Very Fine-Grained Parallelization of Approximate Sparse Matrix Computations

Edmond Chow
School of Computational Science and Engineering
Georgia Institute of Technology

Innovative Computing Lab, UTK, Oct. 12, 2015

Two sparse matrix computations

- A new method for parallelizing the construction of incomplete LU factorizations

$$A \approx LU$$

  Method designed for large amounts of fine-grained parallelism, such as in GPUs, Intel Xeon Phi, and future processors

- Approximate sparse triangular solves using iterative methods

NVIDIA GPU with 2880 cores; Intel Xeon Phi with 60 cores and 240 threads.

Collaborators

- Hartwig Anzt, ICL – GPU implementations for iterative ILU and iterative triangular solves, including asynchronous versions and use in model order reduction applications
- Jennifer Scott, Harwell Lab – comprehensive tests with iterative triangular solves

Conventional ILU factorization

Given sparse A, compute $LU \approx A$, where L and U are sparse.

Define S to be the sparsity pattern, $(i, j) \in S$ if l_{ij} or u_{ij} can be nonzero.

```
for i = 2 to n do
    for k = 1 to i − 1 and (i, k) ∈ S do
        a_ik = a_ik / a_kk
        for j = k + 1 to n and (i, j) ∈ S do
            a_ij = a_ij − a_ik a_kj
        end
    end
end
```

Existing parallel ILU methods

At each step, find all rows that can be eliminated in parallel (rows that only depend on rows already eliminated)

Level scheduling ILU

Regular grids: van der Vorst 1989, Joubert-Oppe 1994
Irregular problems: Heroux-Vu-Yang 1991, Pakzad-Lloyd-Phillips 1997, Gonzalez-Cabaleiro-Pena 1999, Dong-Cooperman 2011, Gibou-Min 2012, Naumov 2012
Triangular solves: Anderson-Saad 1989, Saltz 1990, Hammond-Schreiber 1992

Multicolor reordering ILU

Poole-Ortega 1987, Elman-Agron 1989, Jones-Plassmann 1994, Nakajima 2005, Li-Saad 2010, Heuveline-Lukarski-Weiss 2011

Domain decomposition ILU

Ma-Saad 1994, Karypis-Kumar 1997, Vuik et al 1998, Hysom-Pothen 1999, Magolu monga Made-van der Vorst 2002

Fine-grained parallel ILU factorization

An ILU factorization, $A \approx LU$, with sparsity pattern S has the property

$$(LU)_{ij} = a_{ij}, \quad (i,j) \in S.$$

Instead of Gaussian elimination, we compute the *unknowns*

$$\begin{aligned} l_{ij}, \qquad & i > j, \qquad (i,j) \in S \\ u_{ij}, \qquad & i \le j, \qquad (i,j) \in S \end{aligned}$$

using the *constraints*

$$\sum_{k=1}^{\min(i,j)} l_{ik} u_{kj} = a_{ij}, \quad (i,j) \in S.$$

If the diagonal of L is fixed, then there are $|S|$ unknowns and $|S|$ constraints.

Solving the constraint equations

The equation corresponding to (i,j) gives

$$\begin{aligned} l_{ij} &= \frac{1}{u_{jj}}\left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj}\right), \quad i > j \\ u_{ij} &= a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj}, \qquad i \leq j. \end{aligned}$$

The equations have the form $x = G(x)$. It is natural to try to solve these equations via a fixed-point iteration

$$x^{(k+1)} = G(x^{(k)})$$

with an initial guess $x^{(0)}$. We update each component of $x^{(k+1)}$ in parallel and asynchronously (each thread uses latest available values).

Ref: Frommer-Szyld 2000

Fine-grained ILU algorithm

Set unknowns l_{ij} and u_{ij} to initial values
for *sweep* $= 1, 2, \ldots$ *until convergence* **do**
  parallel for $(i, j) \in S$ **do**
    if $i > j$ **then**
      $l_{ij} = \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj} \right) / u_{jj}$
    else
      $u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj}$
    end
  end
end

Arithmetic intensity can be tuned by controlling how often updates are exposed to other threads, at the cost of convergence degradation.

Actual implementation uses sparse data structures.

A non-intuitive approach

Write matrix factorizations as bilinear equations and then solve asynchronously

- More bilinear equations than original equations
- Equations are nonlinear

Potential advantages

- Lots of parallelism: up to one thread per nonzero in L and U
- Do not need to solve the nonlinear equations exactly (no need to compute the incomplete factorization exactly)
- Nonlinear equations may have a good initial guess (e.g., time-dependent problems)
- Rich structure in the nonlinear equations that can be exploited

Measuring convergence of the nonlinear iterations

Nonlinear residual

$$\|(A - LU)_S\|_F = \left[\sum_{(i,j)\in S} \left(a_{ij} - \sum_{k=1}^{\min(i,j)} l_{ik} u_{kj} \right)^2 \right]^{1/2}$$

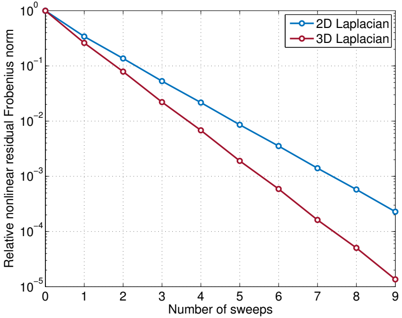
(or some other norm)

ILU residual

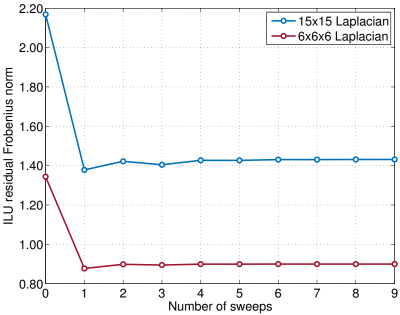
$$\|A - LU\|_F$$

Convergence of the preconditioned linear iterations is known to be strongly related to the ILU residual

Laplacian problem



Relative nonlinear residual norm



ILU residual norm

An aside: what to optimize?

Example: Laplacian on 30×30 grid, natural ordering, scaled

	$\|A - LU\|_F$	PCG count
Traditional ILU	2.98	27
Sparsification of exact LU factors	3.96	24
Optimization of norm	2.70	31

Numerical tests for new parallel ILU algorithm

- Do the asynchronous iterations converge?
- How fast is convergence with the number of threads?
- How good are the approximate factorizations as preconditioners?

Measure performance in terms of solver iteration count.

Tests on Intel Xeon Phi.

Initial L and U are the lower and upper triangular parts of A.

Use Gaussian elimination ordering for the nonlinear equations.

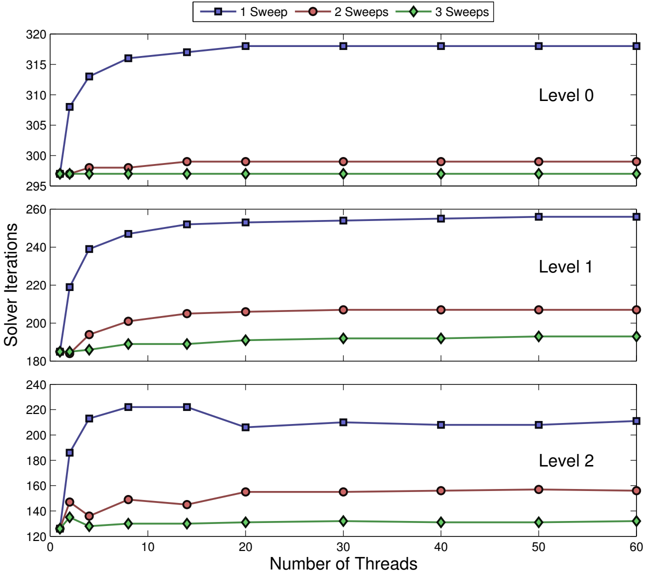
2D FEM Laplacian, $n = 203841$, RCM ordering, 240 threads on Intel Xeon Phi

	Level 0			Level 1			Level 2		
Sweeps	PCG iter	nonlin resid	ILU resid	PCG iter	nonlin resid	ILU resid	PCG iter	nonlin resid	ILU resid
0	404	1.7e+04	41.1350	404	2.3e+04	41.1350	404	2.3e+04	41.1350
1	318	3.8e+03	32.7491	256	5.7e+03	18.7110	206	7.0e+03	17.3239
2	301	9.7e+02	32.1707	207	8.6e+02	12.4703	158	1.5e+03	6.7618
3	298	1.7e+02	32.1117	193	1.8e+02	12.3845	132	4.8e+02	5.8985
4	297	2.8e+01	32.1524	187	4.6e+01	12.4139	127	1.6e+02	5.8555
5	297	4.4e+00	32.1613	186	1.4e+01	12.4230	126	6.5e+01	5.8706
IC	297	0	32.1629	185	0	12.4272	126	0	5.8894

Very small number of sweeps required

(Chow and Patel, SISC 2015)

2D FEM Laplacian, $n = 203841$, RCM ordering

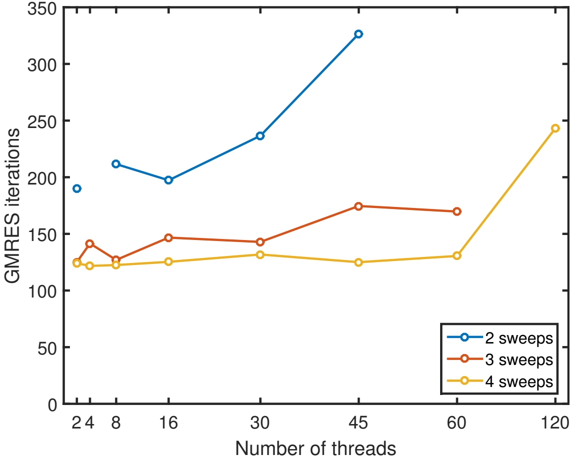


Univ. Florida sparse matrices (SPD cases) 240 threads on Intel Xeon Phi

	Sweeps	Nonlin Resid	PCG iter
af_shell3	0	1.58e+05	852.0
	1	1.66e+04	798.3
	2	2.17e+03	701.0
	3	4.67e+02	687.3
	IC	0	685.0
thermal2	0	1.13e+05	1876.0
	1	2.75e+04	1422.3
	2	1.74e+03	1314.7
	3	8.03e+01	1308.0
	IC	0	1308.0
ecology2	0	5.55e+04	2000+
	1	1.55e+04	1776.3
	2	9.46e+02	1711.0
	3	5.55e+01	1707.0
	IC	0	1706.0
apache2	0	5.13e+04	1409.0
	1	3.66e+04	1281.3
	2	1.08e+04	923.3
	3	1.47e+03	873.0
	IC	0	869.0

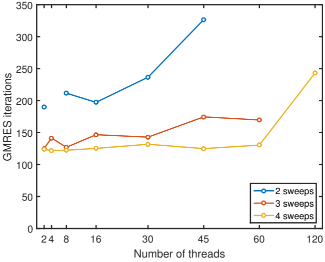
	Sweeps	Nonlin Resid	PCG iter
G3_circuit	0	1.06e+05	1048.0
	1	4.39e+04	981.0
	2	2.17e+03	869.3
	3	1.43e+02	871.7
	IC	0	871.0
offshore	0	3.23e+04	401.0
	1	4.37e+03	349.0
	2	2.48e+02	299.0
	3	1.46e+01	297.0
	IC	0	297.0
parabolic_fem	0	5.84e+04	790.0
	1	1.61e+04	495.3
	2	2.46e+03	426.3
	3	2.28e+02	405.7
	IC	0	405.0

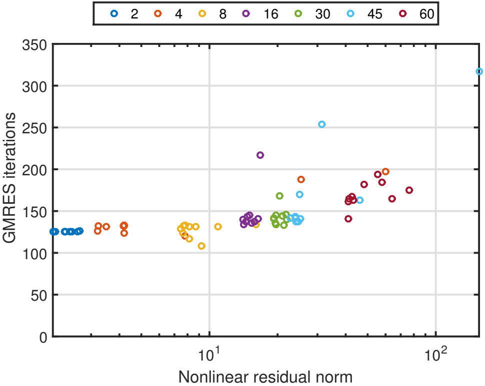
BCSSTK24 matrix, ILU(1)



Each point is average of 10 trials.
SPD version of algorithm generally fails (negative pivots), especially with more sweeps.

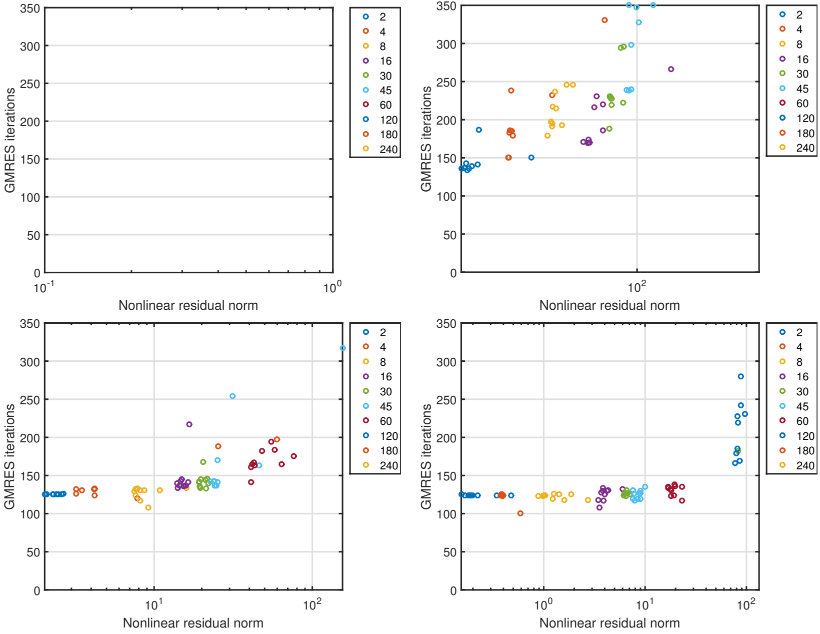
BCSSTK24 matrix, ILU(1)



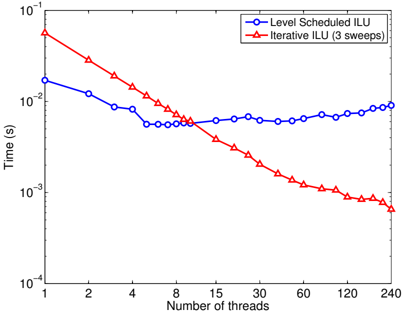


3 sweeps

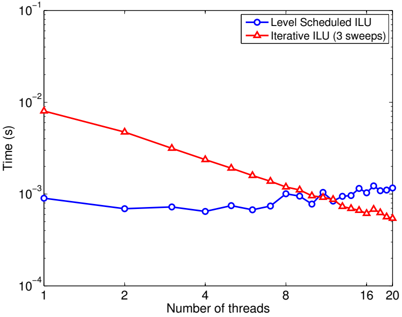
BCSSTK24 matrix, ILU(1)



Timing comparison, ILU(2) on 100 × 100 grid (5-point stencil)



Intel Xeon Phi



Intel Xeon E5-2680v2, 20 cores

Results for NVIDIA Tesla K40c

	PCG iteration counts for given number of sweeps							Timings [ms]		
	IC	0	1	2	3	4	5	IC	5 swps	s/up
apache2	958	1430	1363	1038	965	960	958	61.	8.8	6.9
ecology2	1705	2014	1765	1719	1708	1707	1706	107.	6.7	16.0
G3_circuit	997	1254	961	968	993	997	997	110.	12.1	9.1
offshore	330	428	556	373	396	357	332	219.	25.1	8.7
parabolic_fem	393	763	636	541	494	454	435	131.	6.1	21.6
thermal2	1398	1913	1613	1483	1341	1411	1403	454.	15.7	28.9

IC denotes the exact factorization computed using the NVIDIA cuSPARSE library.

(Chow-Anzt-Dongarra 2015)

Iterative Threshold Incomplete Cholesky

Idea: Modify a given pattern to obtain a better pattern with the same number of nonzeros

Algorithm:

$L =$ initial pattern and values (e.g., tril(a))

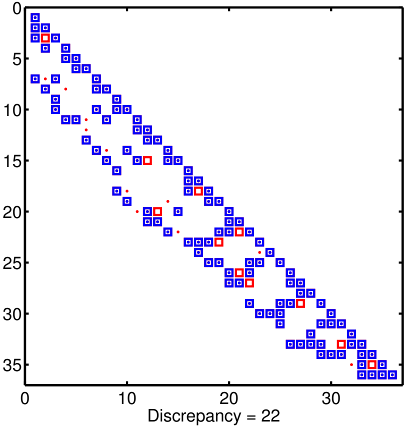
repeat

- Remove m smallest elements in L
- Run 1 sweep of fixed-point method
- Construct candidate nonzeros for L
- Compute residuals for candidate nonzeros:
  $r_{ij} = \left| a_{ij} - \sum_{k=1}^{\min(i,j)} l_{ik} u_{kj} \right|$
- Add m elements to L (candidates with largest residuals)
- Run 1 sweep of fixed-point method

until *pattern no longer changes*

Anisotropic test problems

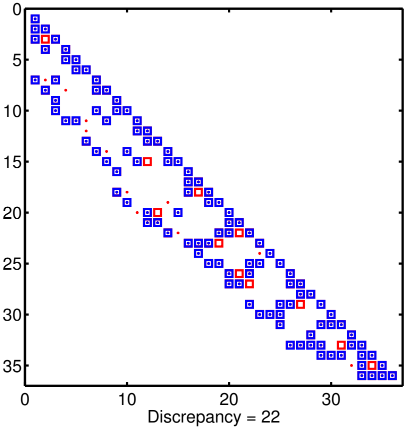
ANI1 test problem, IC pattern



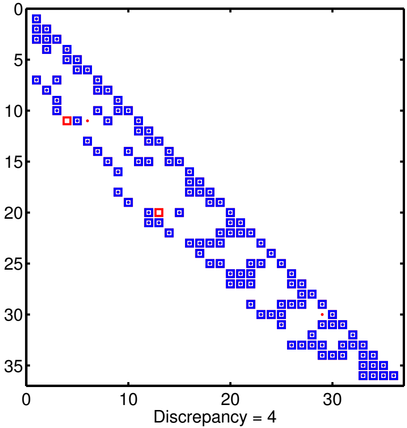
Squares: ICT pattern
Dots: IC(0) pattern
(same number of nonzeros)

- Anisotropic diffusion on square domain, Dirichlet BC
- x/y diffusion ratio: 1000
- Linear triangular FEM on unstructured mesh
- Matrix diagonally scaled, RCM reordering

Result for ANI1

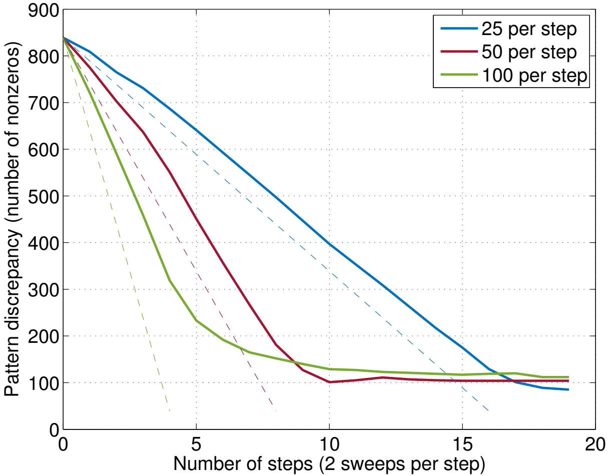


Squares: ICT pattern
Dots: IC(0) pattern



Squares: ICT pattern
Dots: Iterative ICT pattern

Result for ANI3



IC(0)-PCG: 31 iterations

ICT-PCG: 16 iterations

IterICT-PCG: 14,16,16 iterations ($m = 25, 50, 100$)

Sparse triangular solves with ILU factors

Iterative and approximate triangular solves

Trade accuracy for parallelism

Approximately solve the triangular system $Rx = b$

$$x_{k+1} = (I - D^{-1}R)x_k + D^{-1}b$$

where D is the diagonal part of R. In general, $x \approx p(R)b$ for a polynomial $p(R)$.

- implementations depend on SpMV
- iteration matrix $G = I - D^{-1}R$ is strictly triangular and has spectral radius 0 (trivial asymptotic convergence)
- for fast convergence, want the norm of G to be small
- R from stable ILU factorizations of physical problems are often close to being diagonally dominant
- preconditioner is fixed linear operator in non-asynchronous case

Related work

If the triangular factors are scaled to have a unit diagonal, Jacobi method is equivalent to *approximating the ILU factors with a truncated Neumann series*

- van der Vorst 1982
  - Tests on CRAY-1 for regular grid problems
- Benzi and Tuma 1999
  - Tests on CRAY C98 comparing several preconditioners
  - For nonsymmetric problems, ILU with truncated Neumann series approximations were often best, but robustness could be a problem
  - For SPD problems, truncated Neumann series method is not competitive

IC-PCG with exact and iterative triangular solves on Intel Xeon Phi

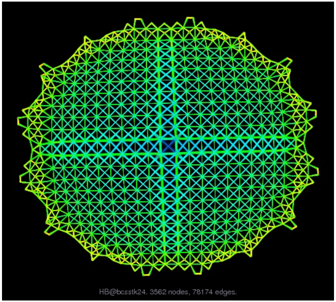
	IC	PCG iterations		Timing (seconds)		Num.
	level	Exact	Iterative	Exact	Iterative	sweeps
af_shell3	1	375	592	79.59	23.05	6
thermal2	0	1860	2540	120.06	48.13	1
ecology2	1	1042	1395	114.58	34.20	4
apache2	0	653	742	24.68	12.98	3
G3_circuit	1	329	627	52.30	32.98	5
offshore	0	341	401	42.70	9.62	5
parabolic_fem	0	984	1201	15.74	16.46	1

Table: Results using 60 threads on Intel Xeon Phi. Exact solves used level scheduling.

Message from Jennifer Scott:

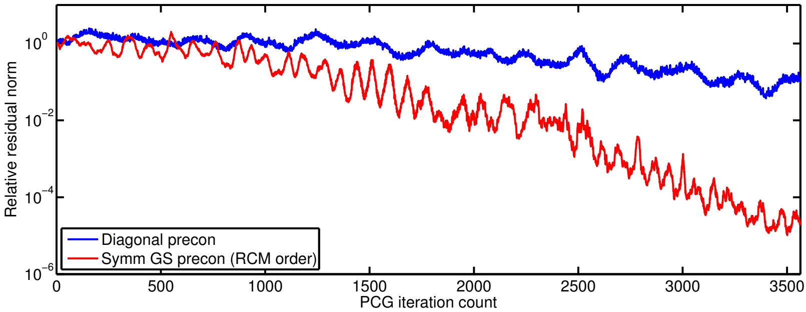
However, for HB/bcsstk24, I find that I need $>$ *200 sweeps for each solve to get CG to converge, with 280 sweeps needed to get the same number of CG iterations as the exact triangular solves.*

BCSSTK24



Saddledome in Calgary, Canada

BCSSTK24



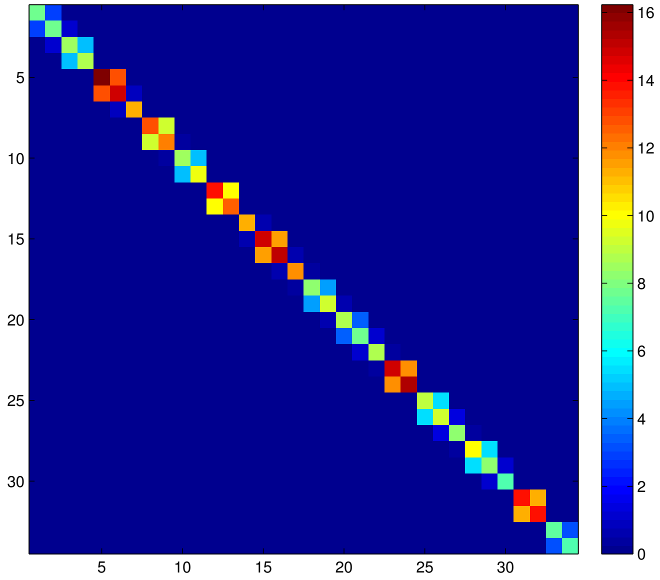
- PCG does not converge with diagonal and Symmetric Gauss–Seidel preconditioning
- Matrix has large off-diagonal entries
- IC(0) does not exist

Block Jacobi for Triangular solves

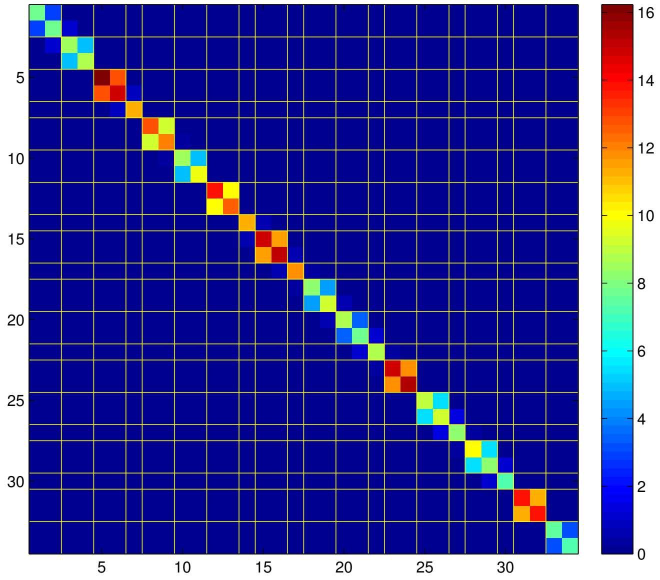
$$x_{k+1} = (I - D^{-1}R)x_k + D^{-1}b$$

- Find a blocking of the matrix
- Reorder the blocking using RCM
- BCSSTK24 almost has 6×6 blocks

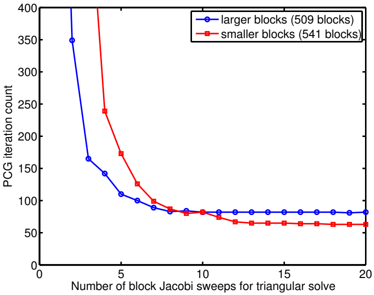
Closeup of BCSSTK24 (each entry is 6 × 6 block)

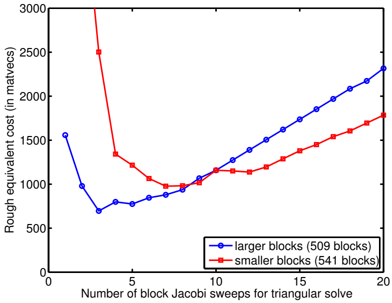


Closeup of BCSSTK24 (each entry is 6 × 6 block)

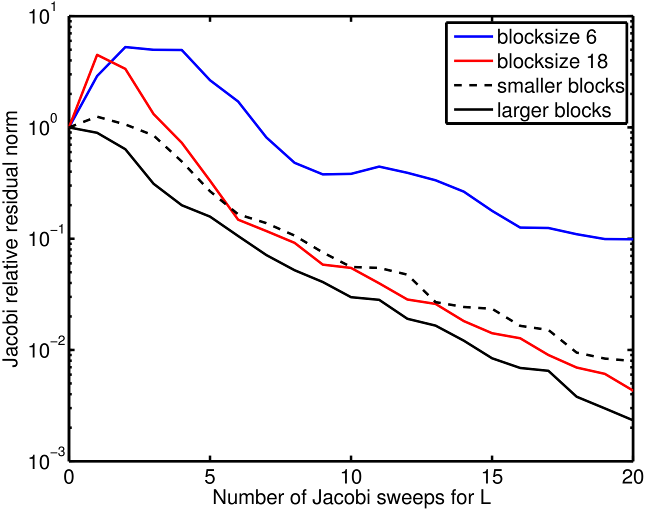


Block Jacobi for Triangular solves





Convergence of iterative solve with L



Comprehensive tests on SPD problems

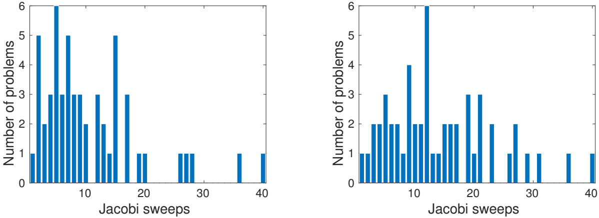
Test all SPD matrices in the University of Florida Sparse Matrix collection with *nrows* $\geq$ 1000 except diagonal matrices: 171 matrices.

Among these, find all problems that can be solved with IC(0) or IC(1).

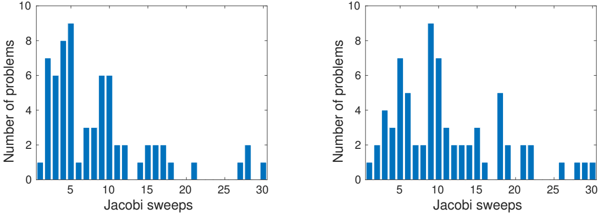
	IC(0)	IC(1)
Total	73	86
Num solved using iter trisol	54 (74%)	52 (60%)
Num solved using block iter trisol	68 (93%)	70 (81%)

Number of Jacobi sweeps for solution in same number of PCG iterations when exact solves are used

Iterative triangular solves, IC(0) and IC(1)



Block iterative triangular solves (max block size 12)



Conclusions

Fine-grained algorithm for ILU factorization

- ILU factorization computed approximately, leading to large amounts of fine-grained parallelism
- ILU factorization can be updated iteratively from an initial guess

Solving sparse triangular systems via iteration

- Will not work for arbitrary triangular matrices
- Block Jacobi variant can reduce required number of sweeps
- Permute large entries into diagonal blocks (if reordering possible)
- Block variant could aid convergence in fine-grained ILU algorithm

An Intel Parallel Computing Center grant is gratefully acknowledged. This material is based upon work supported by the U.S. DOE Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program under Award Number DE-SC-0012538.