Multilevel preconditioning techniques with applications

Yousef Saad

Department of Computer Science and Engineering

University of Minnesota

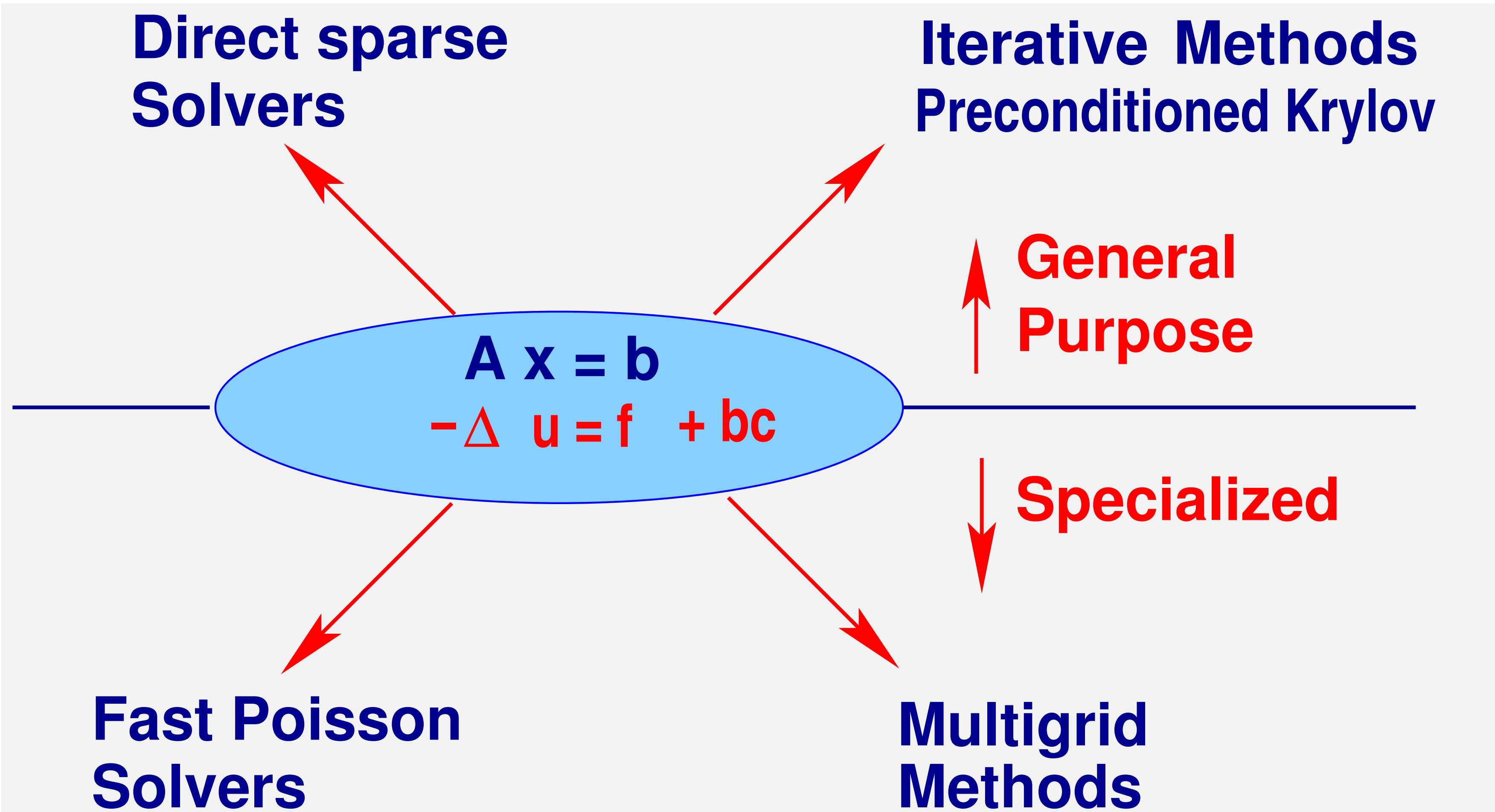
Syracuse University, Sep. 20, 2013

First:

- Joint work with Ruipeng Li

- Work supported by NSF-DMS

Introduction: Linear System Solvers



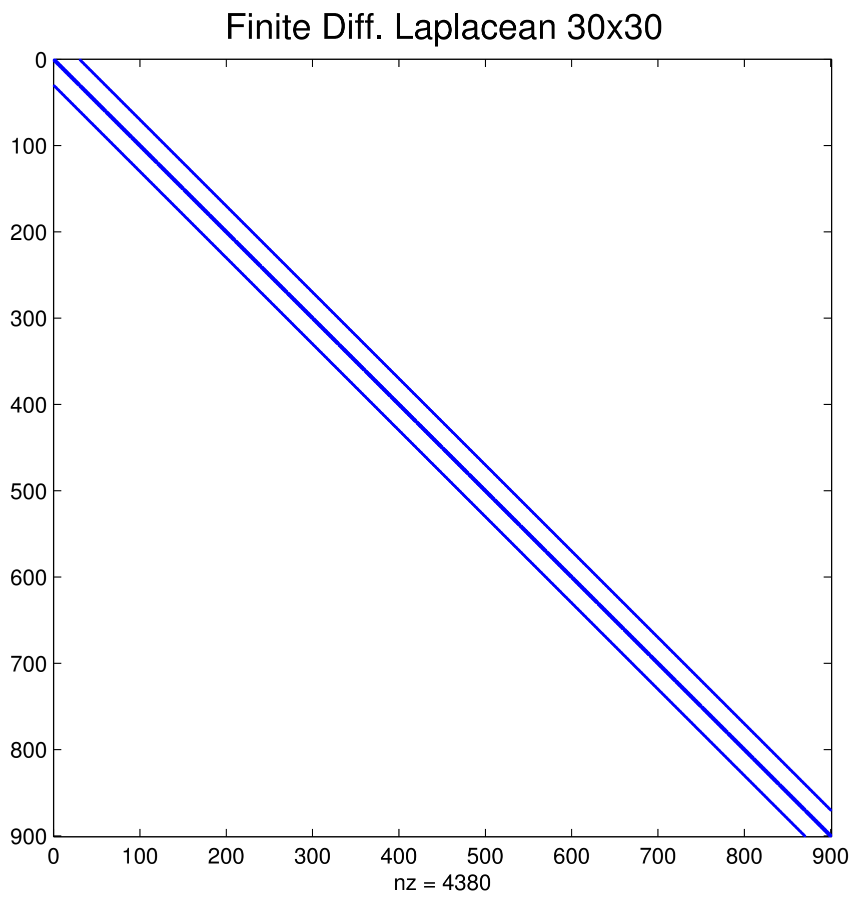
Long standing debate: direct vs. iterative

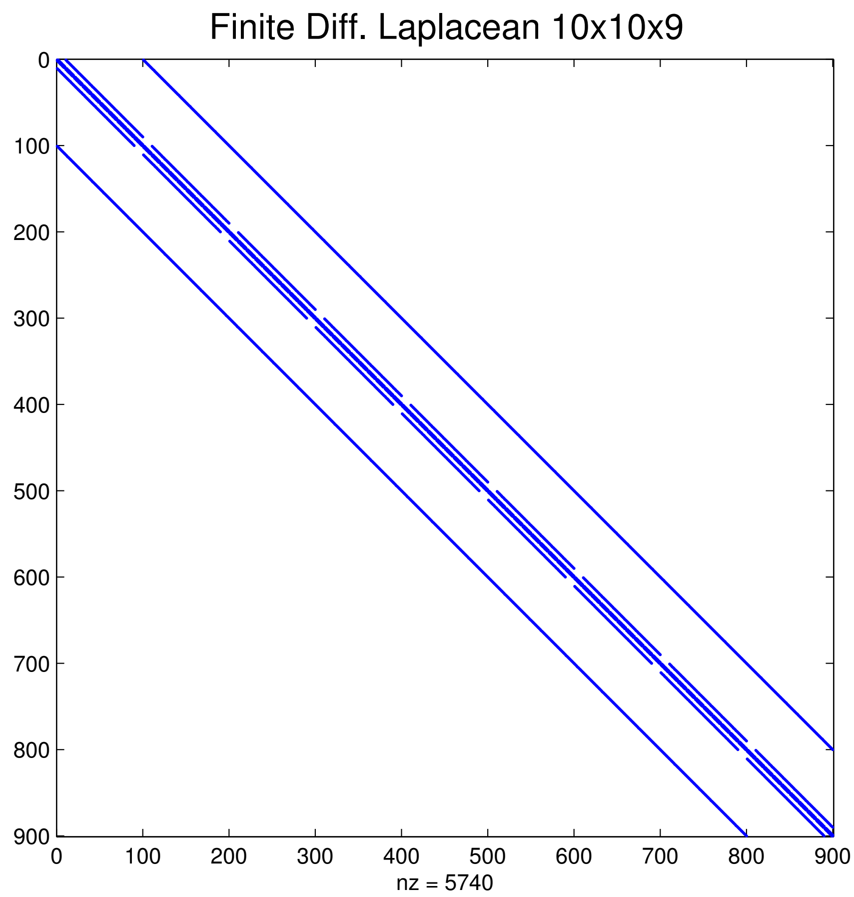
- Starting in the 1970's: huge progress of sparse direct solvers
- Iterative methods - much older - not designed for 'general systems'. Big push in the 1980s with help from 'preconditioning'
- General consensus now: Direct methods do well for 2-D problems and some specific applications [e.g., structures, ...]
- Usually too expensive for realistic 3-D problems
- Huge difference between 2-D and 3-D case
- $\to$ Do the test: Two Laplacean matrices of same dimension $\boldsymbol{n = 122,500}$.

First: on a 350×350 grid (2D);

Second: on a $50 \times 50 \times 49$ grid (3D)

➤ Pattern of similar [much smaller] coefficient matrices





Background: Preconditioned iterative solvers

Two ingredients:

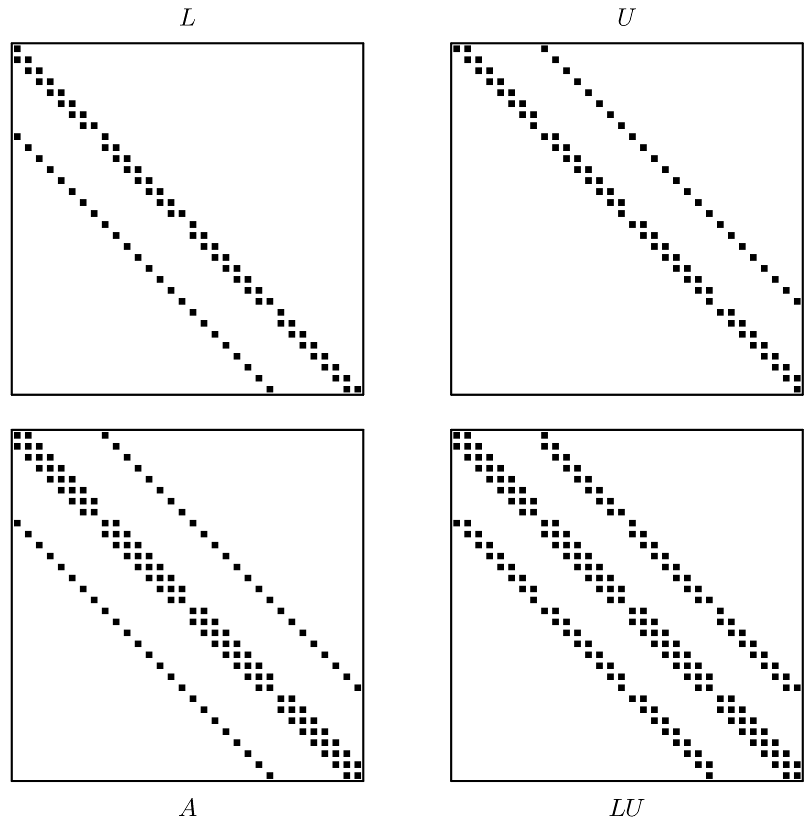
- An accelerator: Conjugate gradient, BiCG, GMRES, BICGSTAB,.. ['Krylov subspace methods']
- A preconditioner: makes the system easier to solve by accelerator, e.g. Incomplete LU factorizations; SOR/SSOR; Multigrid, ...

One viewpoint:

➤ Goal of accelerator: find best combination of basic iterates

➤ Goal of preconditioner: generate good basic iterates.. [Gauss-Seidel, ILU, ...]

Background: Incomplete LU (ILU) preconditioners



ILU: $A \approx LU$

Simplest Example: ILU(0) $\longrightarrow$

Common difficulties of ILUs:

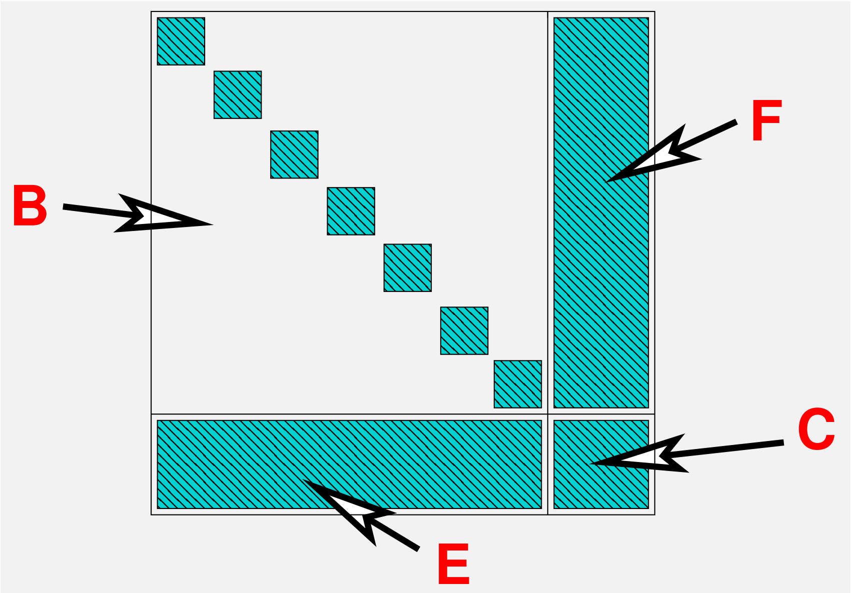
Often fail for indefinite problems

Not too good for highly parallel environments

Past work: Algebraic Recursive Multilevel Solver (ARMS)

- Reorder matrix using 'group-independent sets'. Result

$$PAP^T = \begin{pmatrix} B & F \\ E & C \end{pmatrix} =$$



- Block factorize:

$$\begin{pmatrix} B & F \\ E & C \end{pmatrix} = \begin{pmatrix} L & 0 \\ EU^{-1} & I \end{pmatrix} \begin{pmatrix} U & L^{-1}F \\ 0 & S \end{pmatrix}$$

- $S = C - EB^{-1}F$ = Schur complement + dropping to reduce fill

- Next step: treat the Schur complement recursively

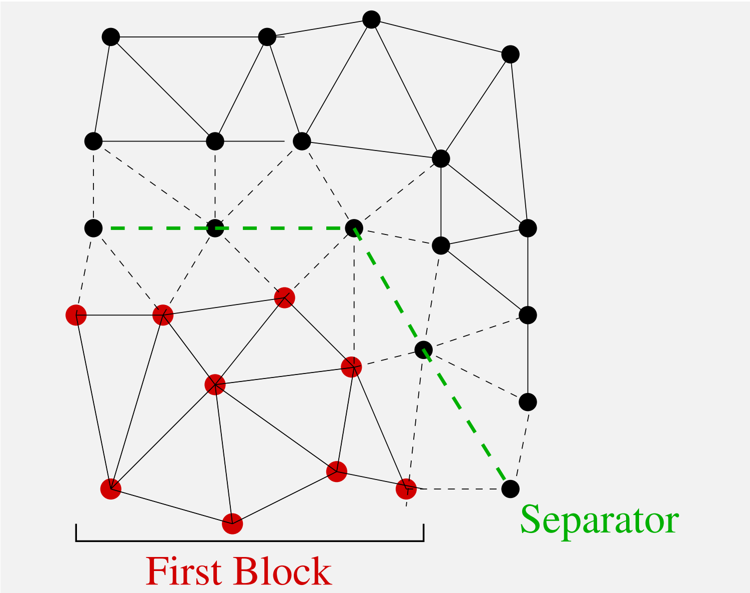
Algebraic Recursive Multilevel Solver (ARMS)

Level l Factorization:

$$\begin{pmatrix} B_l & F_l \\ E_l & C_l \end{pmatrix} \approx \begin{pmatrix} L_l & 0 \\ E_l U_l^{-1} & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & A_{l+1} \end{pmatrix} \begin{pmatrix} U_l & L_l^{-1} F_l \\ 0 & I \end{pmatrix}$$

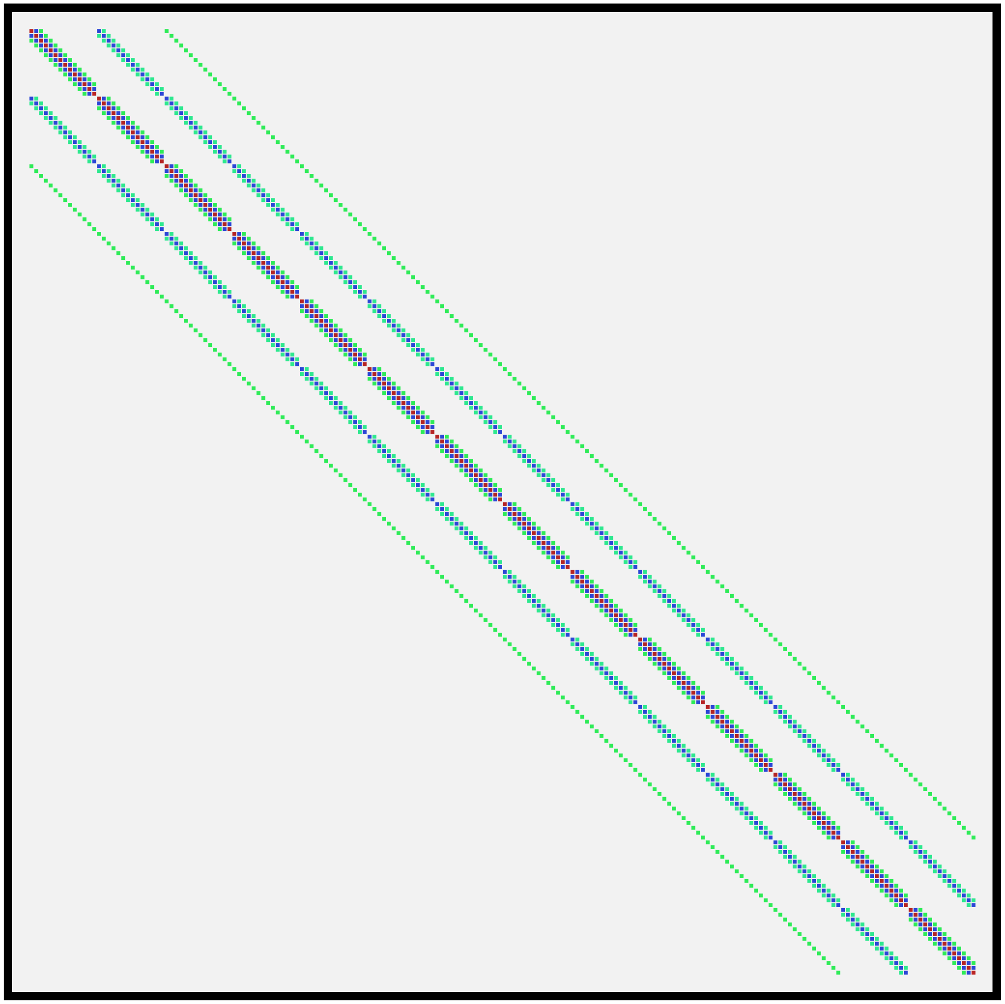
- Perform above block factorization recursively on A_{l+1}
- Blocks in B_l treated as sparse. Can be large or small.
- Algorithm is fully recursive
- L-solve $\sim$ restriction; U-solve $\sim$ prolongation.
- Stability criterion in block independent sets algorithm
- A few similar ideas in the literature: Y. Notay '05, AMLI work (Axelson et al. 2000's), MLILU (Bank Wagner '99), ...

Group Independent Set reordering

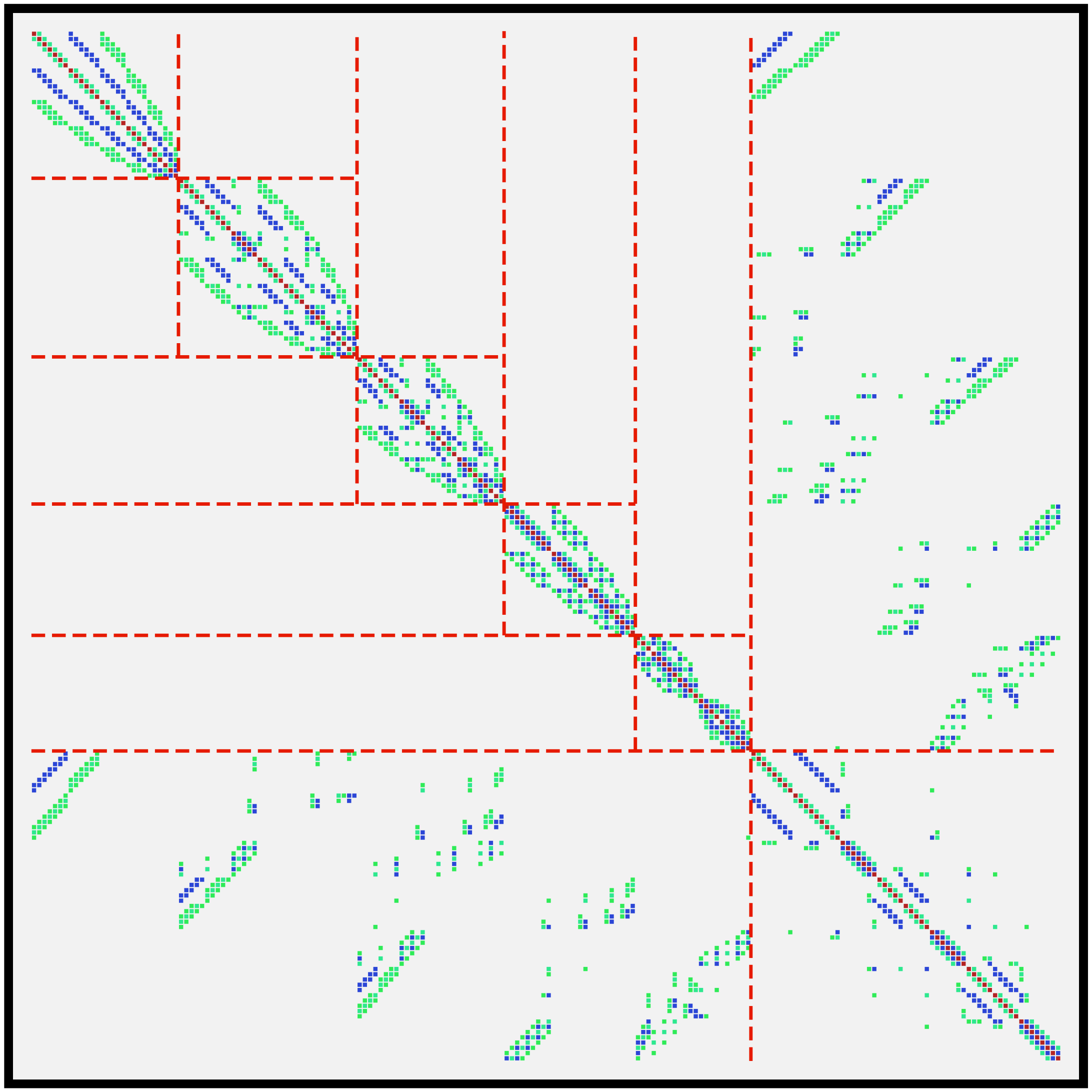


Simple strategy: Level taversal until there are enough points to form a block. Reverse ordering. Start new block from non-visited node. Continue until all points are visited. Add criterion for rejecting "not sufficiently diagonally dominant rows."

Original matrix



Block size of 20



NONSYMMETRIC REORDERINGS

Enhancing robustness: One-sided permutations

➤ Very useful techniques for matrices with extremely poor structure. Not as helpful in other cases.

Previous work:

- Benzi, Haws, Tuma '99 [compare various permutation algorithms in context of ILU]
- Duff '81 [Propose max. transversal algorithms. Basis of many other methods. Also Hopcroft & Karp '73, Duff '88]
- Olchowsky and Neumaier '96 maximize the product of diagonal entries $\longrightarrow$ LP problem
- Duff, Koster, '99 [propose various permutation algorithms. Also discuss preconditioners] Provide MC64

Two-sided permutations with diagonal dominance

Idea: ARMS + exploit nonsymmetric permutations

- No particular structure or assumptions for B block
- Permute rows * and * columns of A. Use two permutations P (rows) and Q (columns) to transform A into

$$PAQ^T = \begin{pmatrix} B & F \\ E & C \end{pmatrix}$$

P, Q is a pair of permutations (rows, columns) selected so that the B block has the 'most diagonally dominant' rows (after nonsym perm) and few nonzero elements (to reduce fill-in).

Multilevel framework

➤ At the l-th level reorder matrix as shown above and then carry out the block factorization 'approximately'

$$P_l A_l Q_l^T = \begin{pmatrix} B_l & F_l \\ E_l & C_l \end{pmatrix} \approx \begin{pmatrix} L_l & 0 \\ E_l U_l^{-1} & I \end{pmatrix} \times \begin{pmatrix} U_l & L_l^{-1} F_l \\ 0 & A_{l+1} \end{pmatrix},$$

where

$$\begin{aligned} B_l &\approx L_l U_l \\ A_{l+1} &\approx C_l - (E_l U_l^{-1})(L_l^{-1} F_l) \, . \end{aligned}$$

➤ As before the matrices $E_l U_l^{-1}, L_l^{-1} F_l$ or their approximations

$$G_l \approx E_l U_l^{-1} \, , \qquad W_l \approx L_l^{-1} F_l$$

need not be saved.

Interpretation in terms of complete pivoting

Rationale: Critical to have an accurate and well-conditioned B block [Bollhöfer, Bollhöfer-YS'04]

➤ Case when B is of dimension 1 $\rightarrow$ a form of complete pivoting ILU. Procedure $\sim$ block complete pivoting ILU

Matching sets: define B block. $\mathcal{M}$ is a set of n_M pairs (p_i, q_i) where $n_M \le n$ with $1 \le p_i, q_i \le n$ for $i = 1, \ldots, n_M$ and

$$p_i \neq p_j, \text{ for } i \neq j \qquad q_i \neq q_j, \text{ for } i \neq j$$

➤ When $n_M = n \rightarrow$ (full) permutation pair (P, Q). A partial matching set can be easily completed into a full pair (P, Q) by a greedy approach.

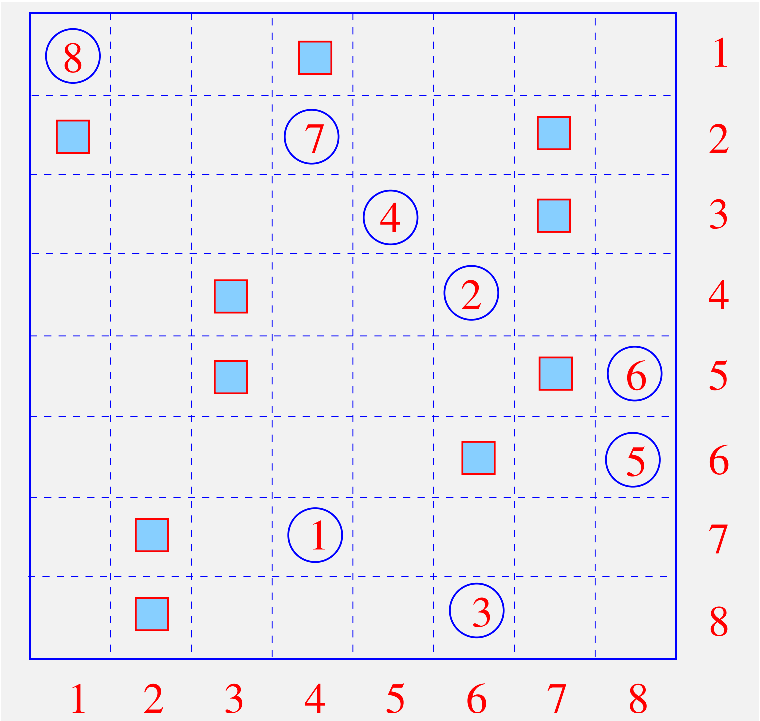
Matching - preselection

Algorithm to find permutation consists of 3 phases.
(1) $\boxed{\text{Preselection:}}$ to filter out poor rows (dd. criterion) and sort the selected rows.
(2) $\boxed{\text{Matching:}}$ scan candidate entries in order given by preselection and accept them into the $\mathcal{M}$ set, or reject them.
(3) $\boxed{\text{Complete the matching set:}}$ into a complete pair of permutations (greedy algorithm)

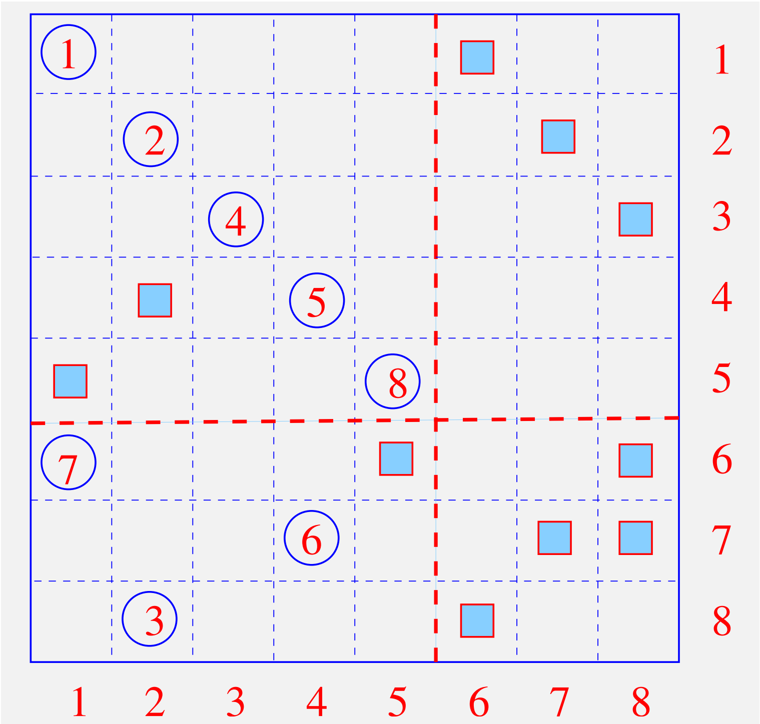
- Let $j(i) = \mathbf{argmax}_j |a_{ij}|$.
- Use the ratio $\gamma_i = \frac{|a_{i,j(i)}|}{\|a_{i,:}\|_1}$ as a measure of diag. domin. of row i

Matching: Greedy algorithm

- Simple algorithm: scan pairs (i_k, j_k) in the given order.
- If i_k and j_k not already assigned, assign them to $\mathcal{M}$.



Matrix after preselection



Matrix after Matching perm.

COMPLEX SHIFTING

Use of complex shifts

➤ Several papers promoted the use of complex shifts [or very similar approaches] for Helmholtz

[1] X. Antoine – Private comm.

[2] Y.A. Erlangga, C.W. Oosterlee and C. Vuik, SIAM J. Sci. Comput.,27, pp. 1471-1492, 2006

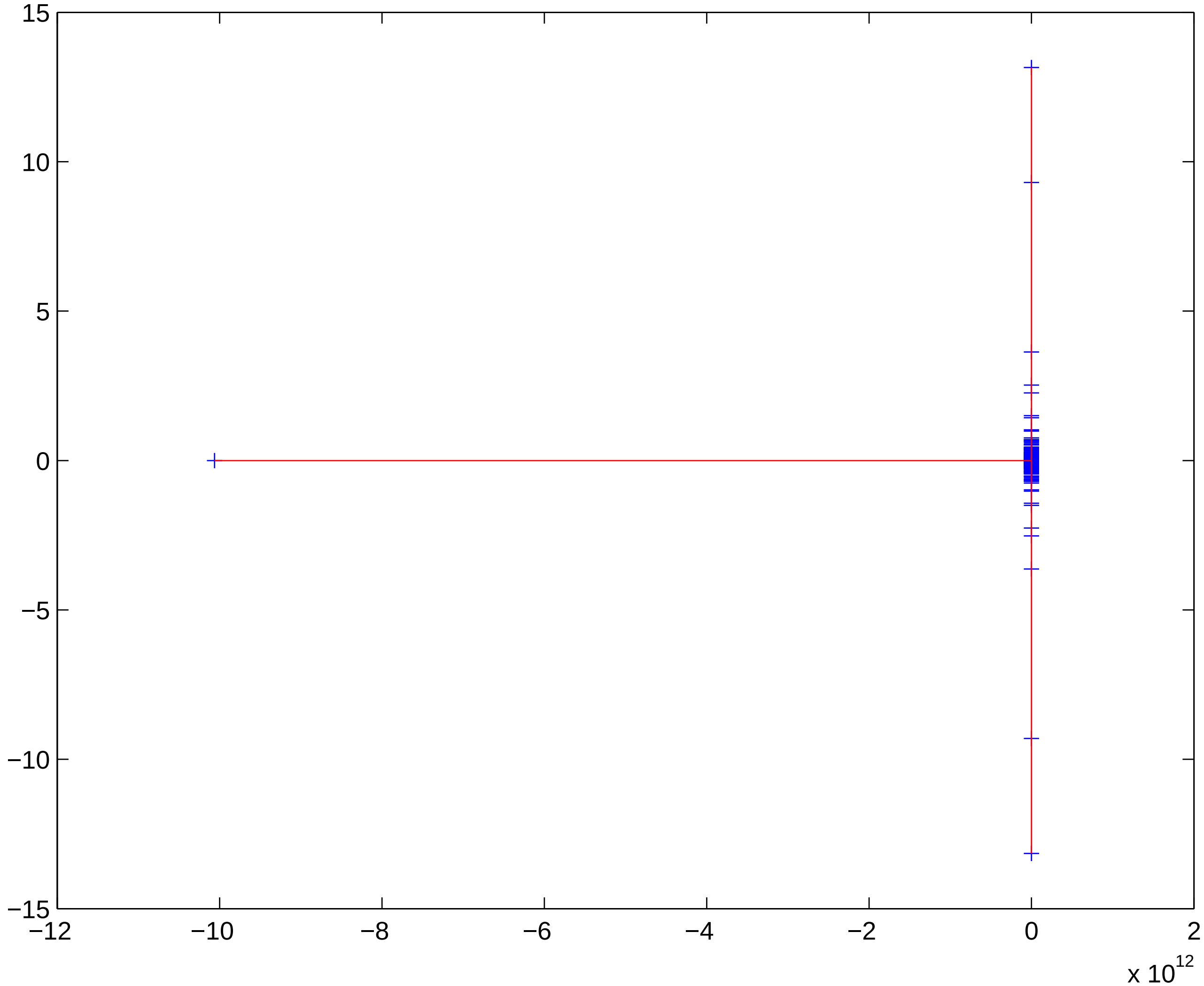
[3] M. B. van Gijzen, Y. A. Erlangga, and C. Vuik, SIAM J. Sci. Comput., Vol. 29, pp. 1942-1958, 2007

[4] M. Magolu Monga Made, R. Beauwens, and G. Warzée, Comm. in Numer. Meth. in Engin., 16(11) (2000), pp. 801-817.

** Joint work with Daniel Osei-Kuffuor

- Illustration with an experiment: finite difference discretization of $-\boldsymbol{\Delta}$ on a $\mathbf{25} \times \mathbf{20}$ grid.
- Add a negative shift of $-\mathbf{1}$ to resulting matrix.
- Do an ILU factorization of $\boldsymbol{A}$ and plot eigs of $\boldsymbol{L}^{-1}\boldsymbol{A}\boldsymbol{U}^{-1}$.
- Used LUINC from matlab - no-pivoting and threshold = 0.1.

➤ Terrible spectrum:

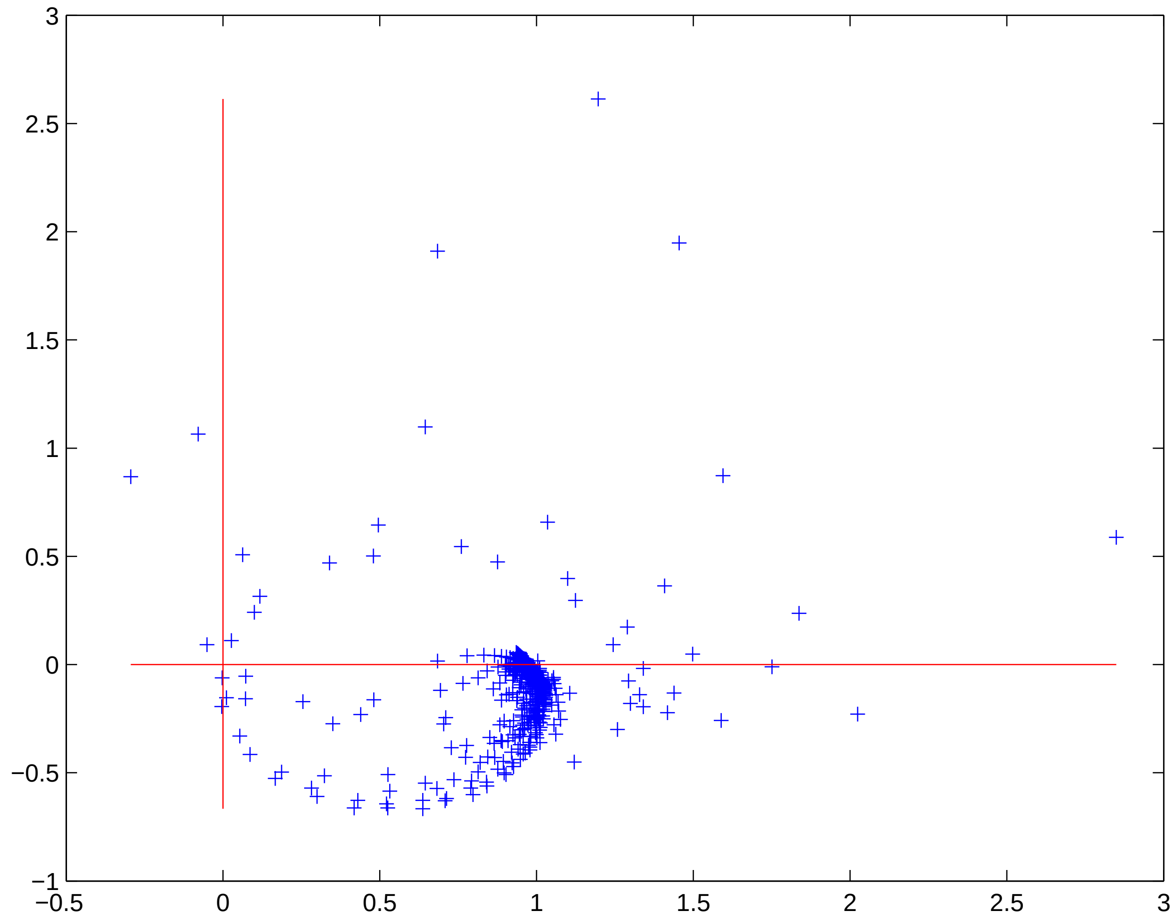


➤ Now plot eigs of $L^{-1}AU^{-1}$ where L, U are inc. LU factors of $B = A + 0.25 * i$

➤ Much better! Observed by many [PDE viewpoint]

Idea:

Adapt technique to ILU:
Add complex shifts before ILU

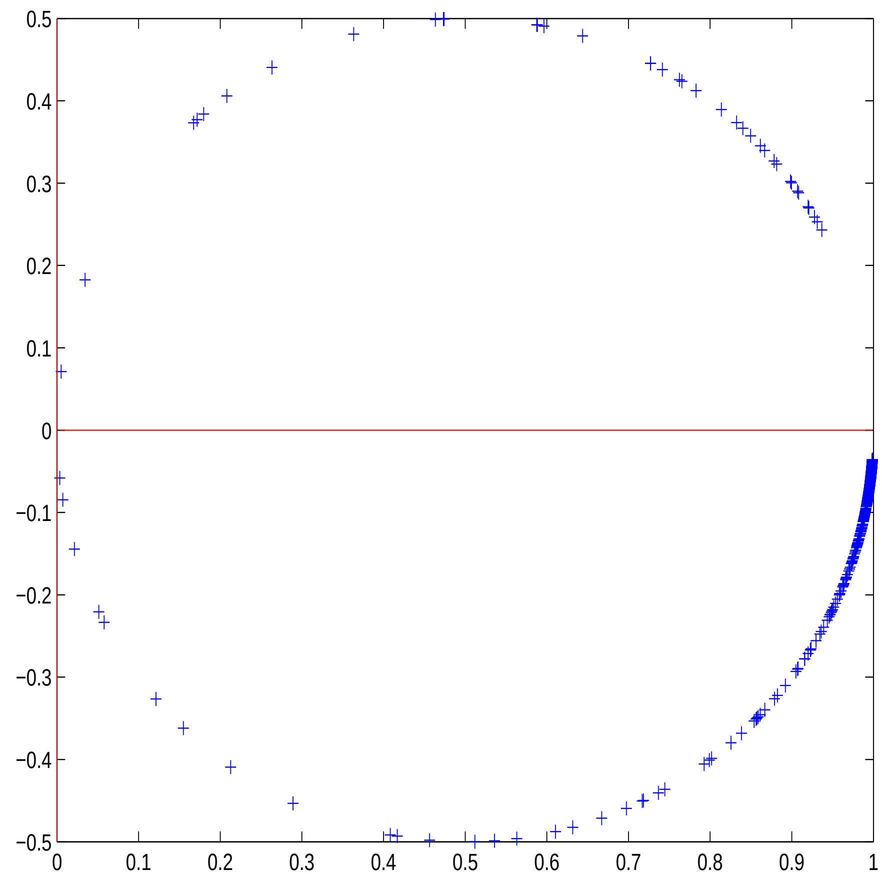


Explanation

Question:

What if we do an exact factorization [droptol = 0]?

- $\Lambda(L^{-1}AU^{-1}) = \Lambda[(A+\alpha iI)^{-1}A]$
- $\Lambda = \left\{\frac{\lambda_j}{\lambda_j + i\alpha}\right\}$
- Located on a circle – with a cluster at one.
- Figure shows situation on the same example



Application to the Helmholtz equation

➤ Started from collaboration with Riyad Kechroud, Azzeddine Soulaimani (ETS, Montreal), and Shiv Gowda: [Math. Comput. Simul., vol. 65., pp 303–321 (2004)]

➤ Problem is set in the open domain Ω_e of $\mathbb{R}^d$

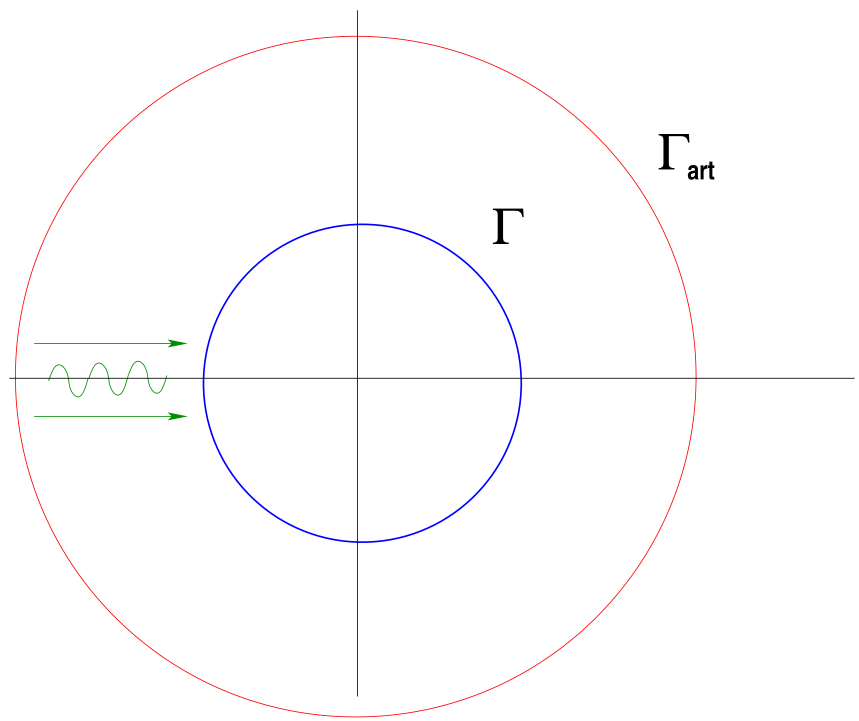
$$\begin{cases} \Delta u + k^2 u = f & \text{in } \Omega \\ u = -u_{inc} & \text{on } \Gamma \\ or \ \frac{\partial u}{\partial n} = -\frac{\partial u_{inc}}{\partial n} & \text{on } \Gamma \\ \\ lim_{r \to \infty} r^{(d-1)/2} \left(\frac{\partial u}{\partial \vec{n}} - iku \right) = 0 & \text{Sommerfeld cond.} \end{cases}$$

where: u the wave diffracted by Γ, f = source function = zero outside domain

- Issue: non-reflective boundary conditions when making the domain finite.
- Artificial boundary Γ_{art} added – Need non-absorbing BCs.
- For high frequencies, linear systems become very ‘indefinite’ – [eigenvalues on both sides of the imaginary axis]
- Not very good for iterative methods

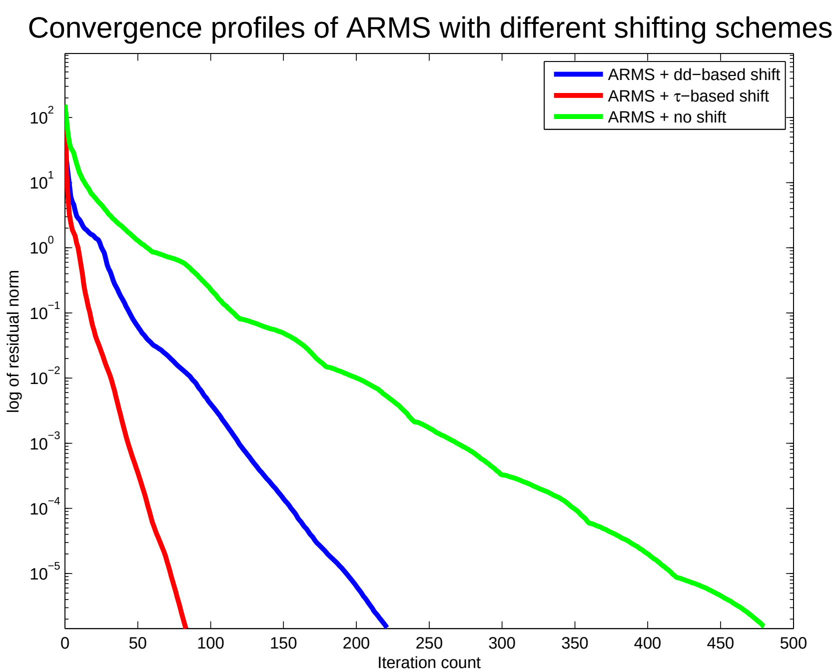
Application to the Helmholtz equation

Test Problem Soft obstacle = disk of radius $r_0 = 0.5m$. Incident plane wave with a wavelength λ; propagates along the x-axis. 2nd order Bayliss-Turkel boundary conditions used on Γ_{art}, located at a distance $2r_0$ from obstacle. Discretization: isoparametric elements with 4 nodes. Analytic solution known.

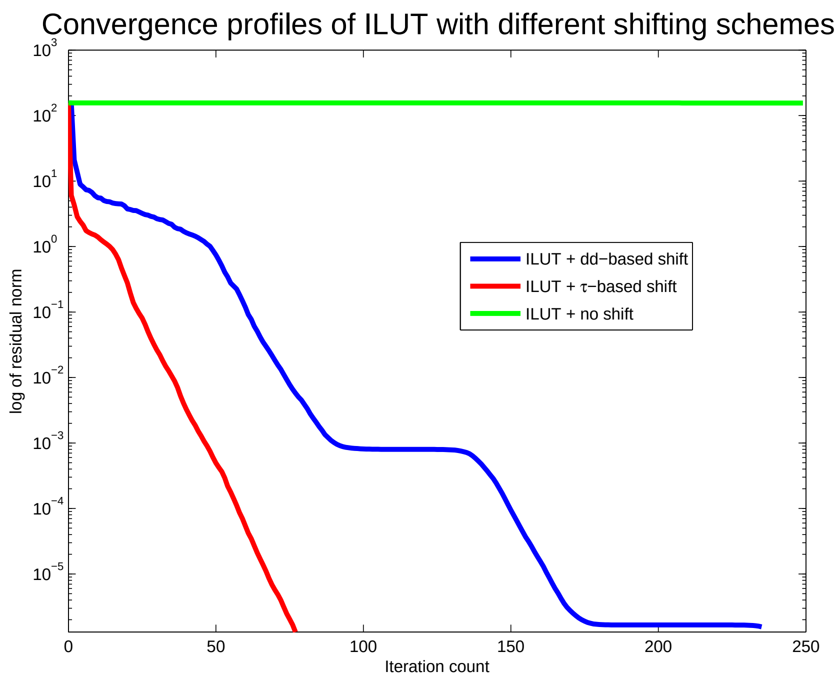


Comparisons

➤ Test problem just seen. Mesh size $1/h = 160 \to n = 28,980$, $nnz = 260,280$



ARMS & shifted variants



ILUT & shifted variants

➤ Wavenumber varied - tests with ILUT

Preconditioner	k	$\frac{\lambda}{h}$	**Iters.**	**Fill Factor**	$\|\|(LU)^{-1}e\|\|_2$
ILUT (no shift)	4π	60	134	2.32	$3.65e+03$
	8π	30	263	2.25	1.23e+04
	16π	15	$-$	-	-
	24π	10	$-$	-	-
ILUT (dd-based)	4π	60	267	2.24	$2.29e+03$
	8π	30	255	2.23	4.73e+03
	16π	15	101	3.14	6.60e+02
	24π	10	100	3.92	2.89e+02
ILUT (τ-based)	4π	60	132	2.31	$2.98e+03$
	8π	30	195	2.19	4.12e+03
	16π	15	75	3.11	7.46e+02
	24π	10	86	3.85	2.73e+02

➤ Wavenumber varied - tests with ARMS

Preconditioner	k	$\frac{\lambda}{h}$	Iters.	Fill Factor	$\|\|(LU)^{-1}e\|\|_2$
ARMS (no shift)	4π	60	120	3.50	$7.48e+03$
	8π	30	169	4.03	1.66e+04
	16π	15	282	4.50	2.44e+03
	24π	10	—	-	-
ARMS (dd-based)	4π	60	411	3.83	$5.12e+02$
	8π	30	311	4.37	5.67e+02
	16π	15	187	4.71	3.92e+02
	24π	10	185	3.00	2.54e+02
ARMS (τ-based)	4π	60	106	3.45	$7.56e+03$
	8π	30	79	3.84	6.41e+03
	16π	15	39	3.95	1.26e+03
	24π	10	94	3.02	4.71e+02

SPARSE MATRIX COMPUTATIONS ON GPUS

Sparse matrix computations with GPUs **

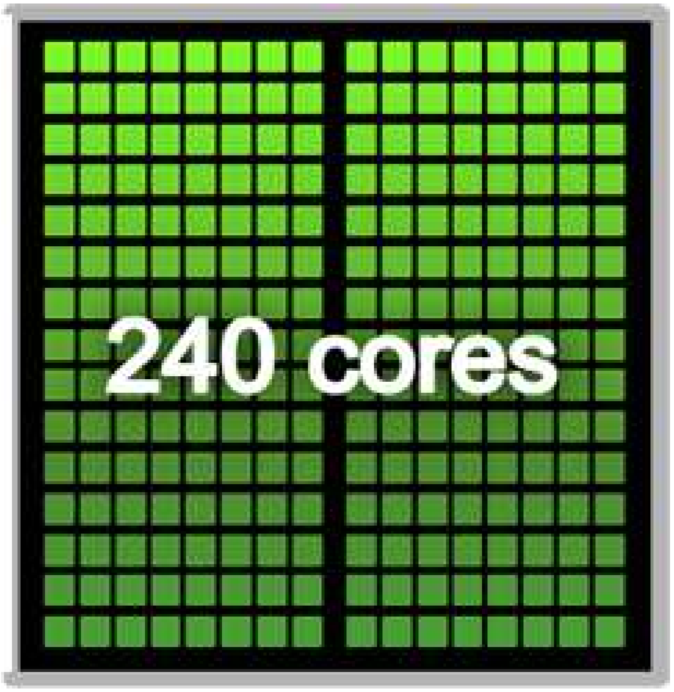
- GPUs Currently a very popular approach to: inexpensive supercomputing

- Can buy ∼ one Teraflop peak power for around a little more tham $1,000

Tesla C1060

** Joint work with Ruipeng Li

Tesla C 1060:



* 240 cores
* 4 GB memory
* Peak rate: 930 GFLOPS [single]
* Clock rate: 1.3 Ghz
* 'Compute Capability': 1.3 [allows double precision]

➤ Next: Fermi [48 cores/SM]— followed by [very recently]:

➤ Kepler [note: 6 GHz (!), 192 cores/SMX, 4 SMXs in a GPC]

➤ Tesla K10 : $2 \times$ (8 SMXs) $\rightarrow 2 \times$ 1,536 cores, 8GB Mem.; Peak: $\approx$ 4.6 TFLOPS]

The CUDA environment: The big picture

- A host (CPU) and an attached device (GPU)

Typical program:

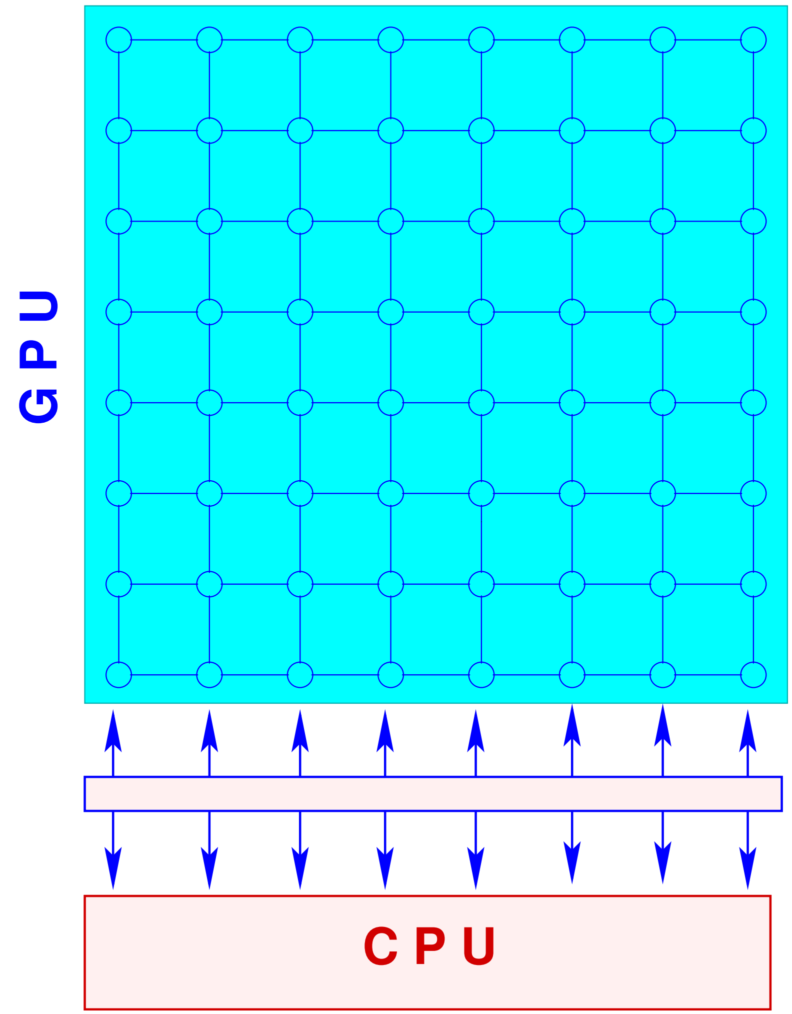
1. Generate data on CPU
2. Allocate memory on GPU

   `cudaMalloc(...)`
3. Send data Host $\rightarrow$ GPU

   `cudaMemcpy(...)`
4. Execute GPU 'kernel':

   `kernel <<<(...)>>>(..)`
5. Copy data GPU $\rightarrow$CPU

   `cudaMemcpy(...)`



Sparse matrix computations on GPUs

Main issue in using GPUs for sparse computations:

- Huge performance degradation due to 'irregular sparsity'

➤ Matrices:

Matrix -name	N	NNZ
FEM/Cantilever	62,451	4,007,383
Boeing/pwtk	217,918	11,634,424

➤ Performance of Mat-Vecs on NVIDIA Tesla C1060

	Single Precision			*Double Precision*		
Matrix	CSR	JAD	DIA	CSR	JAD	DIA
FEM/Cantilever	9.4	10.8	25.7	7.5	5.0	13.4
Boeing/pwtk	8.9	16.6	29.5	7.2	10.4	14.5

Sparse Forward/Backward Sweeps

- Next major ingredient of precond. Krylov subs. methods

- ILU preconditioning operations require L/U solves: $x \leftarrow U^{-1}L^{-1}x$
- Sequential outer loop.

```
for i=1:n
    for j=ia(i):ia(i+1)
            x(i) = x(i) - a(j)*x(ja(j))
    end
end
```

- Parallelism can be achieved with level scheduling:

- Group unknowns into levels
- Compute unknowns $x(i)$ of same level simultaneously
- $1 \le nlev \le n$

ILU: Sparse Forward/Backward Sweeps

- Very poor performance [relative to CPU]

Matrix	N	CPU	GPU-Lev	
		Mflops	#lev	Mflops
Boeing/bcsstk36	23,052	627	4,457	43
FEM/Cantilever	62,451	653	2,397	168
COP/CASEYK	696,665	394	273	142
COP/CASEKU	208,340	373	272	115

Prec: miserable :-)

GPU Sparse Triangular Solve with Level Scheduling

➤ Very poor performance when #levs is large

➤ A few things can be done to reduce the # levels but perf. will remain poor

So...

Either GPUs must go...

or ILUs must go...

Alternatives to ILU preconditioners

➤ What would be a good alternative?

Wish-list:

- A preconditioner requiring few 'irregular' computations
- Something that trades volume of computations for speed
- If possible something that is robust for indefinite case

➤ Good candidate:

- Multilevel Low-Rank (MLR) approximate inverse preconditioners

Related work:

- Work on HSS matrices [e.g., JIANLIN XIA, SHIVKUMAR CHANDRASEKARAN, MING GU, AND XIAOYE S. LI, *Fast algorithms for hierarchically semiseparable matrices*, Numerical Linear Algebra with Applications, 17 (2010), pp. 953–976.]
- Work on H-matrices [Hackbusch, ...]
- Work on 'balanced incomplete factorizations' (R. Bru et al.)
- Work on "sweeping preconditioners" by Engquist and Ying.
- Work on computing the diagonal of a matrix inverse [Jok Tang and YS (2010) ..]

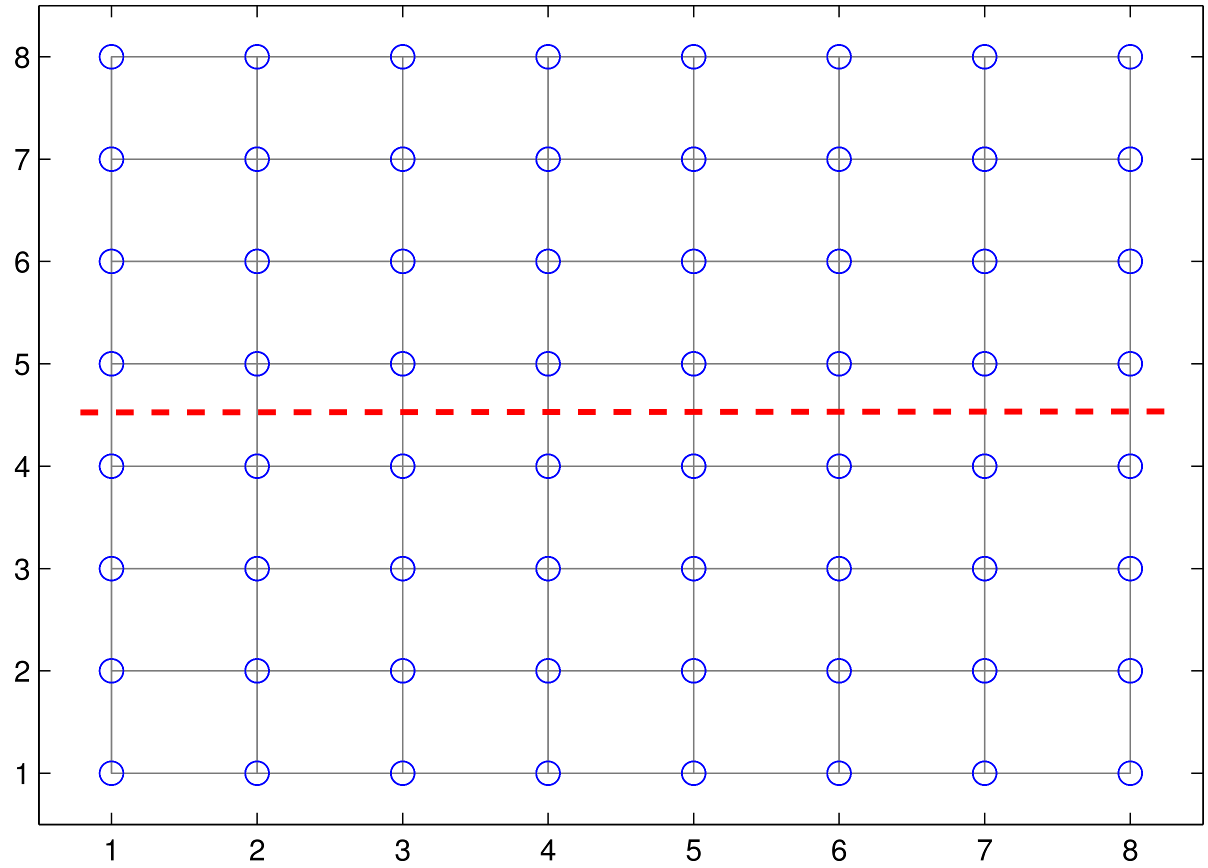
Low-rank Multilevel Approximations

➤ Starting point: symmetric matrix derived from a 5-point discretization of a 2-D Pb on $n_x \times n_y$ grid

$$A = \left(\begin{array}{cccc|ccc} A_1 & D_2 & & & & & \\ D_2 & A_2 & D_3 & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & D_\alpha & A_\alpha & D_{\alpha+1} & & \\ \hline & & & D_{\alpha+1} & A_{\alpha+1} & \ddots & \\ & & & & \ddots & \ddots & \ddots \\ & & & & & D_{n_y} & A_{n_y} \end{array}\right)$$

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \equiv \begin{pmatrix} A_{11} & \\ & A_{22} \end{pmatrix} + \begin{pmatrix} & A_{12} \\ A_{21} & \end{pmatrix}$$

Corresponding splitting on FD mesh:



- $A_{11} \in \mathbb{R}^{m\times m}$, $A_{22} \in \mathbb{R}^{(n-m)\times(n-m)}$

- In the simplest case second matrix is:

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & \\ & A_{22} \end{pmatrix} + \left[\begin{array}{c|c} & -I \\ \hline -I & \end{array}\right]$$

- Write 2nd matrix as:

$$\left[\begin{array}{c|c} & -I \\ \hline -I & \end{array}\right] = \left[\begin{array}{c|c} +I & \\ \hline & +I \end{array}\right] - \underbrace{\left[\begin{array}{c|c} I & I \\ \hline I & I \end{array}\right]}_{E\,E^T}$$

$$E^T = \left[\begin{array}{c|c} I & I \end{array}\right]$$

- Above splitting can be rewritten as

$$A = \underbrace{(A + EE^T)}_{B} - EE^T$$

$$A = B - EE^T,$$

$$B := \begin{pmatrix} B_1 & \\ & B_2 \end{pmatrix} \in \mathbb{R}^{n \times n}, \quad E := \begin{pmatrix} E_1 \\ E_2 \end{pmatrix} \in \mathbb{R}^{n \times n_x},$$

Note: $B_1 := A_{11} + E_1 E_1^T, \quad B_2 := A_{22} + E_2 E_2^T.$

- Shermann-Morrison formula:

$$A^{-1} = B^{-1} + B^{-1}E(\overbrace{I - E^T B^{-1} E}^{X})^{-1} E^T B^{-1}$$

$$\begin{aligned} A^{-1} &= B^{-1} + (B^{-1}E)X^{-1}(B^{-1}E)^T \\ X &= I - E^T B^{-1} E \end{aligned}$$

- Note: $E \in \mathbb{R}^{n \times n_x}$, $X \in \mathbb{R}^{n_x \times n_x}$
- n_x = number of points in separator [$O(n^{1/2})$ for 2-D mesh, $O(n^{2/3})$ for 3-D mesh]

- Use in a recursive framework
- Similar idea was used for computing the diagonal of the inverse [J. Tang YS '10]

- First thought : approximate X and exploit recursivity

$$B^{-1}[v + E\tilde{X}^{-1}E^T B^{-1}v].$$

- However wont work : cost explodes with # levels

- Alternative: low-rank approx. for $B^{-1}E$ $\boxed{B^{-1}E \approx U_k V_k^T,}$ $U_k \in \mathbb{R}^{n\times k}$, $V_k \in \mathbb{R}^{n_x \times k}$,

Multilevel Low-Rank (MLR) algorithm

- Method: Use low-rank approx. for $B^{-1}E$ $\boxed{B^{-1}E \approx U_k V_k^T,}$ $U_k \in \mathbb{R}^{n\times k}$, $V_k \in \mathbb{R}^{n_x\times k}$,

- Replace $B^{-1}E$ by $U_k V_k^T$ in $X = I - (E^T B^{-1})E$:

  $X \approx G_k = I - V_k U_k^T E, \quad (\in \mathbb{R}^{n_x\times n_x})$ Leads to ...

Preconditioner

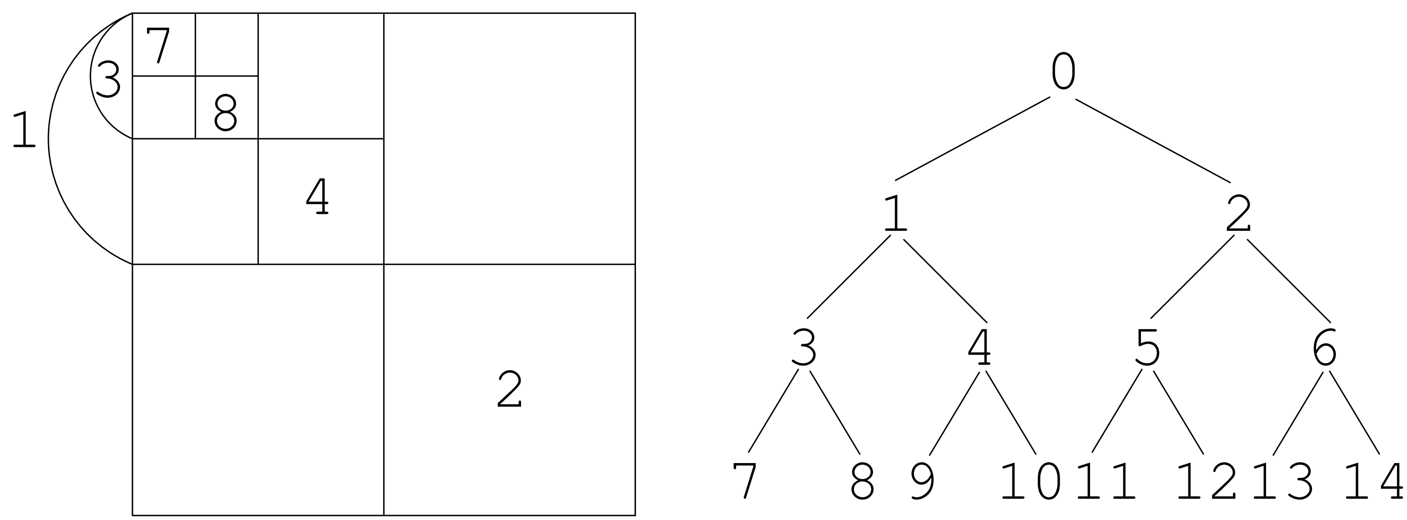
$$M^{-1} = B^{-1} + U_k H_k U_k^T, \quad H_k = V_k^T G_k^{-1} V_k$$

Use recursivity

- We can show : $H_k = (I - U_k^T E V_k)^{-1}$ and $H_k^T = H_k$

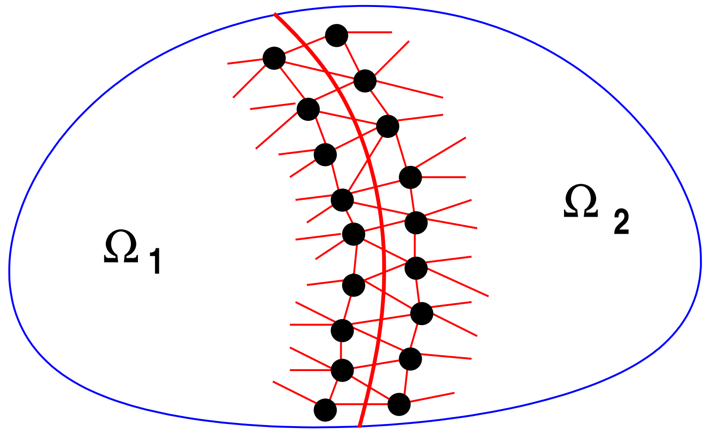
Recursive multilevel framework

- $A_i = B_i + E_i E_i^T$, $B_i \equiv \begin{pmatrix} B_{i_1} & \\ & B_{i_2} \end{pmatrix}$.
- Next level, set $A_{i_1} \equiv B_{i_1}$ and $A_{i_2} \equiv B_{i_2}$
- Repeat on A_{i_1}, A_{i_2}
- Last level, factor A_i (IC, ILU)
- Binary tree structure:



Generalization: Domain Decomposition framework

Domain partitioned into 2 domains with an edge separator



➤ Matrix can be permuted to:

$$PAP^T = \left(\begin{array}{cc|cc} \hat{B}_1 & \hat{F}_1 & & \\ \hat{F}_1^T & C_1 & & -X \\ \hline & & \hat{B}_2 & \hat{F}_2 \\ & -X^T & \hat{F}_2^T & C_2 \end{array}\right)$$

➤ Interface nodes in each domain are listed last.

- Each matrix $\hat{B}_i$ is of size $n_i \times n_i$ (interior var.) and the matrix C_i is of size $m_i \times m_i$ (interface var.)

$$\text{Let:} \qquad E_\alpha = \begin{pmatrix} 0 \\ \alpha I \\ 0 \\ \frac{X^T}{\alpha} \end{pmatrix} \qquad \text{then we have:}$$

$$PAP^T = \begin{pmatrix} B_1 & \\ & B_2 \end{pmatrix} - EE^T \quad \text{with} \quad B_i = \begin{pmatrix} \hat{B}_i & \hat{F}_1 \\ \hat{F}_i^T & C_i + D_i \end{pmatrix}$$

$$\text{and} \quad \begin{cases} D_1 = \alpha^2 I \\ D_2 = \frac{1}{\alpha^2} X^T X \end{cases} .$$

- α used for balancing

- Better results when using diagonals instead of αI

EXPERIMENTS

Experimental setting

- Hardware: Intel Xeon X5675 processor (12 MB Cache, 3.06 GHz, 6-core)
- C/C++; Intel Math Kernel Library (MKL,version 10.2)
- Stop when: $\|r_i\| \le 10^{-8}\|r_0\|$ or `its` exceeds 500
- Model Problems in 2-D/3-D:

$$-\Delta u - cu = g \text{ in } \Omega \quad + \text{B.C.}$$

- 2-D: $g(x, y) = -\left(x^2 + y^2 + c\right) e^{xy}; \quad \Omega = (0, 1)^3.$
- 3-D: $g(x, y, z) = -6 - c\left(x^2 + y^2 + z^2\right); \quad \Omega = (0, 1)^3.$
- F.D. Differences discret.

Symmetric indefinite cases

- $c > 0$ in $-\Delta u - cu$; i.e., $-\Delta$ shifted by $-sI$.
- 2D case: $s = 0.01$, 3D case: $s = 0.05$
- MLR + GMRES(40) compared to ILDLT + GMRES(40)
- 2-D problems: #lev$= 4$, rank$= 5, 7, 7$
- 3-D problems: #lev$= 5$, rank$= 5, 7, 7$
- ILDLT failed for most cases
- Difficulties in MLR: #lev cannot be large, [no convergence]
- inefficient factorization at the last level (memory, CPU time)

Grid	ILDLT-GMRES				MLR-GMRES			
	fill	p-t	its	i-t	fill	p-t	its	i-t
256^2	6.5	0.16	F		6.0	0.39	84	0.30
512^2	8.4	1.25	F		8.2	2.24	246	6.03
1024^2	10.3	10.09	F		9.0	15.05	F	
$32^2 \times 64$	5.6	0.25	61	0.38	5.4	0.98	62	0.22
64^3	7.0	1.33	F		6.6	6.43	224	5.43
128^3	8.8	15.35	F		6.5	28.08	F	

General symmetric matrices - Test matrices

MATRIX	N	NNZ	SPD	DESCRIPTION
Andrews/Andrews	60,000	760,154	yes	computer graphics pb.
Williams/cant	62,451	4,007,383	yes	FEM cantilever
UTEP/Dubcova2	65,025	1,030,225	yes	2-D/3-D PDE pb.
Rothberg/cfd1	70,656	1,825,580	yes	CFD pb.
Schmid/thermal1	82,654	574,458	yes	thermal pb.
Rothberg/cfd2	123,440	3,085,406	yes	CFD pb.
Schmid/thermal2	1,228,045	8,580,313	yes	thermal pb.
Cote/vibrobox	12,328	301,700	no	vibroacoustic pb.
Cunningham/qa8fk	66,127	1,660,579	no	3-D acoustics pb.
Koutsovasilis/F2	71,505	5,294,285	no	structural pb.

Generalization of MLR via DD

- DD: `PartGraphRecursive` from METIS
- balancing with diagonals
- higher ranks used in two problems (`cant` and `vibrobox`)
- Show SPD cases first then non-SPD

MATRIX	ICT/ILDLT				MLR-CG/GMRES					
	fill	p-t	its	i-t	k	lev	fill	p-t	its	i-t
Andrews	2.6	0.44	32	0.16	2	6	2.3	1.38	27	0.08
cant	4.3	2.47	F	19.01	10	5	4.3	7.89	253	5.30
Dubcova2	1.4	0.14	42	0.21	4	4	1.5	0.60	47	0.09
cfd1	2.8	0.56	314	3.42	5	5	2.3	3.61	244	1.45
thermal1	3.1	0.15	108	0.51	2	5	3.2	0.69	109	0.33
cfd2	3.6	1.14	F	12.27	5	4	3.1	4.70	312	4.70
thermal2	5.3	4.11	148	20.45	5	5	5.4	15.15	178	14.96

MATRIX	ICT/ILDLT				MLR-CG/GMRES					
	fill	p-t	its	i-t	k	lev	fill	p-t	its	i-t
vibrobox	3.3	0.19	F	1.06	10	4	3.0	0.45	183	0.22
qa8fk	1.8	0.58	56	0.60	2	8	1.6	2.33	75	0.36
F2	2.3	1.37	F	13.94	5	5	2.5	4.17	371	7.29

Conclusion

- General rule: ILU-based preconditioners not meant to replace tailored preconditioners. Can be very useful as parts of other techniques.

- Robustness can be improved with nonsymmetric permutations and the inclusion of complex shifting strategies

- GPUs for irregular sparse matrix computations: Much remains to be done both in hardware and in algorithms/software. In general, some of the old methods will see a come-back

- More interestingly: new methods such as low-rank approximation methods will be developed