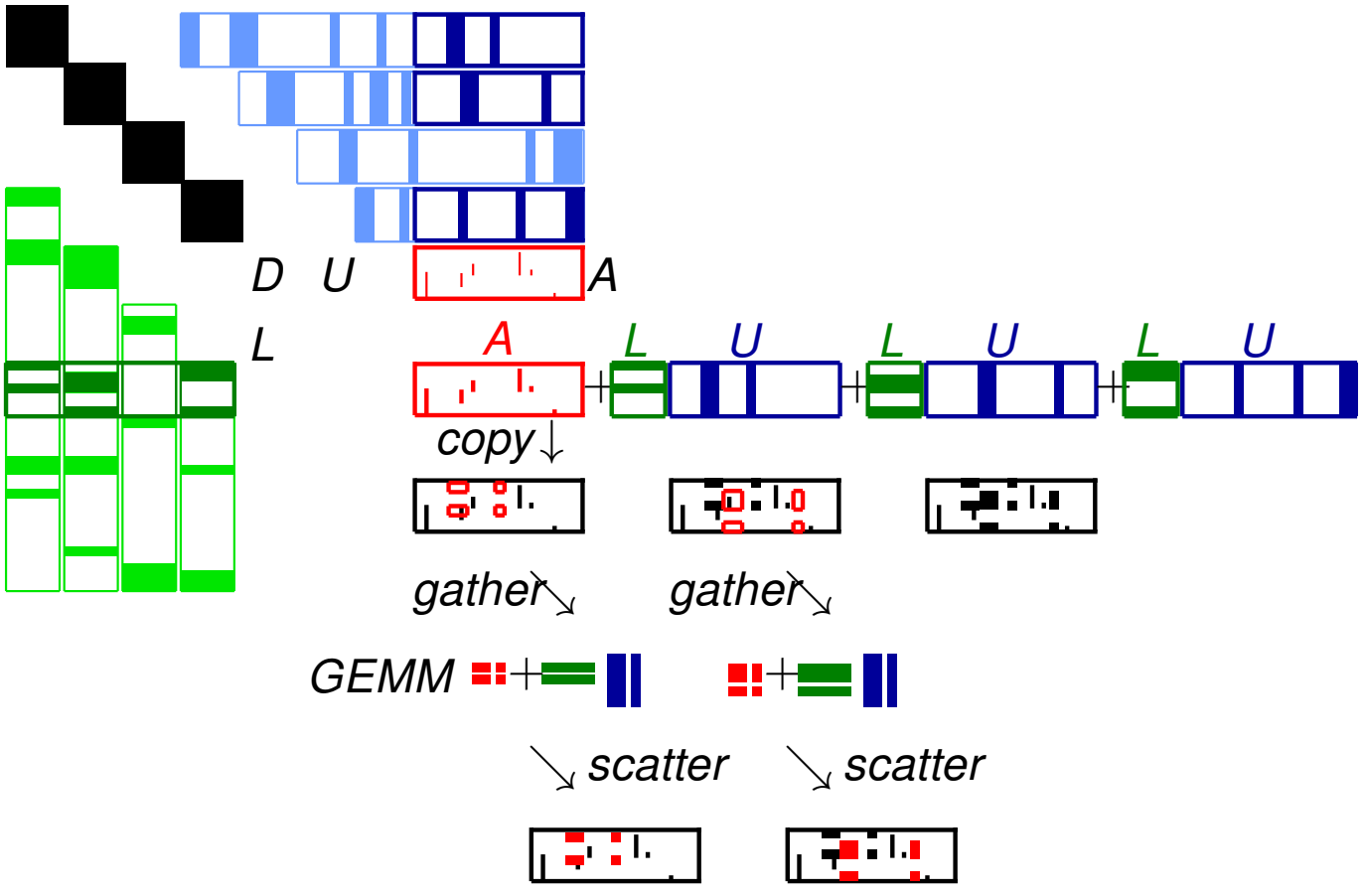
The Crout Block ILU



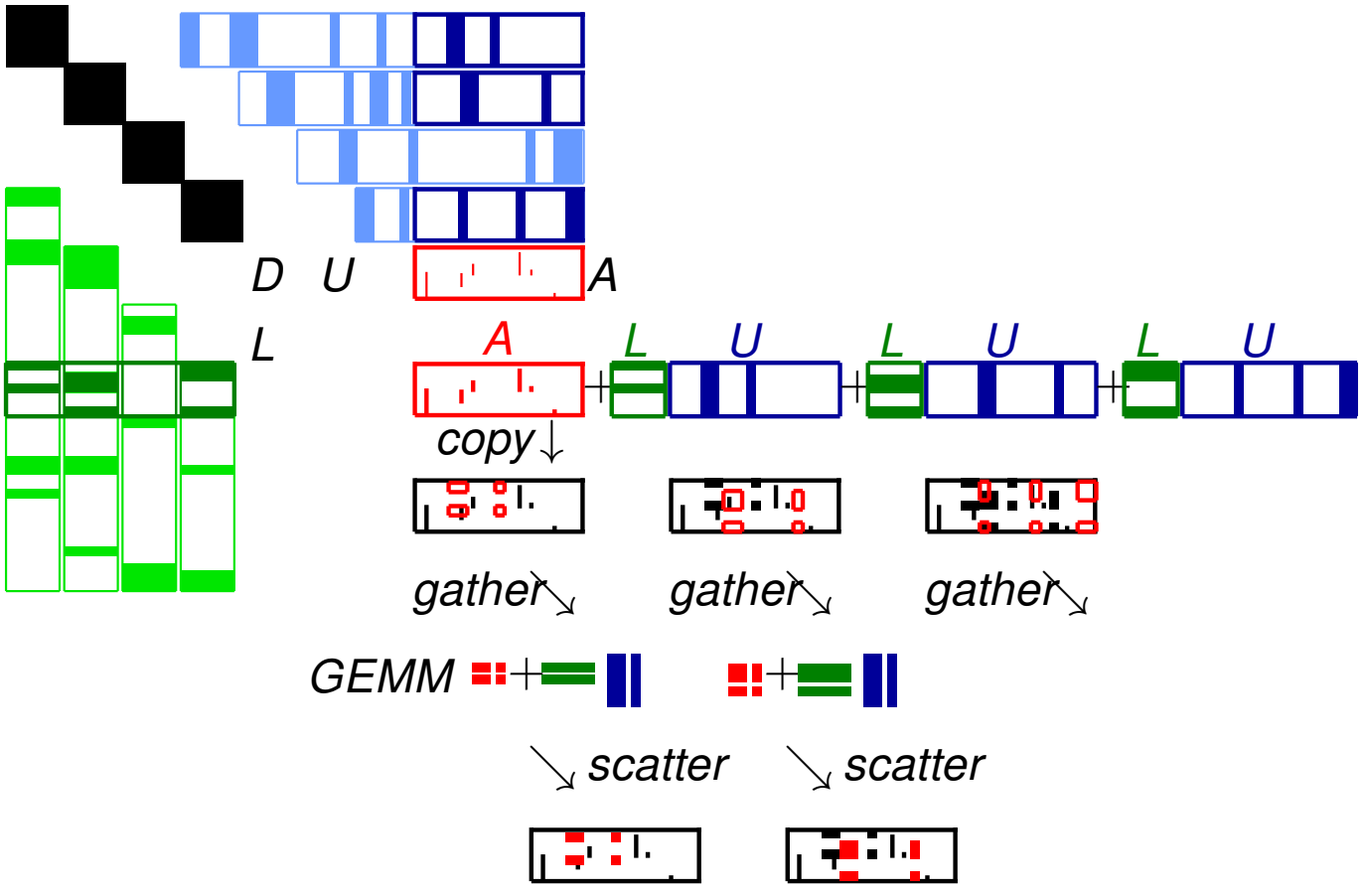
The Crout Block ILU



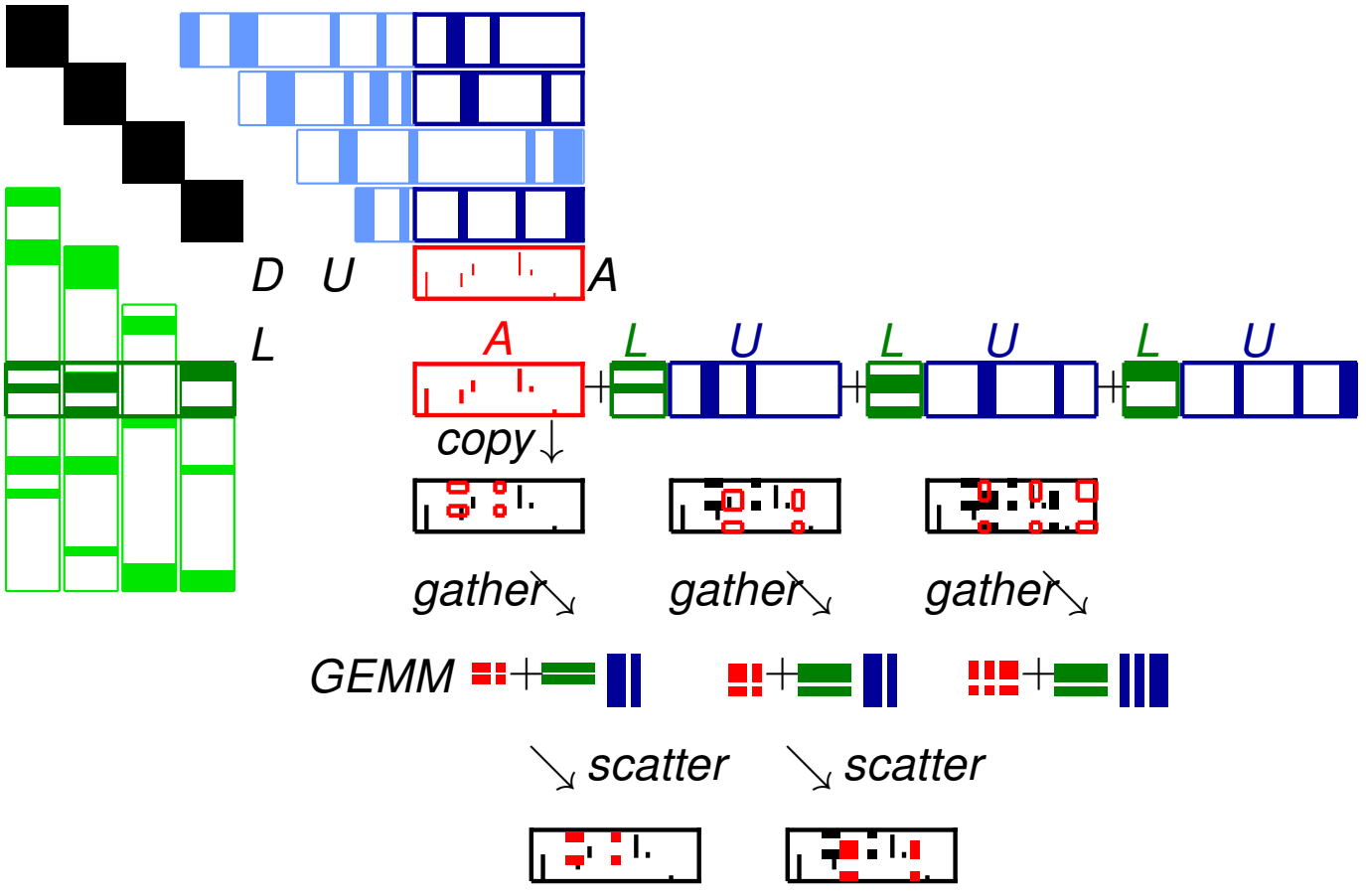
The Crout Block ILU



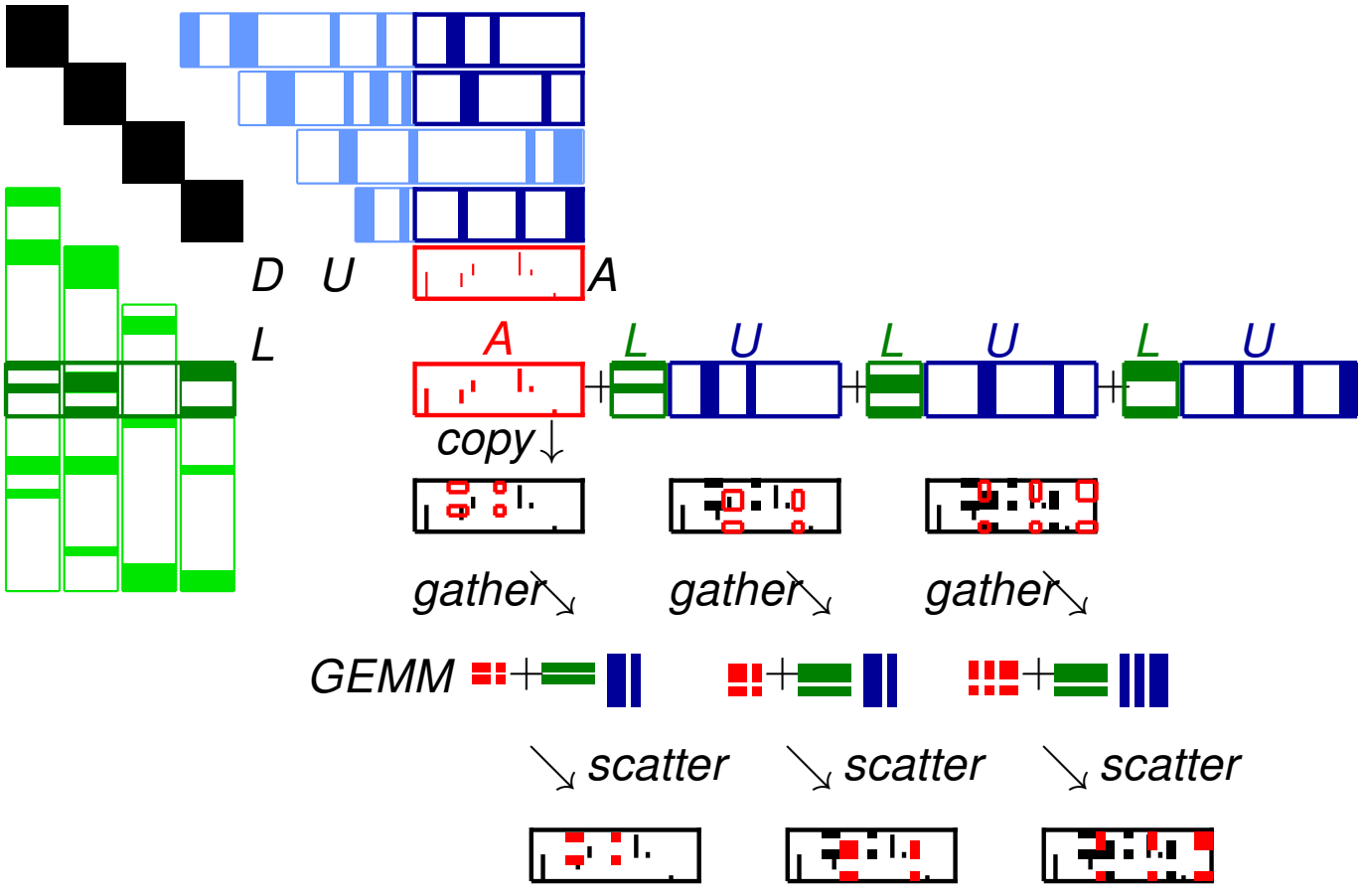
The Crout Block ILU



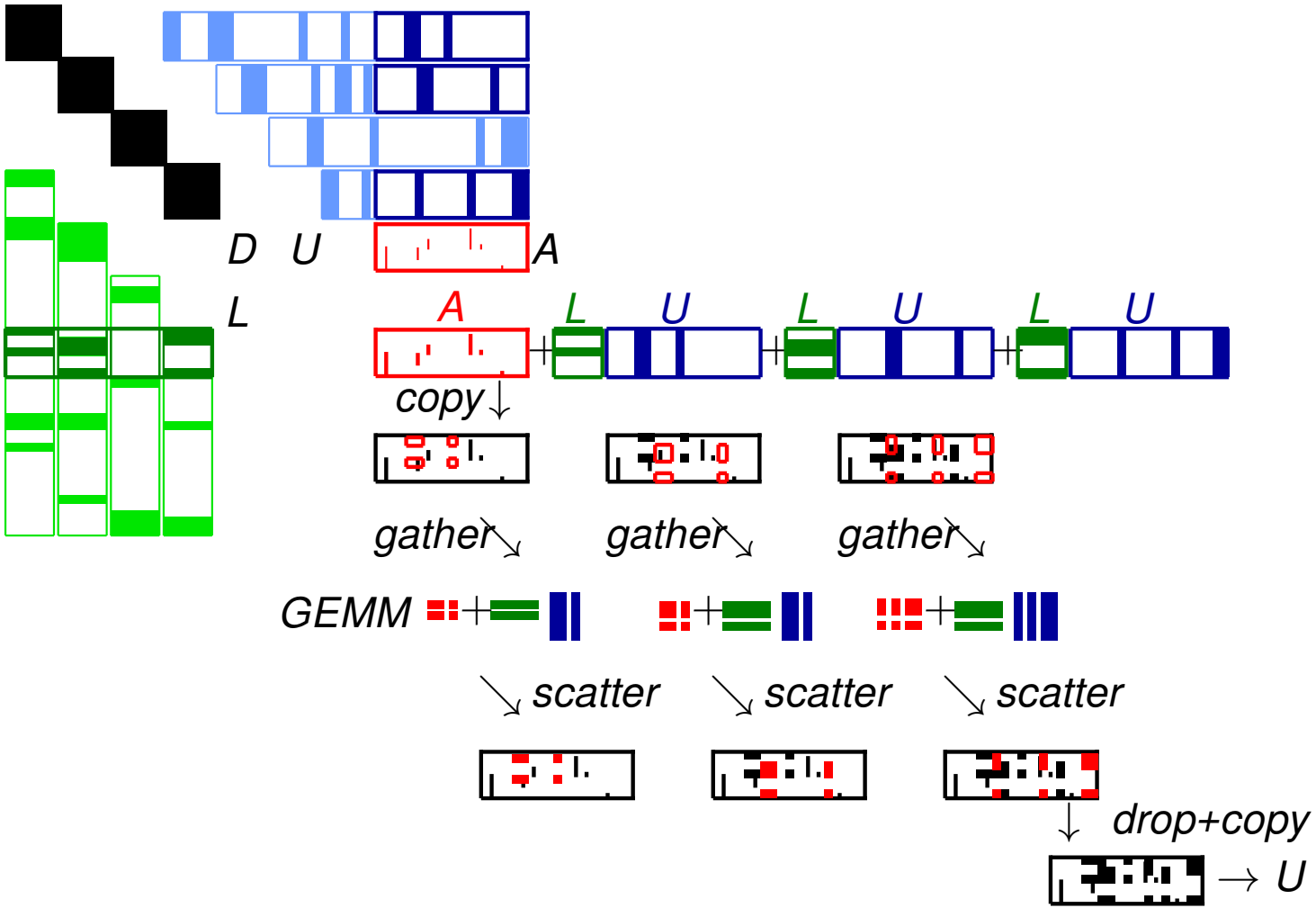
The Crout Block ILU



The Crout Block ILU



The Crout Block ILU



The Crout Block ILU

- downdating performed in two passes
  1. first pass: set up and merge nonzero index list for $L(k:n,k)$ (resp. $(U(k,k:n))$
  2. second pass: downdate $L(k:n,k)$
- As much as possible: multithreaded Intel MKL is used
  1. BLAS (level 1, 2 and 3)
  2. LAPACK
  3. gather, scatter
  4. sparse `dot` and `axpy`
- level–3 BLAS update used whenever sensible, scatter/gather skipped whenever possible
- scalar update bypasses level–3 BLAS and uses sparse `axpy` instead (similar bypasses for rank–1 update and blocks of small size)

Outline

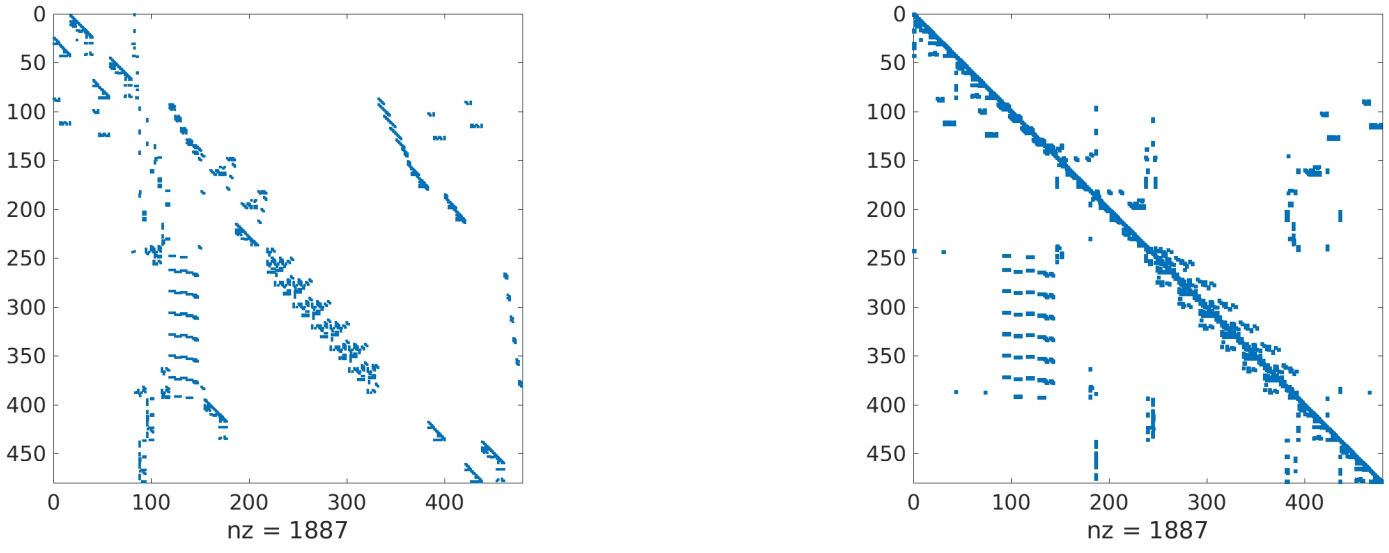
- Motivation
- Block ILUs
- **Setting Up and Improving the Block Structures**
- Numerical Results
- Numerical Results — Symmetric Indefinite Case
- Conclusions

Maximum Weight Matchings

Maximum weight matchings: combinatorial algorithm (MC64 [Duff ,Koster'99]) to improve the diagonal dominance

$$A \rightarrow D_l A D_r \Pi$$

D_l, D_r diagonal scalings, Π permutation



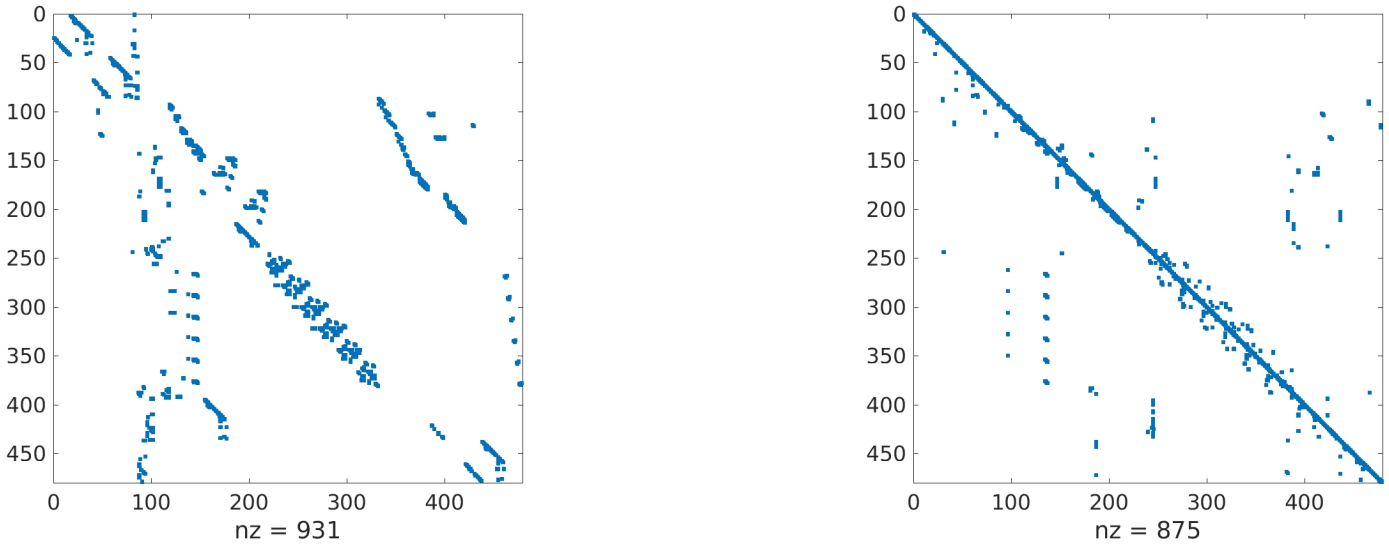
initial matrix versus permuted and rescaled matrix

Maximum Weight Matchings

Maximum weight matchings: combinatorial algorithm (MC64 [Duff ,Koster'99]) to improve the diagonal dominance

$$A \rightarrow D_l A D_r \Pi$$

D_l, D_r diagonal scalings, Π permutation



largest entries in modulus: initial matrix versus permuted and rescaled matrix

Cosine–Based Blocking

Idea: merge rows to block rows the their pattern is similar [Saad'03]

$$C := \text{pattern}(A),\ C = \begin{bmatrix} c_1^T \\ \vdots \\ c_n^T \end{bmatrix}$$

rows c_i^T and c_j^T are considered to have a similar pattern if

$$\frac{\|c_i\|_2 \cdot \|c_j\|_2}{c_i^T c_j} \geqslant \tau,\ \text{e.g.,}\ \tau = 0.8$$

- ratio refers to cosine of the angle between two rows
- integer–based algorithm (only counting)
- extremely fast if AA^T is sparse as well, otherwise simplifications required
- not always usefull

Cosine–Based Blocking — Example

sample matrix A (**venkat50**) from the SuiteSparse Matrix Collection.

- size $n = 62'424$, $nz = 1'717'777$ nonzero entries (approx. $\frac{nz}{n} = 27.5$ nonzeros per row/column)
- system arises from the discretization of the 2D compressible Euler equations. (velocity u, v, pressure p, density ρ)

Cosine–Based Blocking — Example

sample matrix A (**venkat50**) from the SuiteSparse Matrix Collection.

- size $n = 62'424$, $nz = 1'717'777$ nonzero entries (approx. $\frac{nz}{n} = 27.5$ nonzeros per row/column)
- system arises from the discretization of the 2D compressible Euler equations. (velocity u, v, pressure p, density ρ)

1. initially apply maximum weight matching (scalings+permutation)
2. apply cosine–based blocking.

system size	# dgl. blocks	max. size	avg. size	std. deviation
$62'424$	$15'723$	4	3.97	0.297

Fill–Reducing Symmetric Orderings

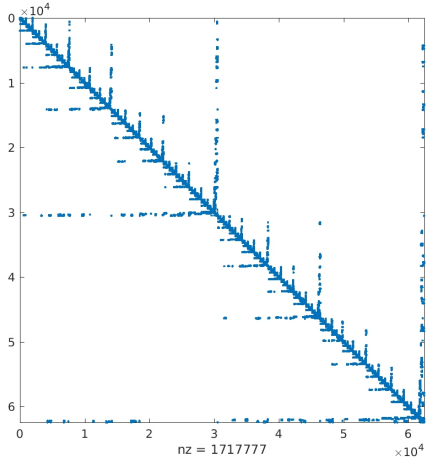
$$A \rightarrow P^T AP, \;\; P \text{ permutation}$$

- standard permutation routines to reduce the fill–in, e.g. approximate minimum degree [Amestoy,Davis,  '96], nested dissection [Karypis,Kumar'98], [LaSalle,Karypis'13] ...
- cosine–based blocking is preserved (apply permutation to the compressed graph)

Fill–Reducing Symmetric Orderings

$$A \rightarrow P^T AP, \;\; P \text{ permutation}$$

- standard permutation routines to reduce the fill–in, e.g. approximate minimum degree [Amestoy,Davis,  '96], nested dissection [Karypis,Kumar'98], [LaSalle,Karypis'13] ...
- cosine–based blocking is preserved (apply permutation to the compressed graph)



Matrix **venkat50** after maximum weight matching, cosine–based blocking and (MT-)METIS reordering based on the compressed graph

Block Pattern via Simplified ILU

- direct solvers build their blocks using combinatorial methods (elimination tree, supernodes)
- often these block structures are too big/too dense for ILUs.
- instead use a simplified quick–&–dirty ILU to guess a larger block pattern.

  rough idea: locally use ILU(1), i.e., roughly perform ILU on the pattern of A^2 with some drop tolerance τ and detect denser blocks
  $\longrightarrow$ ILU(1,τ)–blocking

Block Pattern via Simplified ILU

- direct solvers build their blocks using combinatorial methods (elimination tree, supernodes)
- often these block structures are too big/too dense for ILUs.
- instead use a simplified quick–&–dirty ILU to guess a larger block pattern.

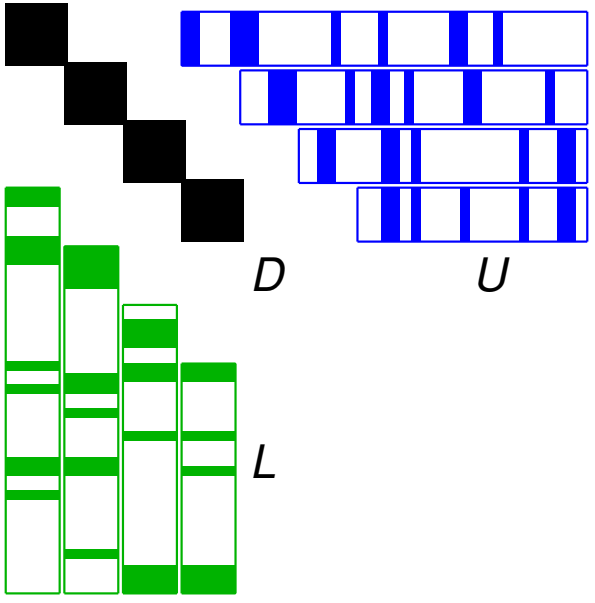
  rough idea: locally use ILU(1), i.e., roughly perform ILU on the pattern of A^2 with some drop tolerance τ and detect denser blocks
  $\longrightarrow$ ILU(1,τ)–blocking

sample matrix A (**venkat50**)

	# blocks	max. size	avg. size	std. deviation
only cosine	15′723	4	3.97	0.297
only ILU(1,10^{-2})	24′138	16	2.59	2.15
cosine+ILU(1,10^{-2})	13′786	16	4.53	1.50

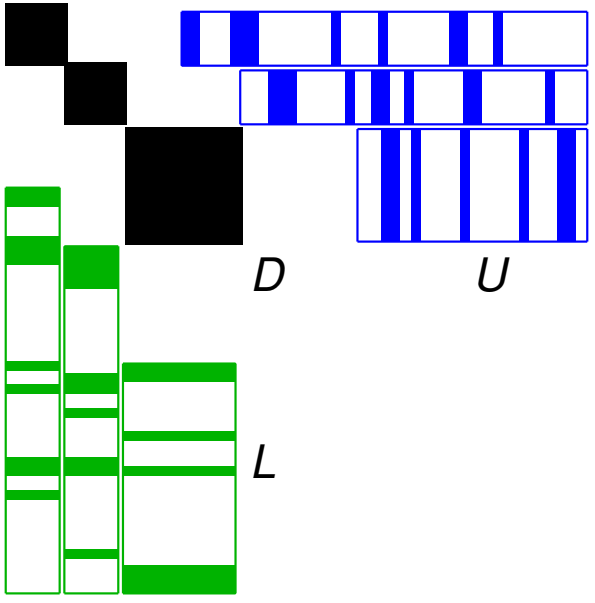
Progressive Aggregation

- improve block size during the factorization
- merge two consecutive block columns/row, when the additional fill is small



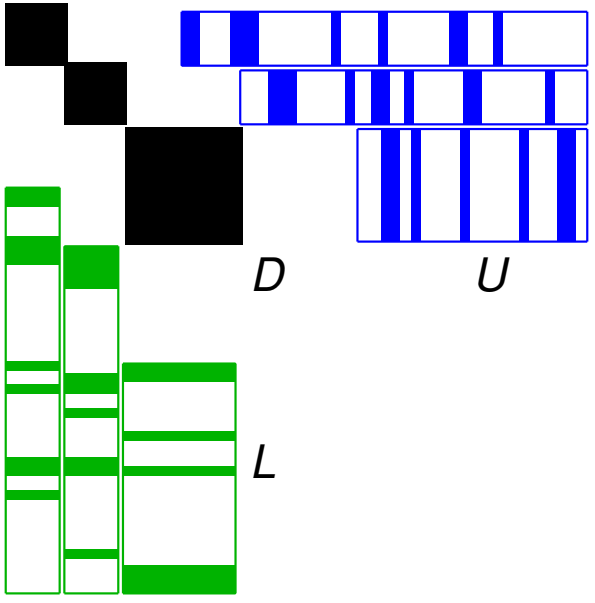
Progressive Aggregation

- improve block size during the factorization
- merge two consecutive block columns/row, when the additional fill is small



Progressive Aggregation

- improve block size during the factorization
- merge two consecutive block columns/row, when the additional fill is small



	# blocks	max. size	avg. size	std. deviation
cosine	15′723	4	3.97	0.297
ILU(1,10^{-2})	24′138	16	2.59	2.15
progr. aggr.	32′996	13	1.89	1.59
cosine+ILU(1,10^{-2})+progr.	8′259	44	7.56	5.23

Combined Approach

Sample matrix A (`venkat50`)

- use scalings + permutation by maximum weight matching (MC64[ ,Koster'99])
- use nested dissection (MT-METIS)
- iterative solution by GMRES(30)[Saad,Schultz'86], preconditioned by block ILU ('B ILU')

Combined Approach

Sample matrix A (`venkat50`)

- use scalings + permutation by maximum weight matching (MC64[ ,Koster'99])
- use nested dissection (MT-METIS)
- iterative solution by GMRES(30)[Saad,Schultz'86], preconditioned by block ILU ('B ILU')

blockings

- cosine-based 'c'
- ILU(1,τ)-based 'i'
- progressive aggregation 'p'

$\longrightarrow$ B ILU(cip), B ILU(c--), ...

Combined Approach

Sample matrix A (`venkat50`)

- use scalings + permutation by maximum weight matching (MC64[Duff,Koster'99])
- use nested dissection (MT-METIS)
- iterative solution by GMRES(30)[Saad,Schultz'86], preconditioned by block ILU ('B ILU')

blockings

- cosine-based 'c'
- ILU(1,τ)-based 'i'
- progressive aggregation 'p'

$\longrightarrow$ B ILU(cip), B ILU(c--), ...

	time ILU[sec]	$\frac{nz(L+U)}{nz(A)}$	time GMRES[sec]	# steps
B ILU(c--)	1.9	4.3	3.0	29
B ILU(-i-)	3.9	4.0	4.0	31
B ILU(--p)	3.7	3.5	7.4	52
B ILU(cip)	1.9	4.9	2.3	26

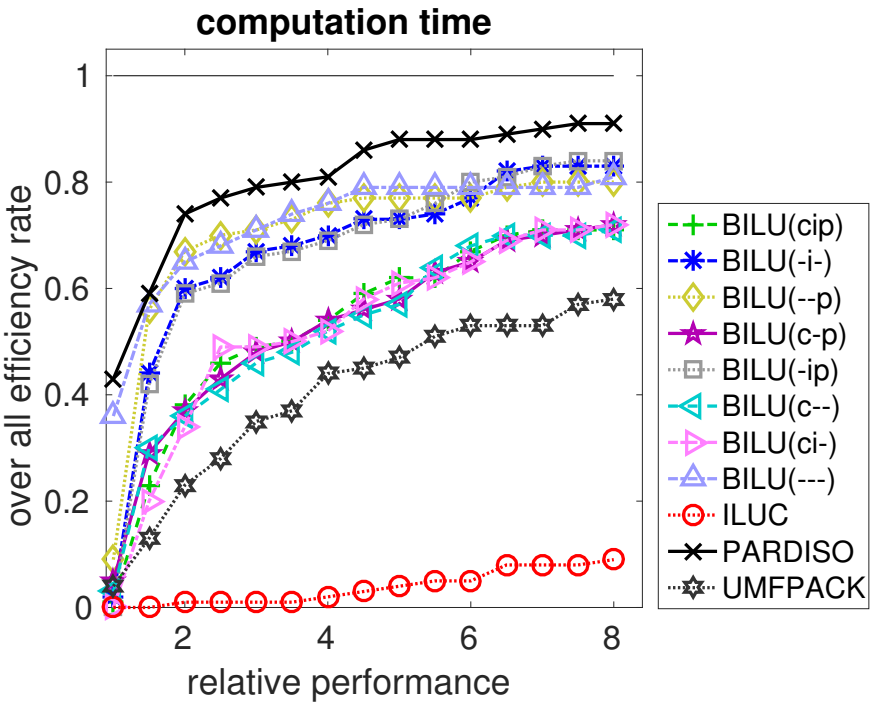
Outline

- Motivation
- Block ILUs
- Setting Up and Improving the Block Structures
- **Numerical Results**
- Numerical Results — Symmetric Indefinite Case
- Conclusions

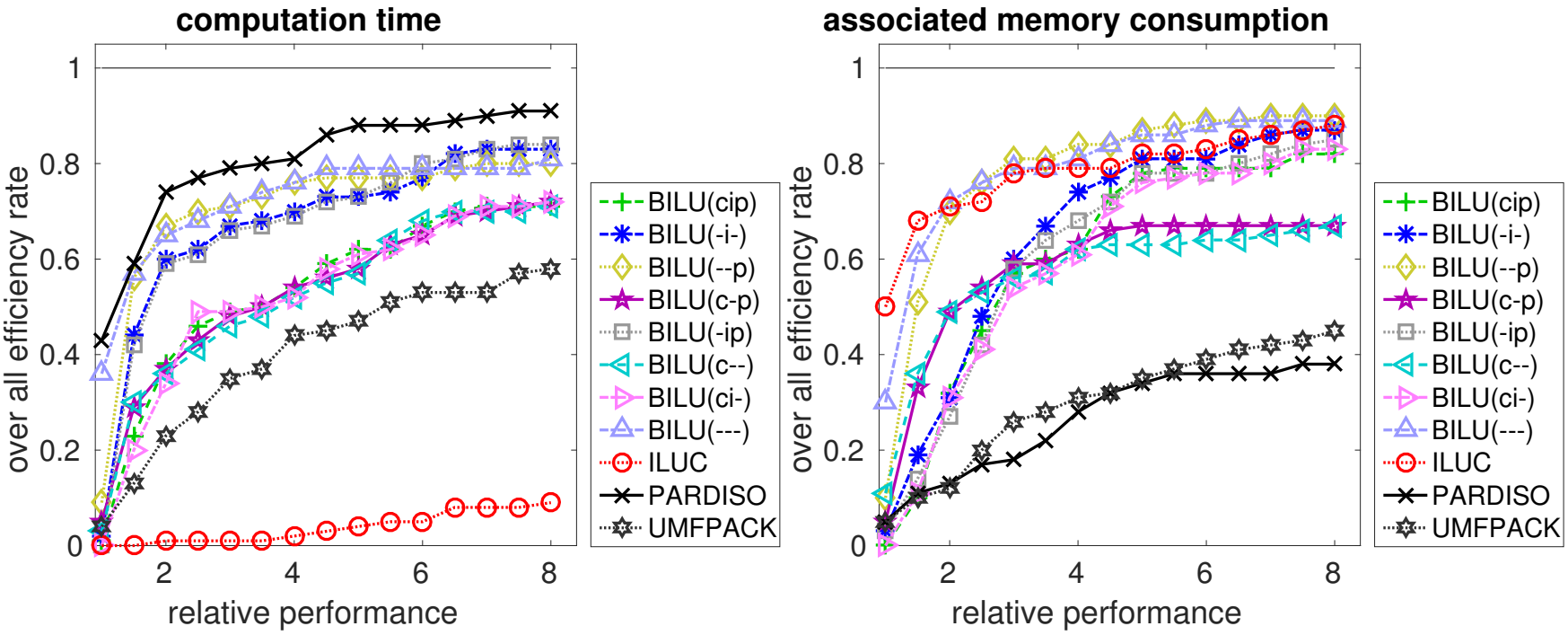
Numerical Results

- 1 TB main memory, 4 Intel Xeon E7-4880 v2 @ 2.5 GHz processors each of them having 12 cores on a socket leading to 60 cores in total.
- MATLAB R2015b as front end using Intel MKL 11.1/11.2
- Numerical methods
  - PARDISO, UMFPACK (as implemented in MATLAB)
  - B ILU(***) + GMRES(30) with relative residual of 10^{-6}
  - MATLAB's Crout ILU (referred to as "ILUC") + GMRES(30) with relative residual of 10^{-6}
- 100 matrices of size $\geqslant 10'000$ from the SuiteSparse Matrix Collection
- performance profile

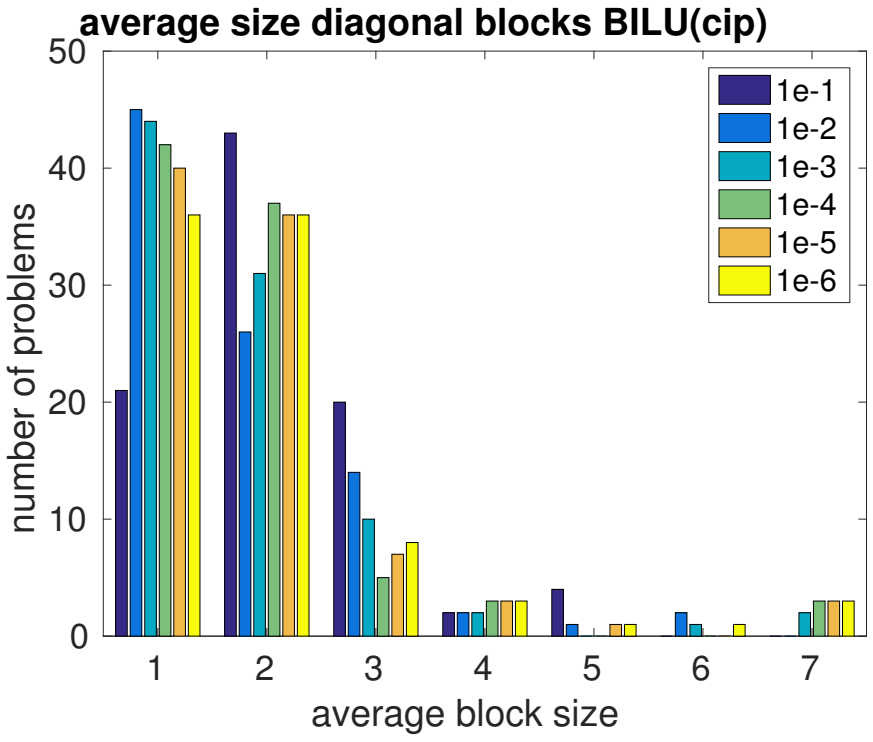
Numerical Results — Performance Profiles



Numerical Results — Performance Profiles



Numerical Results — Block Size of Diagonal Blocks



Average block size of B ILU(cip) depending on the drop tolerance.

Outline

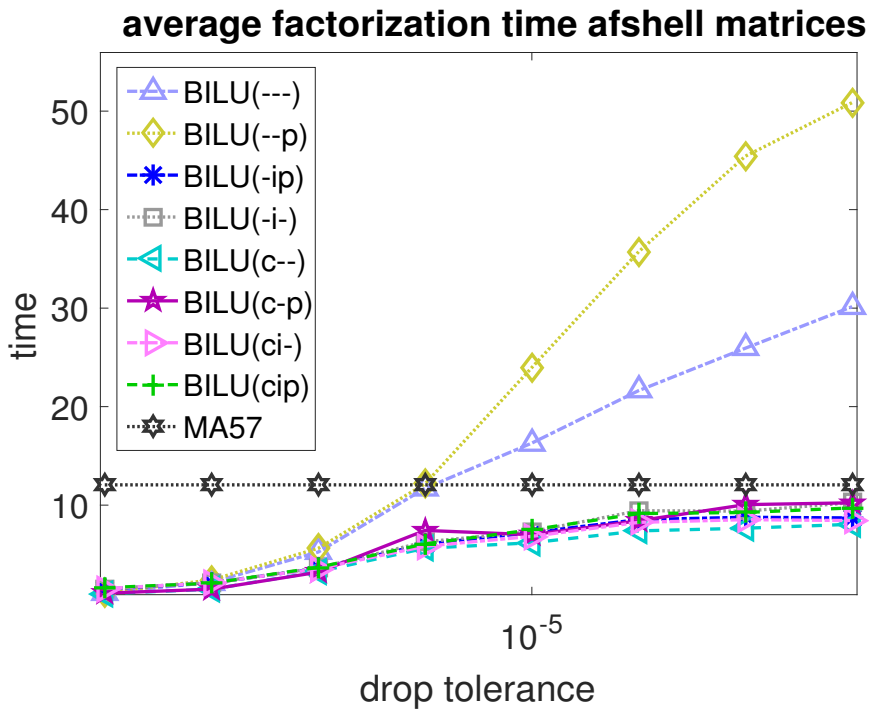
- Motivation
- Block ILUs
- Setting Up and Improving the Block Structures
- Numerical Results
- **Numerical Results — Symmetric Indefinite Case**
- Conclusions

Setup Indefinite Case

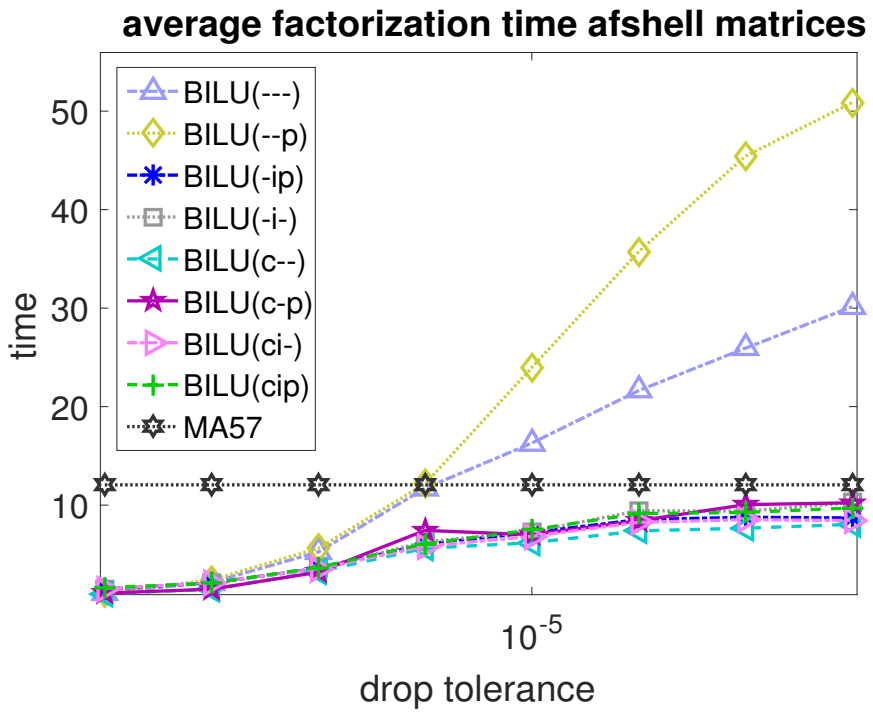
- Symmetrized maximum weight matching [Duff,Pralet'05]
- symmetric reordering applied to the compressed graph (2×2 pivots are collapsed), e.g. (MT-)METIS, AMD
- apply symmetric indefinite variant of B ILU $\rightarrow$ B ILDL
- iterative solver simplified QMR [Freund,Nachtigal'95],[Freund,Jarre'97]
- use symmetrized version of blocking strategies (cosine, BILDL(1,τ), progressive aggregation)
- compare B ILDL with MA57 [

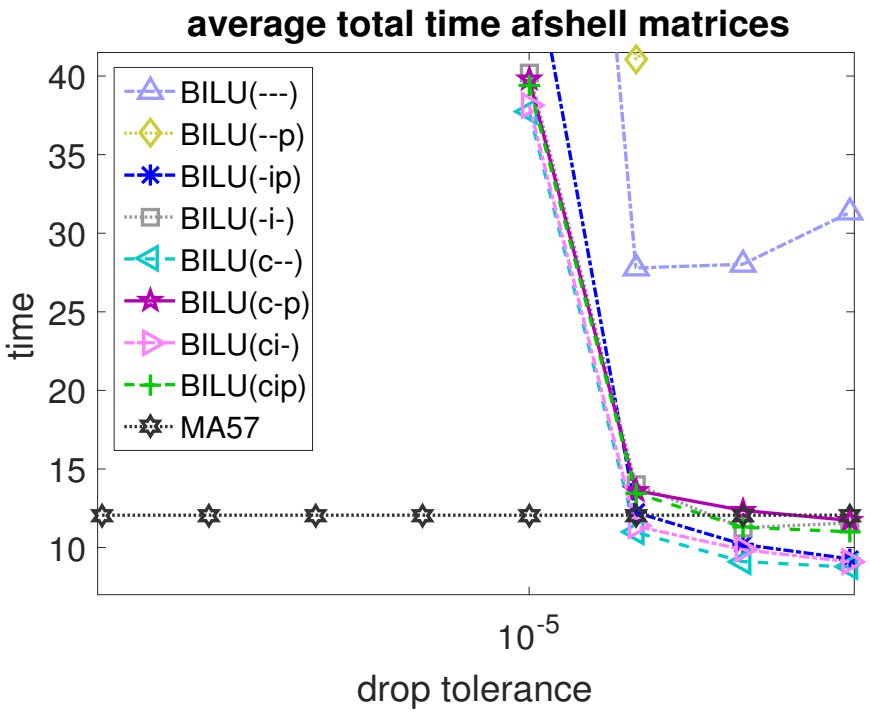
et al.] as implemented in MATLAB

Results Indefinite Case

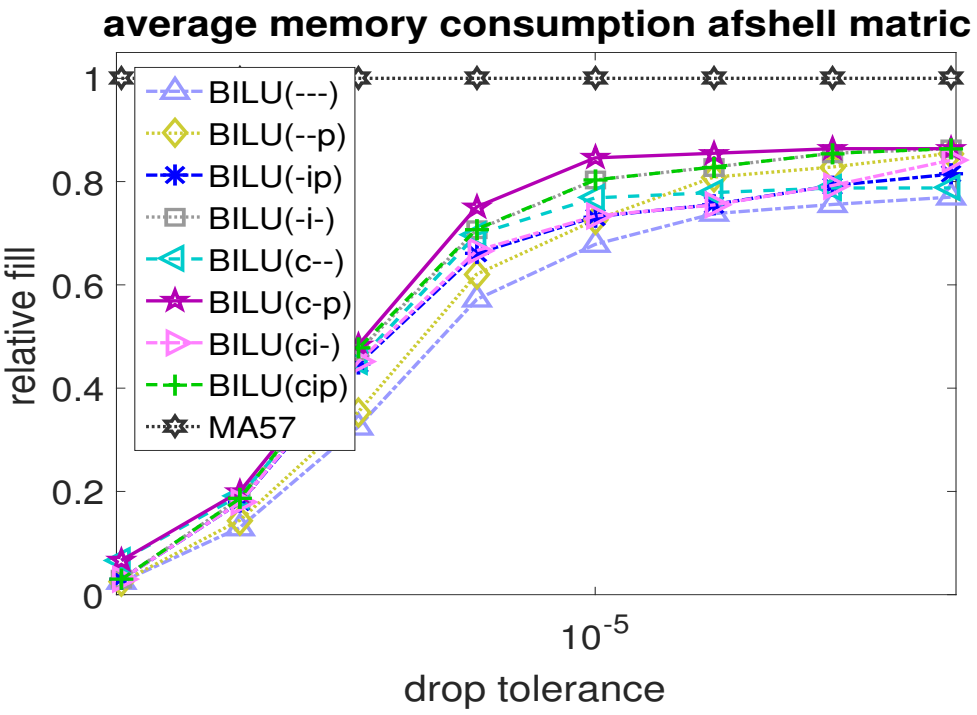


Results Indefinite Case





Results Indefinite Case



Outline

- Motivation
- Block ILUs
- Setting Up and Improving the Block Structures
- Numerical Results
- Numerical Results — Symmetric Indefinite Case
- **Conclusions**

Conclusions

- B ILU leads to a high performance ILU method
- major impact: efficient use of blocking strategies, level–3 BLAS and multithreading
- drawback: straightforward approach (no pivoting, just perturbations)
- general case (real/complex) and SPD case (real/complex), and symmetric indefinite case (real/complex) implemented
- used as template as part of an approximate selective inversion method and inside a large–scale sparse inverse covariance matrix estimation approach

References

P. Amestoy, T. A. Davis, and I. S. Duff. An approximate minimum degree ordering algorithm. *SIAM J. Matrix Analysis and Applications*, 17(4):886–905, 1996.

M. Benzi, J. Haws, and M. Tuma. Preconditioning highly indefinite and nonsymmetric matrices. *SIAM J. Scientific Computing*, 22(4):1333–1353, 2000.

T. A. Davis. Algorithm 832: UMFPACK V4.3—an unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw.*, 30(2):196–199, 2004.

I. S. Duff, A. Erisman, and J. Reid. *Direct Methods for Sparse Matrices (2nd edition)*. Oxford University Press, 2017.

I. S. Duff and J. Koster. The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. *SIAM J. Matrix Analysis and Applications*, 20(4):889–901, 1999.

I. S. Duff and S. Pralet. Strategies for scaling and pivoting for sparse symmetric indefinite problems. *SIAM J. Matrix Analysis and Applications*, 27(2):313–340, 2005.

G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Scientific Computing*, 20(1):359–392, 1998.

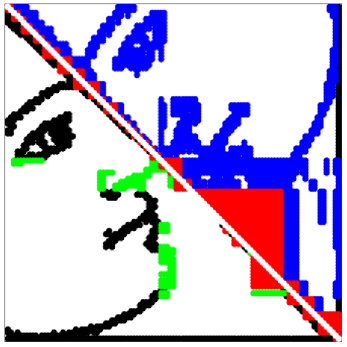
J. K. P. R. Amestoy, I. S. Duff and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal of Matrix Analysis and Applications*, 23(1):15–41, 2001.

Y. Saad. Finding exact and approximate block structures for ILU preconditioning. *SIAM J. Scientific Computing*, 24(4):1107–1123, 2003.

Y. Saad and M. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 7:856–869, 1986.

Epilogue

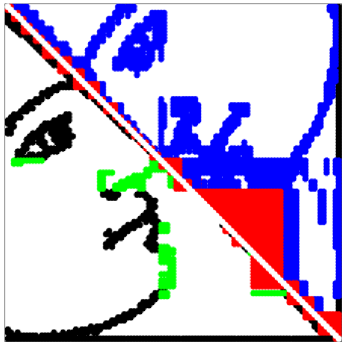
J A N U S



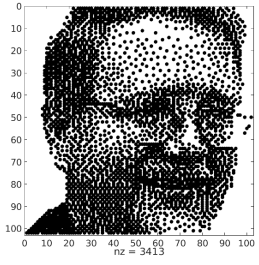
http://www.icm.tu-bs.de/~bolle/janus

Epilogue

J A N U S

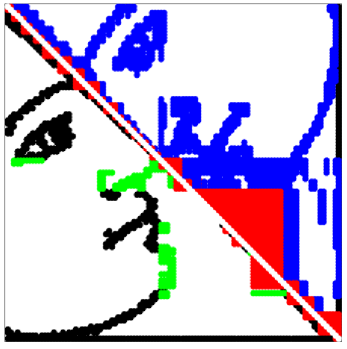


http://www.icm.tu-bs.de/~bolle/janus

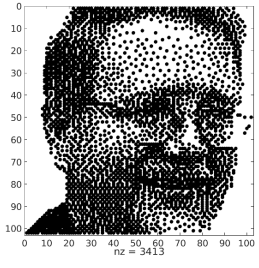


Epilogue

J A N U S

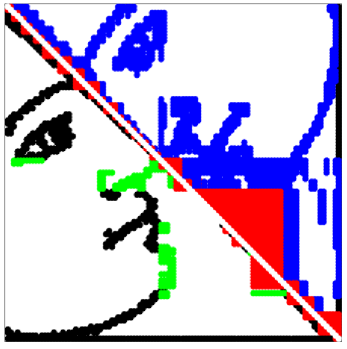


http://www.icm.tu-bs.de/~bolle/janus

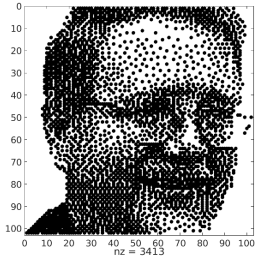


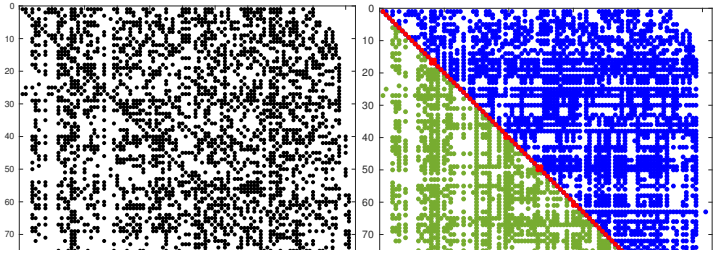
Epilogue

J A N U S



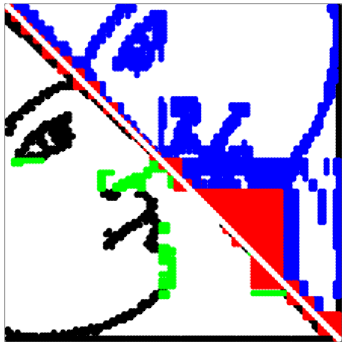
http://www.icm.tu-bs.de/~bolle/janus



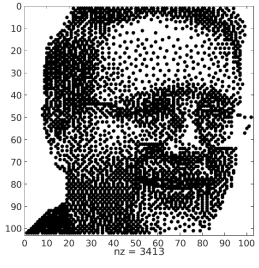


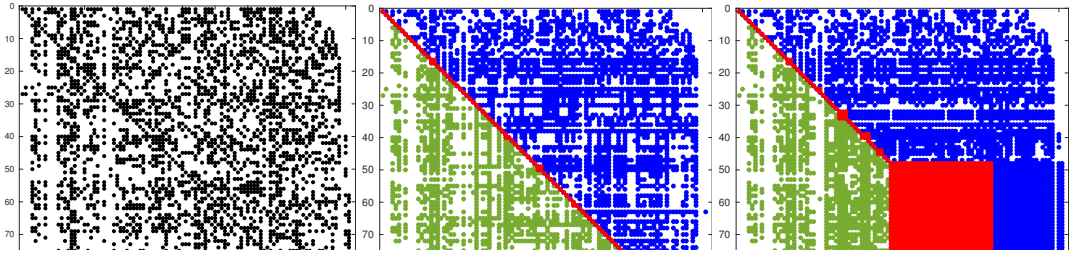
Epilogue

J A N U S



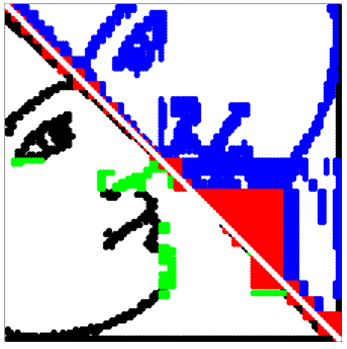
http://www.icm.tu-bs.de/~bolle/janus



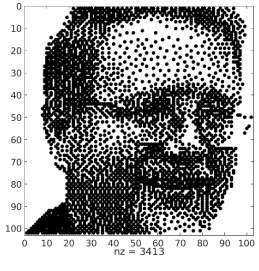


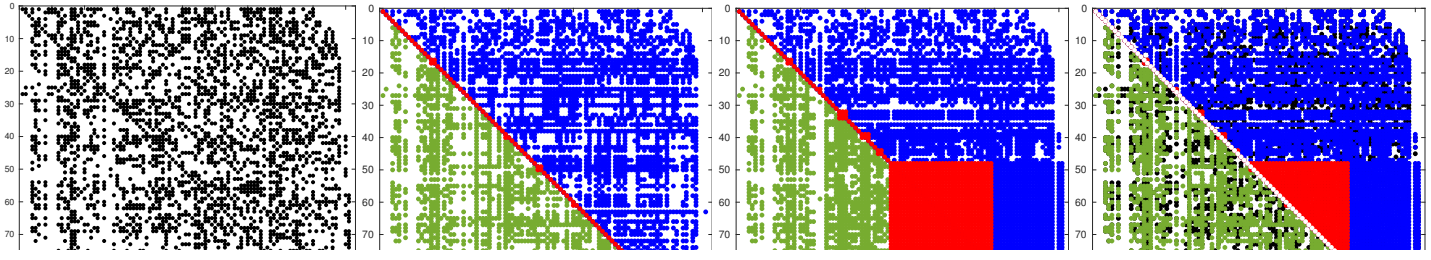
Epilogue

J A N U S



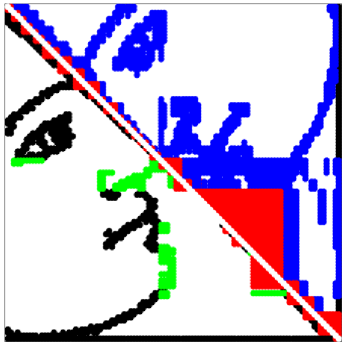
http://www.icm.tu-bs.de/~bolle/janus



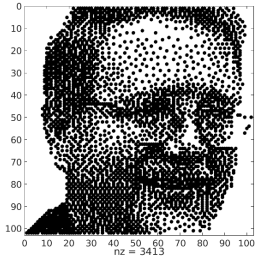


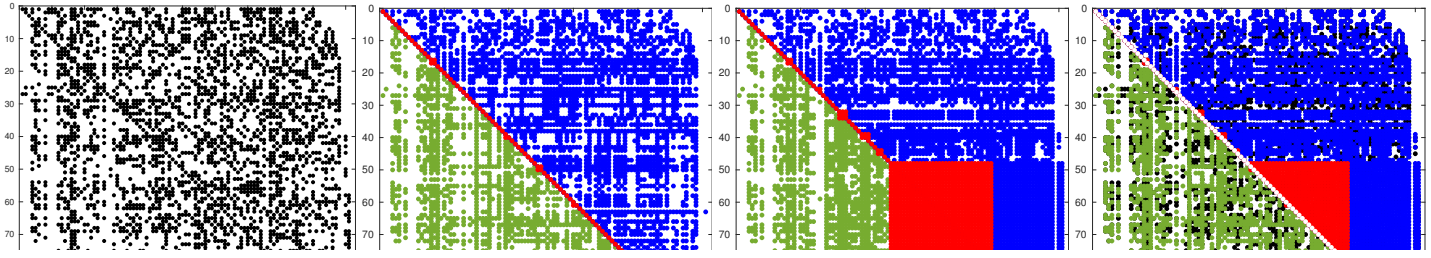
Epilogue

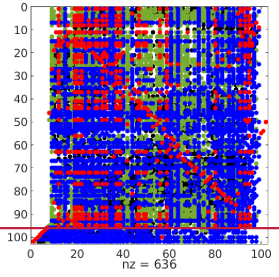
J A N U S



http://www.icm.tu-bs.de/~bolle/janus

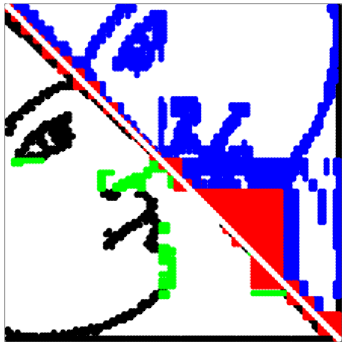






Epilogue

J A N U S



http://www.icm.tu-bs.de/~bolle/janus