UNIVERSITY OF MINNESOTA TWIN CITIES

Multilevel preconditioning techniques with applications

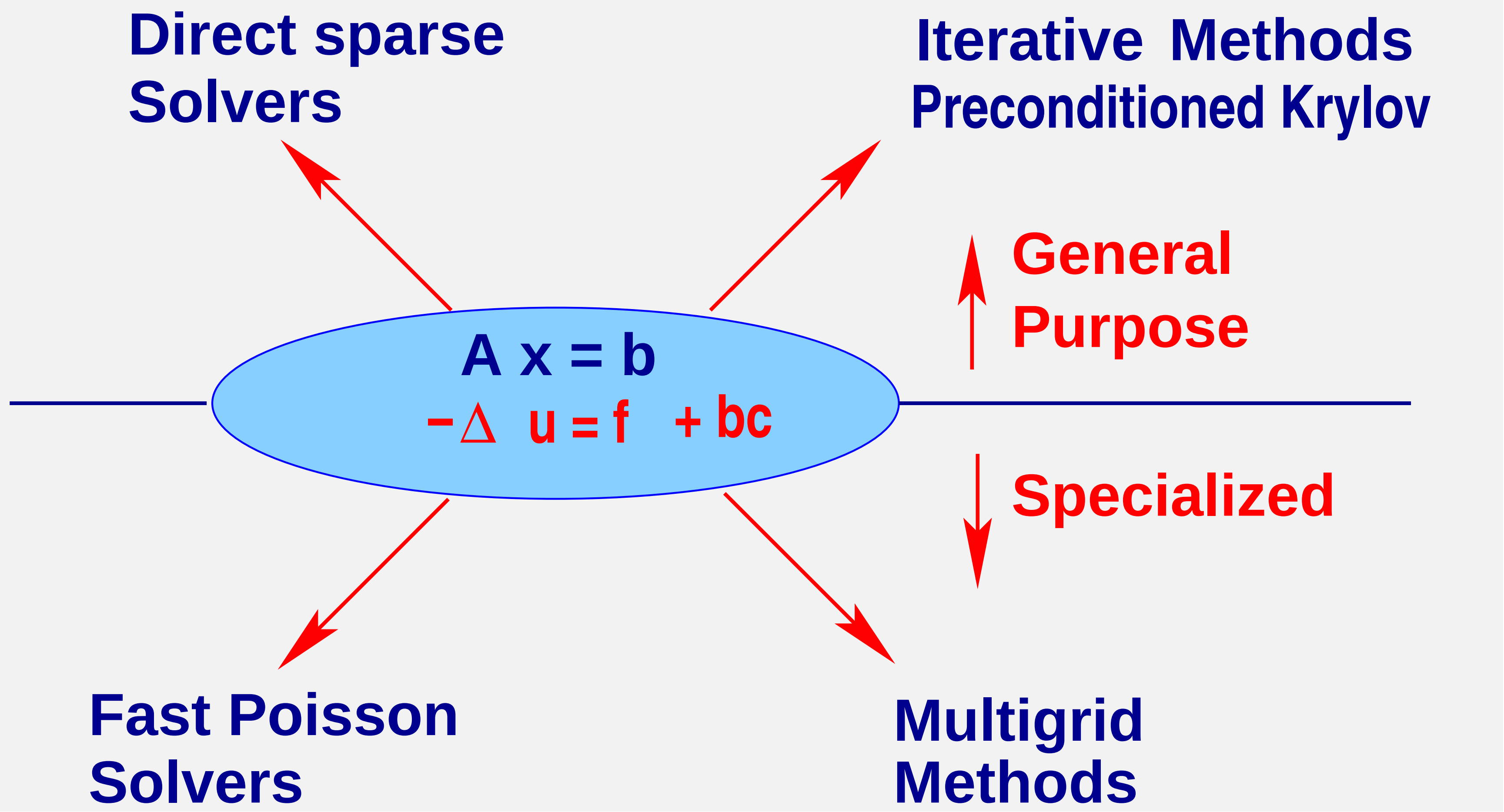
Yousef Saad

Department of Computer Science and Engineering

University of Minnesota

"Maillages et EDP", Nancy, June 9, 2010

Introduction: Linear System Solvers



A few observations

- Problems are getting harder for Sparse Direct methods (more 3-D models, much bigger problems,..)
- Problems are also getting difficult for iterative methods Cause: more complex models - away from Poisson
- Researchers on both camps are learning each other's tricks to develop preconditioners.
- Much of recent work on solvers has focussed on:

(1) Parallel implementation – scalable performance

(2) Improving Robustness, developing more general preconditioners

Background: Independent sets, ILUM, ARMS

Independent set orderings permute a matrix into the form

$$\begin{pmatrix} B & F \\ E & C \end{pmatrix}$$

where B is a diagonal matrix.

➤ Unknowns associated with the B block form an independent set (IS).

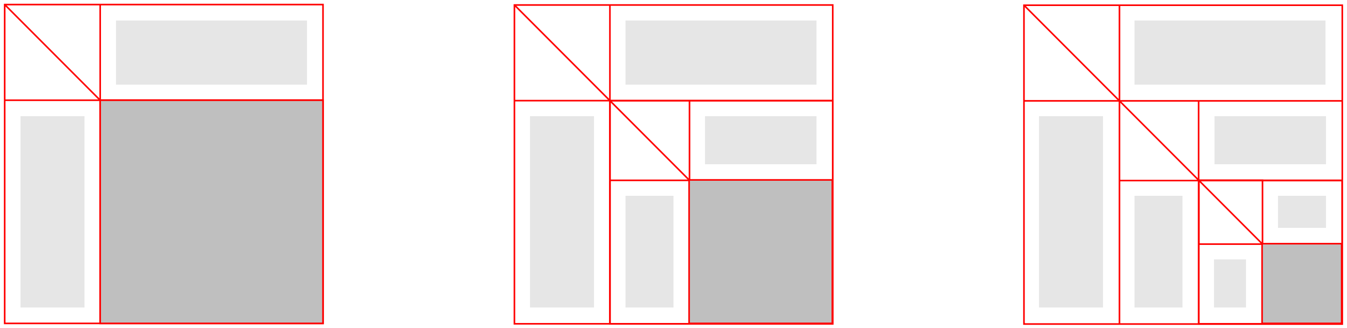
➤ IS is maximal if it cannot be augmented by other nodes

➤ Finding a maximal independent set is inexpensive

Main observation: Reduced system obtained by eliminating the unknowns associated with the IS, is still sparse since its coefficient matrix is the Schur complement

$$S = C - EB^{-1}F$$

- Idea: apply IS set reduction recursively.
- When reduced system small enough solve by any method
- ILUM: ILU factorization based on this strategy. YS '92-94.

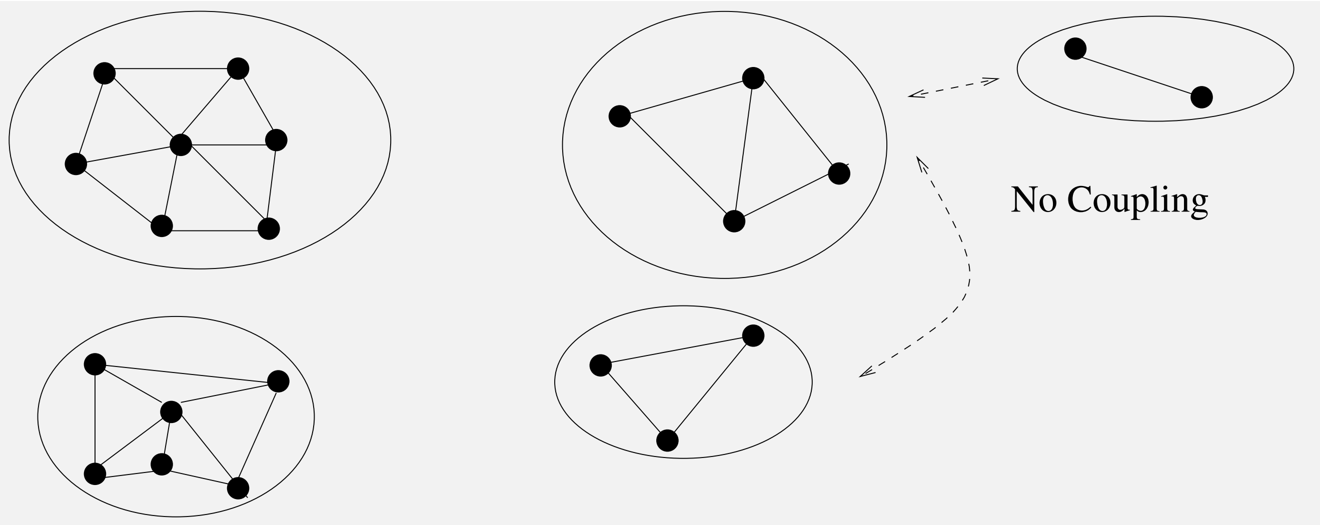


- See work by [Botta-Wubbs '96, '97, YS'94, '96, Leuze '89,..]

Group Independent Sets / Aggregates

Main goal: generalize independent sets to improve robustness

Main idea: use “cliques”, or “aggregates”. No coupling between the aggregates.

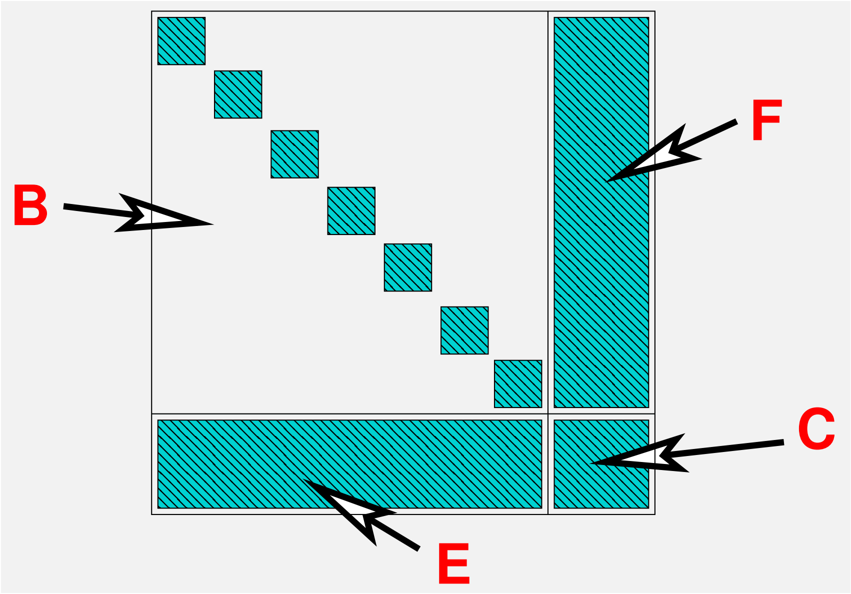


- Label nodes of independent sets first

Algebraic Recursive Multilevel Solver (ARMS)

- Typical shape of reordered matrix:

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



- 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

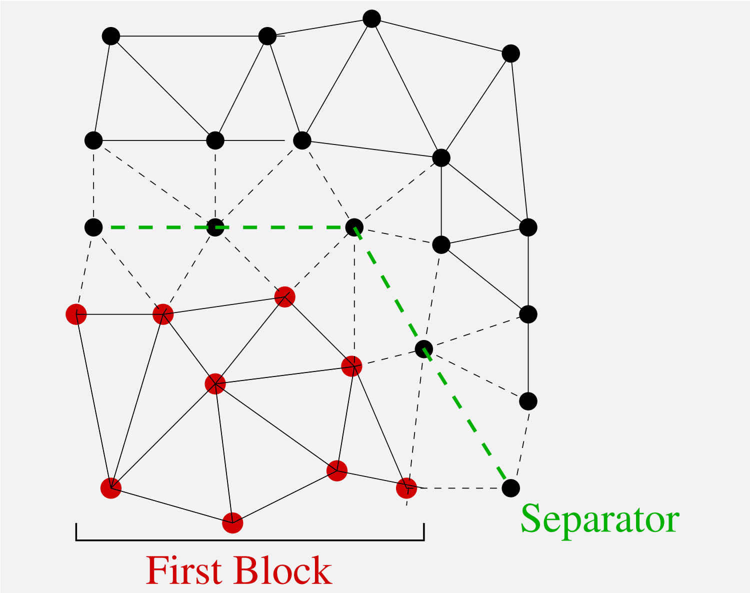
Algebraic Recursive Multilevel Solver (ARMS)

Level l Factorization:

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

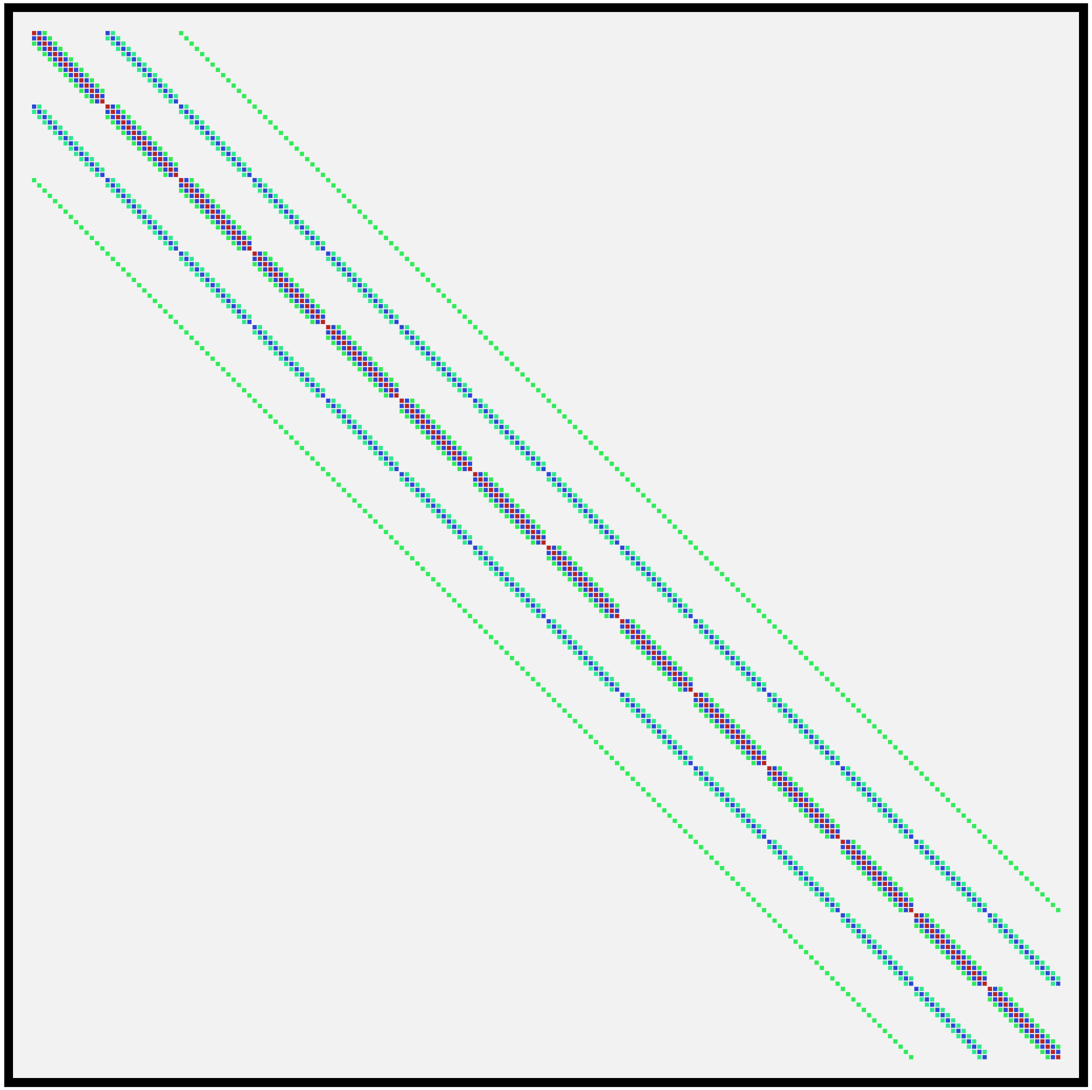
- L-solve $\sim$ restriction; U-solve $\sim$ prolongation.
- Perform above block factorization recursively on A_{l+1}
- Blocks in B_l treated as sparse. Can be large or small.
- Algorithm is fully recursive
- Stability criterion in block independent sets algorithm

Group Independent Set reordering

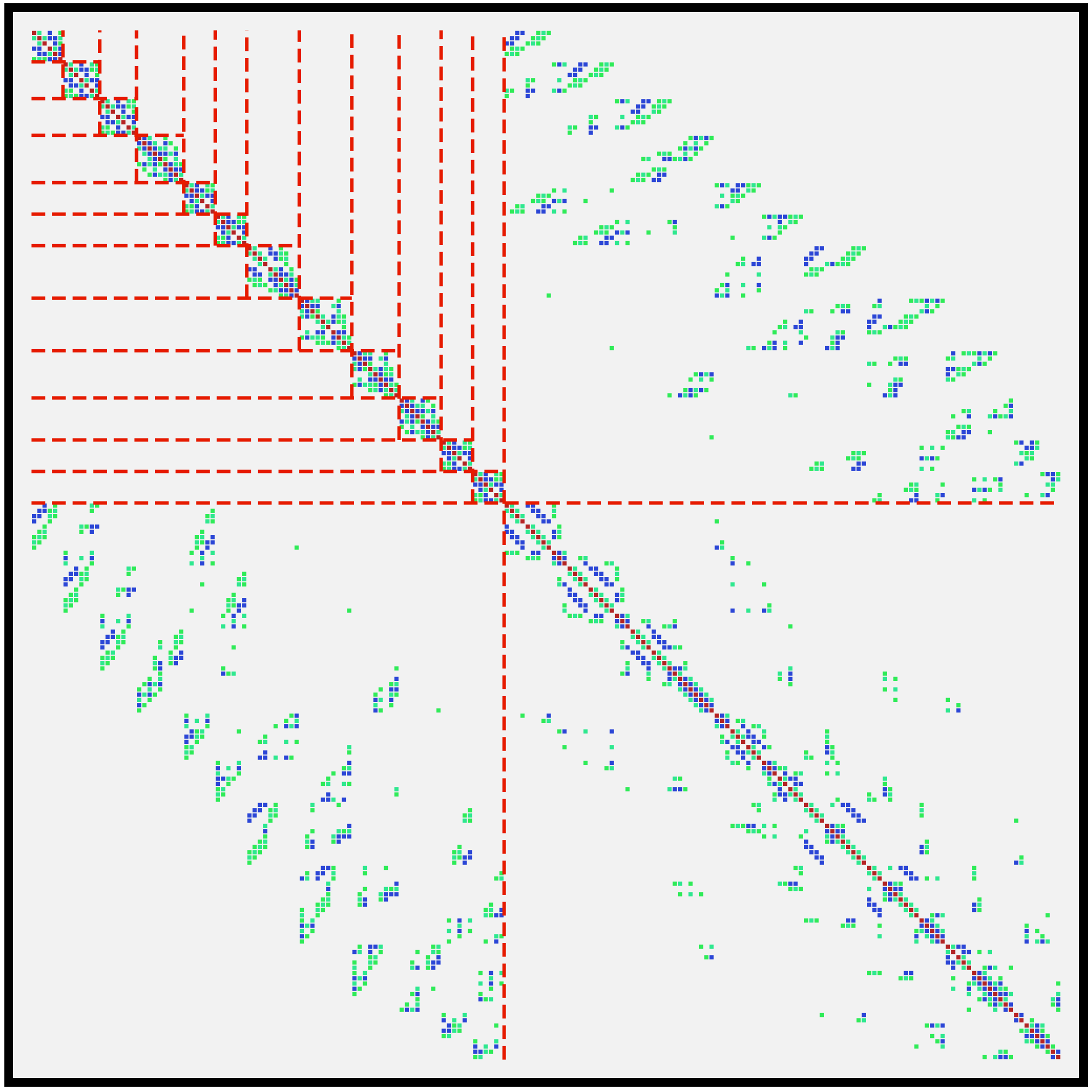


<u>Simple strategy:</u> Level taversal until there are enough points to form a block. Reverse ordering. Start new block from non-visited node. Continue until all points are visited. Add criterion for rejecting "not sufficiently diagonally dominant rows."

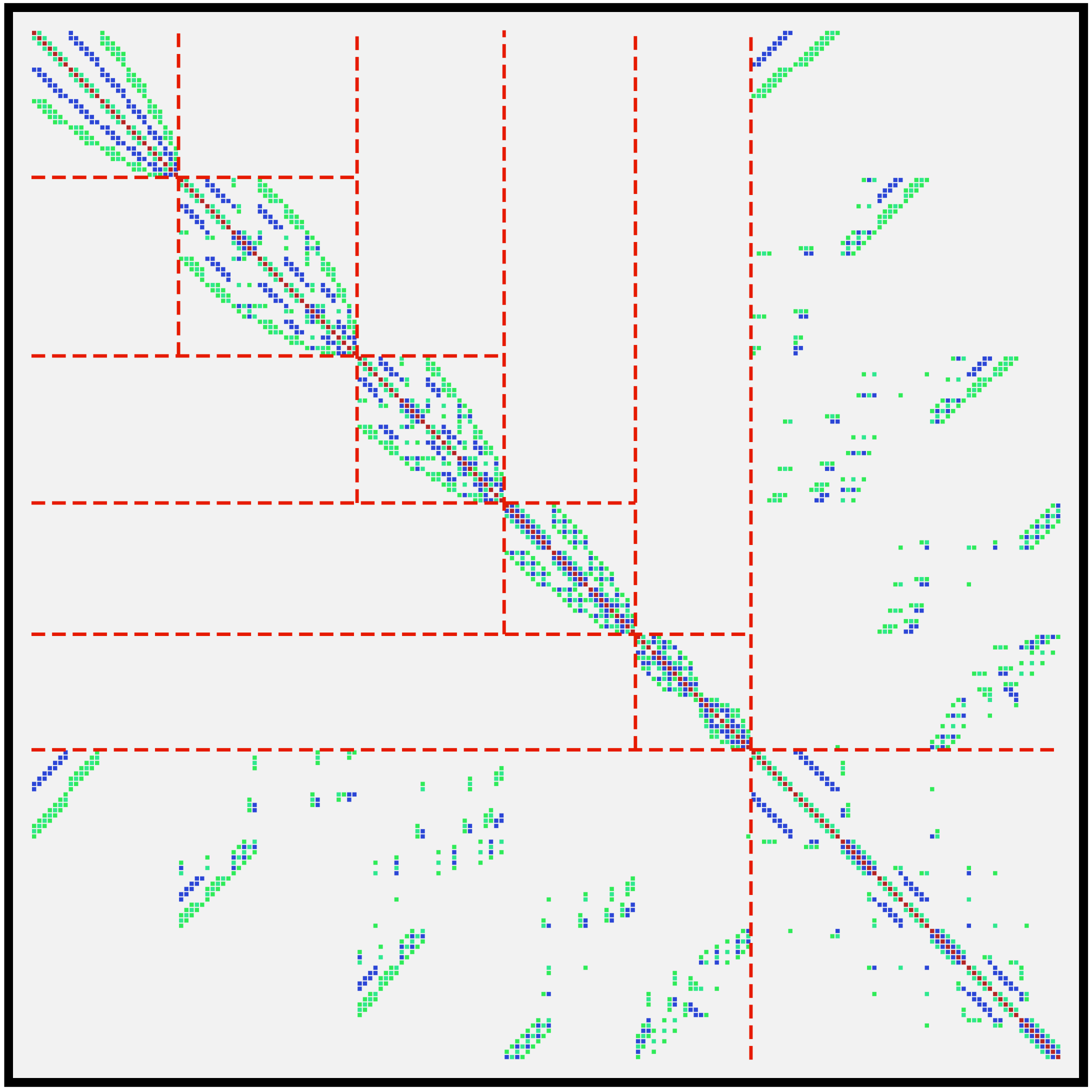
Original matrix



Block size of 6



Block size of 20



Related ideas

➤ See *Y. Notay*, Algebraic Multigrid and algebraic multilevel techniques, a theoretical comparison, NLAA, 2005.

➤ Some of these ideas are related to work by Axelsson and co-workers [e.g., AMLI] – see Axelson's book

➤ Work by Bank & Wagner on MLILU quite similar to ARMS – but uses AMG framework: [*R. E. Bank and C. Wagner*, Multilevel ILU decomposition, Numer. Mat. (1999)]

➤ Main difference with AMG framework: block ILU-type factorization to obtain Coarse-level operator. + use of relaxation.

➤ In AMG $S = P^T A P$ with P of size $(n_F + n_C) \times n_C$

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 $\longrightarrow$ 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 $n_M \leq n$ with $1 \leq p_i, q_i \leq n$ 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 $n_M = n \longrightarrow$ (full) permutation pair (P, Q). A partial matching set can be easily completed into a full pair (P, Q) by a greedy approach.

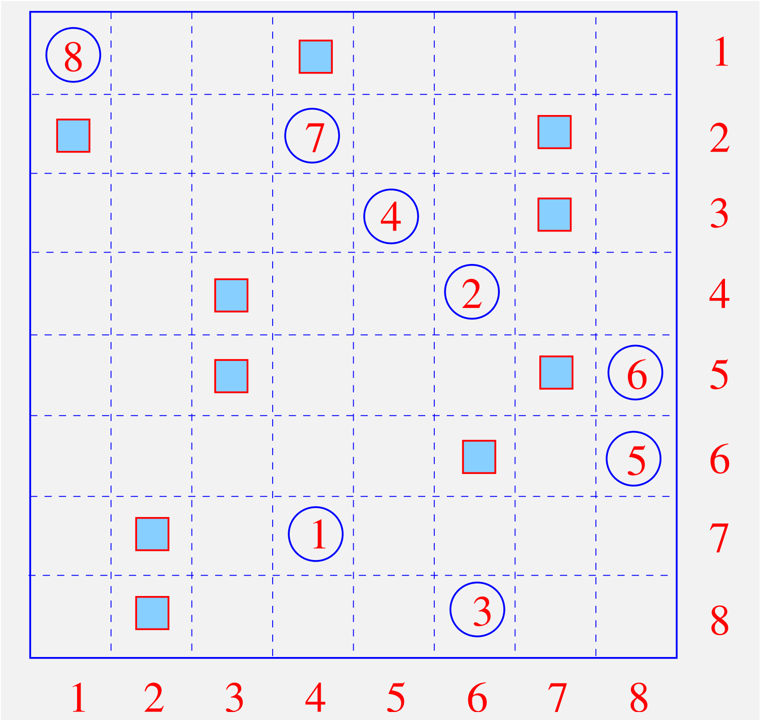
Matching - preselection

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

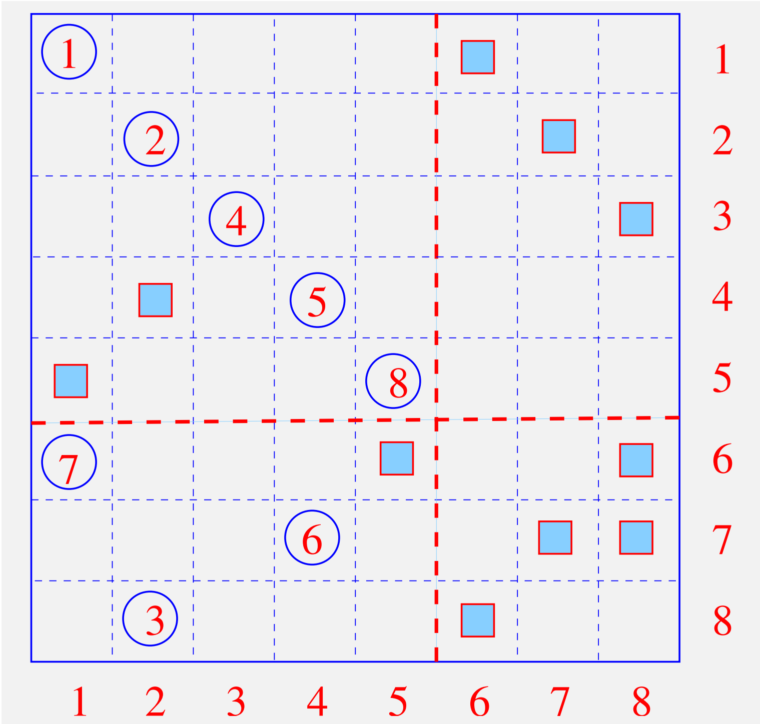
- Let $j(i) = \mathbf{argmax}_j |a_{ij}|$.
- Use the ratio $\gamma_i = \frac{|a_{i,j(i)}|}{\|a_{i,:}\|_1}$ as a measure of diag. domin. of row i

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.

MATLAB DEMO

Software

- The matlab demo just shown available from my web-site. Search for “matlab suite” in

  `http://www.cs.umn.edu/∼saad/software`

- ARMS-C [C-code] - available from ITSOL package..
- Parallel version of ARMS available. pARMS3 released recently
- See also: ILUPACK – developed mainly by Matthias Bollhoefer and his team

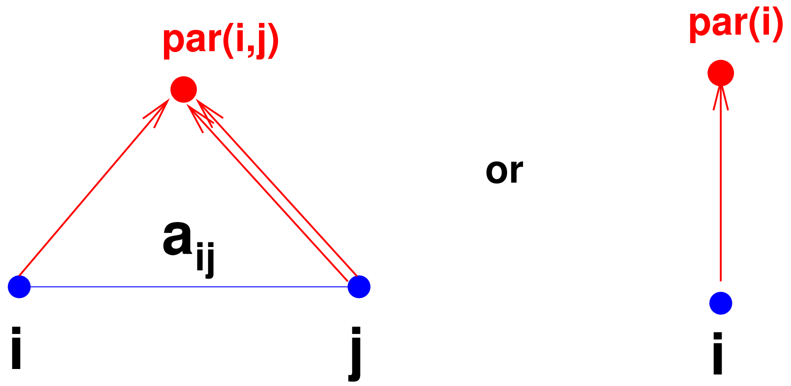
  `http://www.tu-berlin.de/ilupack/.`

COARSENING

Divide and conquer and coarsening (work in progress)

➤ Want to mix ideas from AMG with purely algebraic strategies based on graph coarsening

First step: Coarsen. We use matching: coalesce two nodes into one 'coarse' node

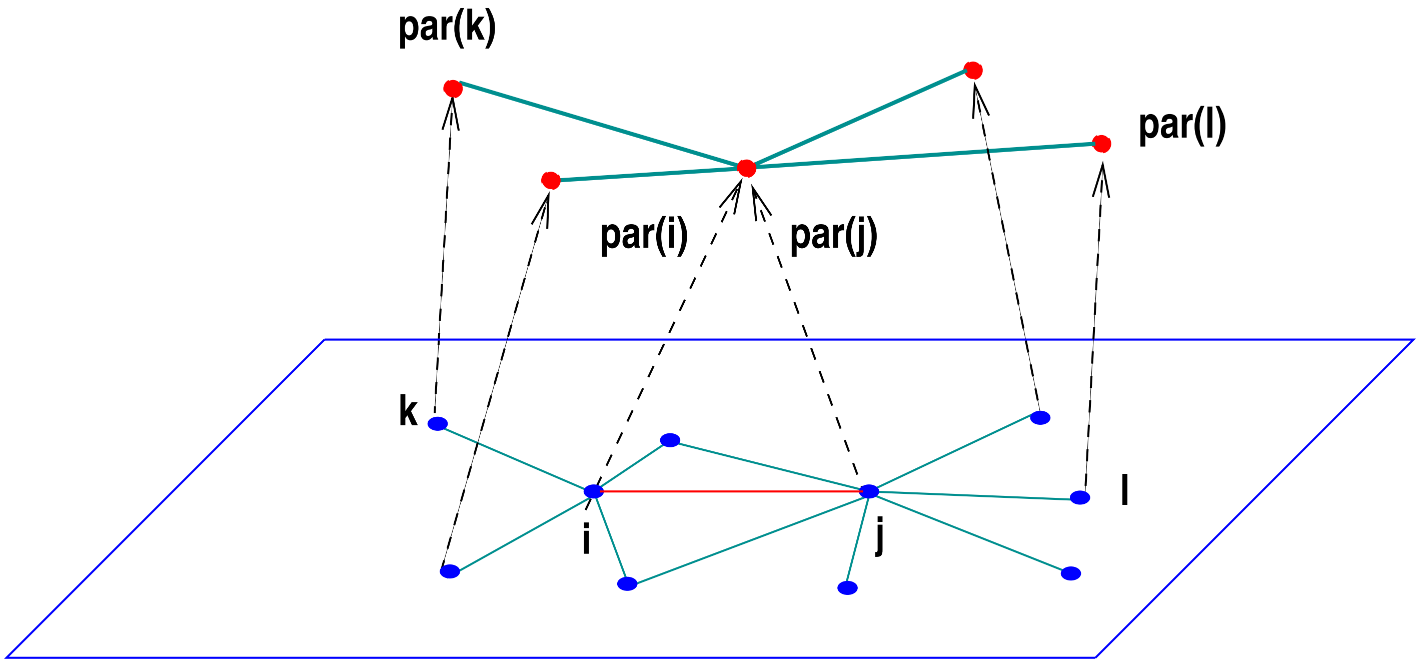


Second step: Get graph (+ weights) for the coarse nodes - Adj$[par(i,j)]$ is:

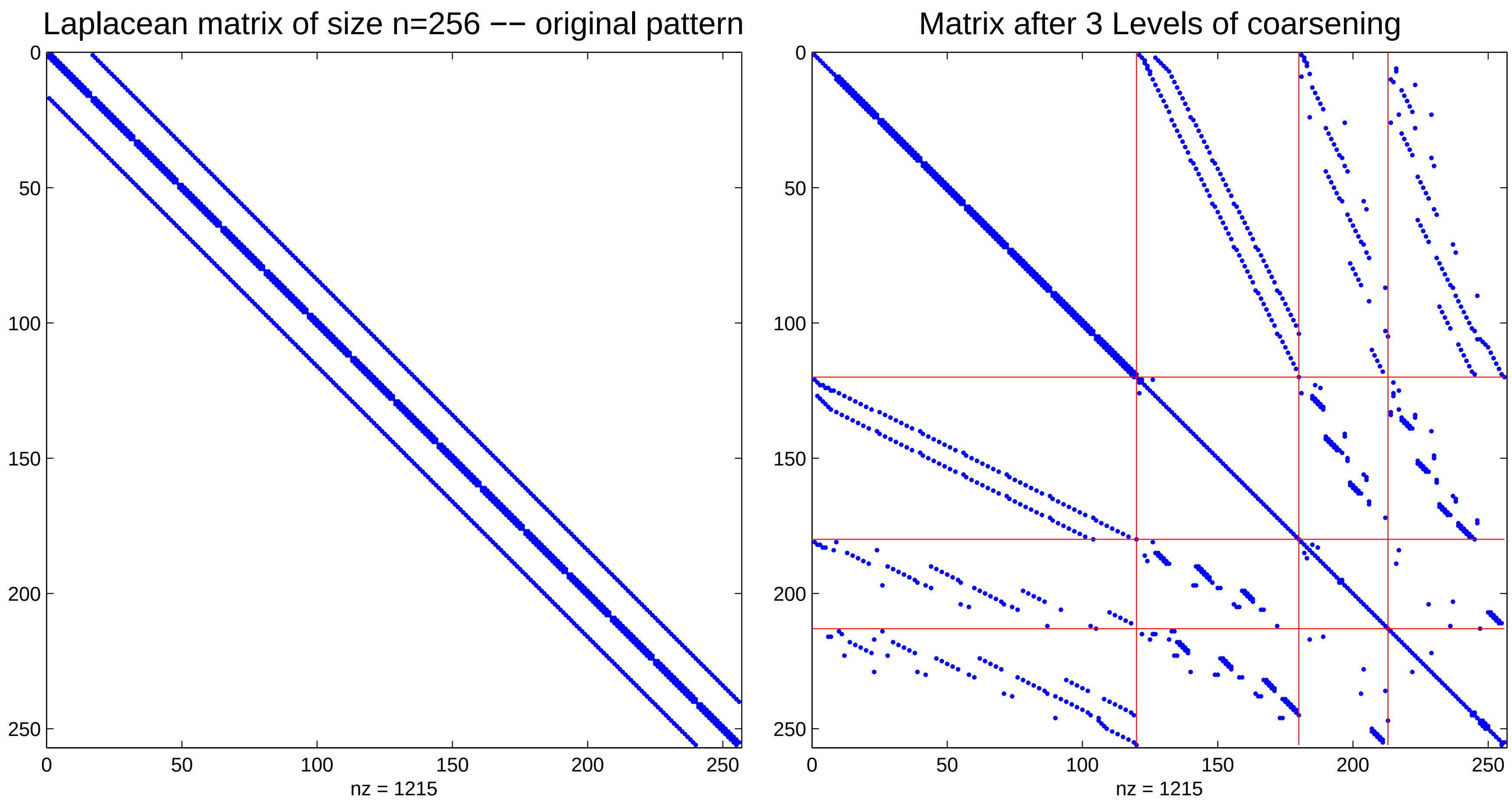
$$\{par(i,k)\ k \in Adj(i)\} \bigcup \{par(j,k)\ k \in Adj(j)\}$$

Third step: Repeat

Illustration of the coarsening step



Example 1: A simple 16×16 mesh ($n = 256$).

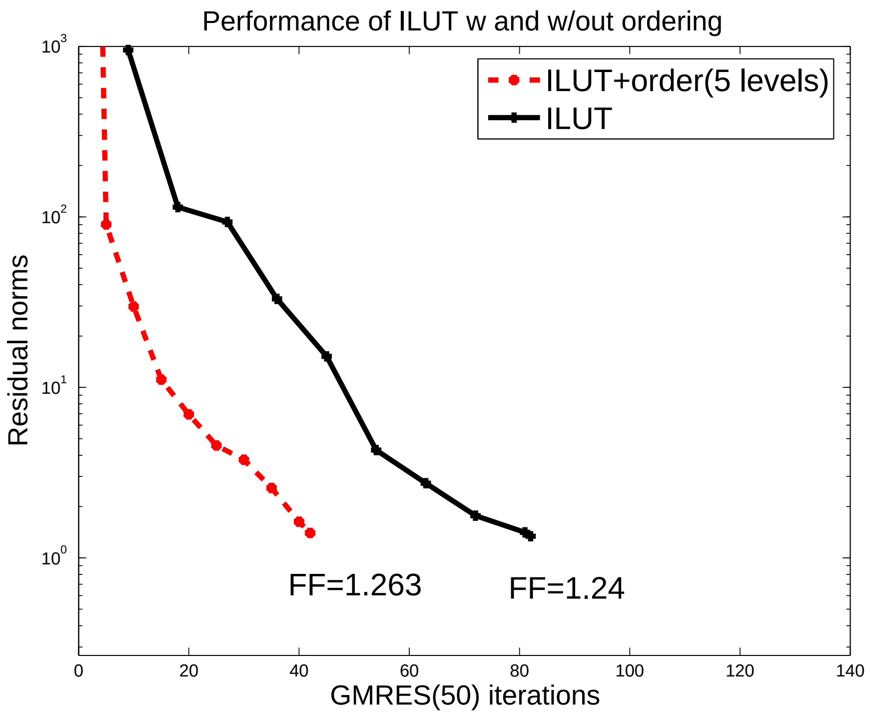


First idea: use ILU on the reordered matrix

- For example: use ILUT

Illustration: Matrix Raj1 from the Florida collection

- Size $n = 263,743$. $Nnz = 1,302,464$ nonzero entries
- Matrix is nearly singular – poorly conditioned. Iterate to reduce residual by 10^{10}.



- Reordering appears to be quite good for ILU.

Saving memory with Pruned ILU

- Let $A = \begin{pmatrix} B & F \\ E & C \end{pmatrix} = \begin{pmatrix} I & 0 \\ EB^{-1} & I \end{pmatrix} \begin{pmatrix} B & F \\ 0 & S \end{pmatrix}$;
- $S = C - EB^{-1}F$ = Schur complement

Solve:

$$\begin{pmatrix} I & 0 \\ EB^{-1} & I \end{pmatrix} \begin{pmatrix} B & F \\ 0 & S \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \cdot\cdot$$

1) $w_1 = B^{-1} b_1$
2) $w_2 = b_2 - E * w_1$
3) $x_2 = S^{-1} w_2$
4) $w_1 = b_1 - F * x_2$
5) $x_1 = B^{-1} w_1$

- Known result: LU factorization of S == trace of LU factorization of A.
- Idea: exploit recursivity for B-solves - keep only the block-diagonals from ILU..

From L U =

$$\left[\begin{array}{cc|c} B_1 & B_1^{-1}F_1 & \\ \hline E_1B_1^{-1} & S_1 & B_2^{-1}F_2 \\ \hline \multicolumn{2}{c|}{E_2B_2^{-1}} & S_2 \end{array}\right]$$

Keep only

$$\left[\begin{array}{cc|c} B_1 & & \\ \hline & S_1 & \\ \hline & & S_2 \end{array}\right]$$

- Big savings in memory
- Additional computational cost
- Expensive for more than a few levels (2 or 3)..

Example : A simple 16×16 mesh ($n = 256$).

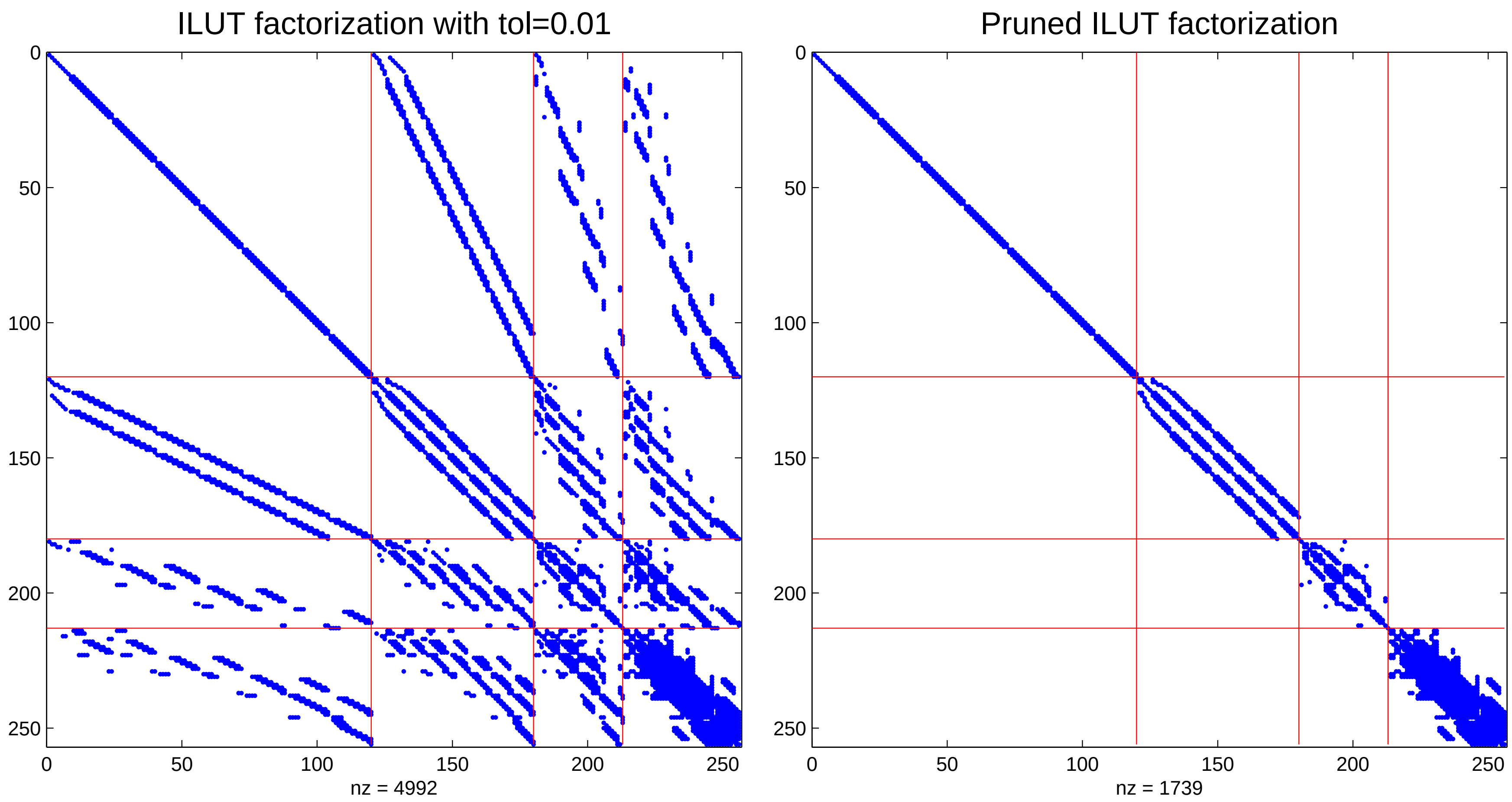
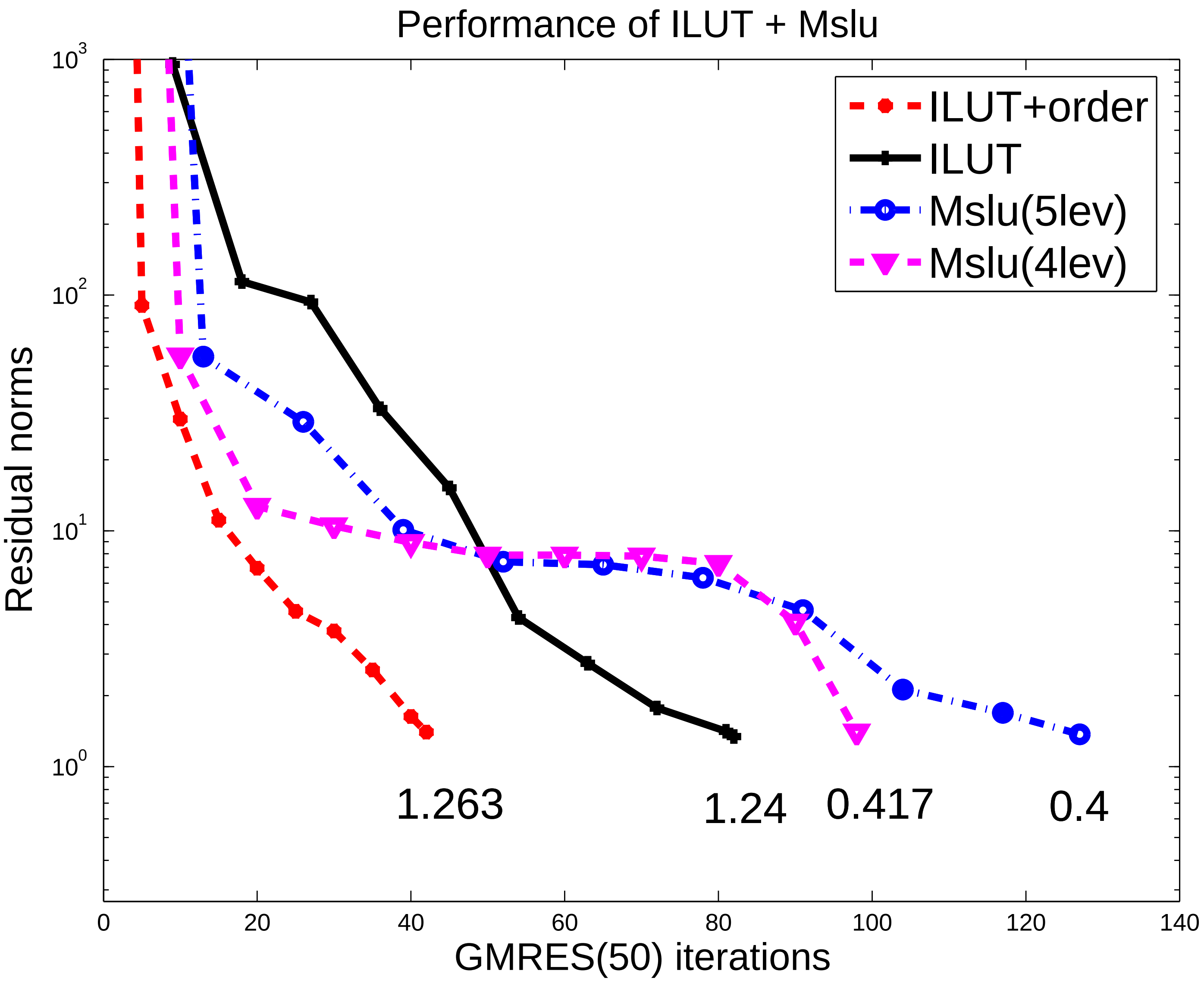


Illustration: Back to Raj1 matrix from the Florida collection



HELMHOLTZ

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

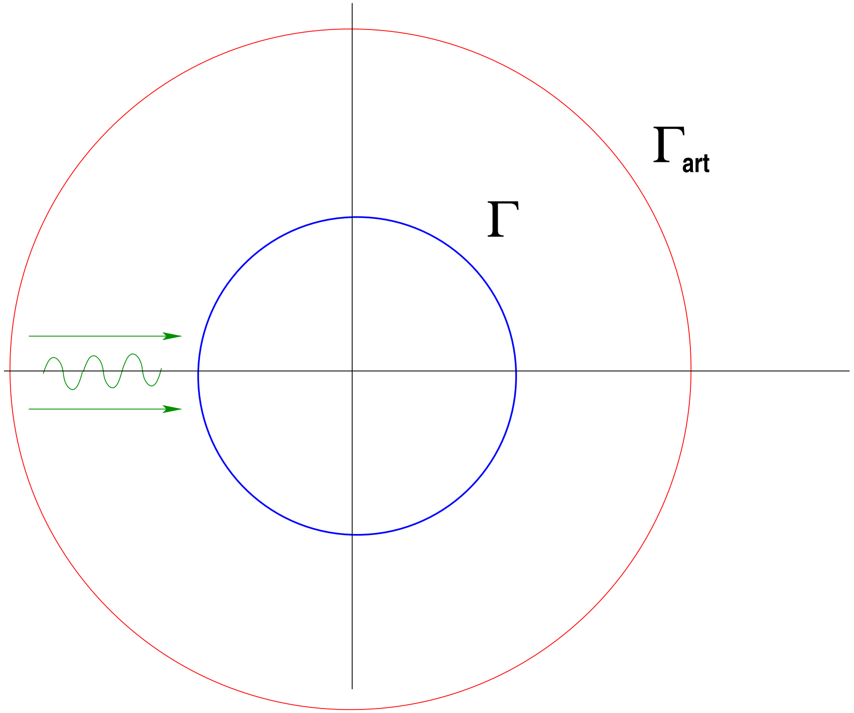
- Issue: non-reflective boundary conditions when making the domain finite.

- 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

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.



Use of complex shifts

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

[1] X. Antoine – Private comm.

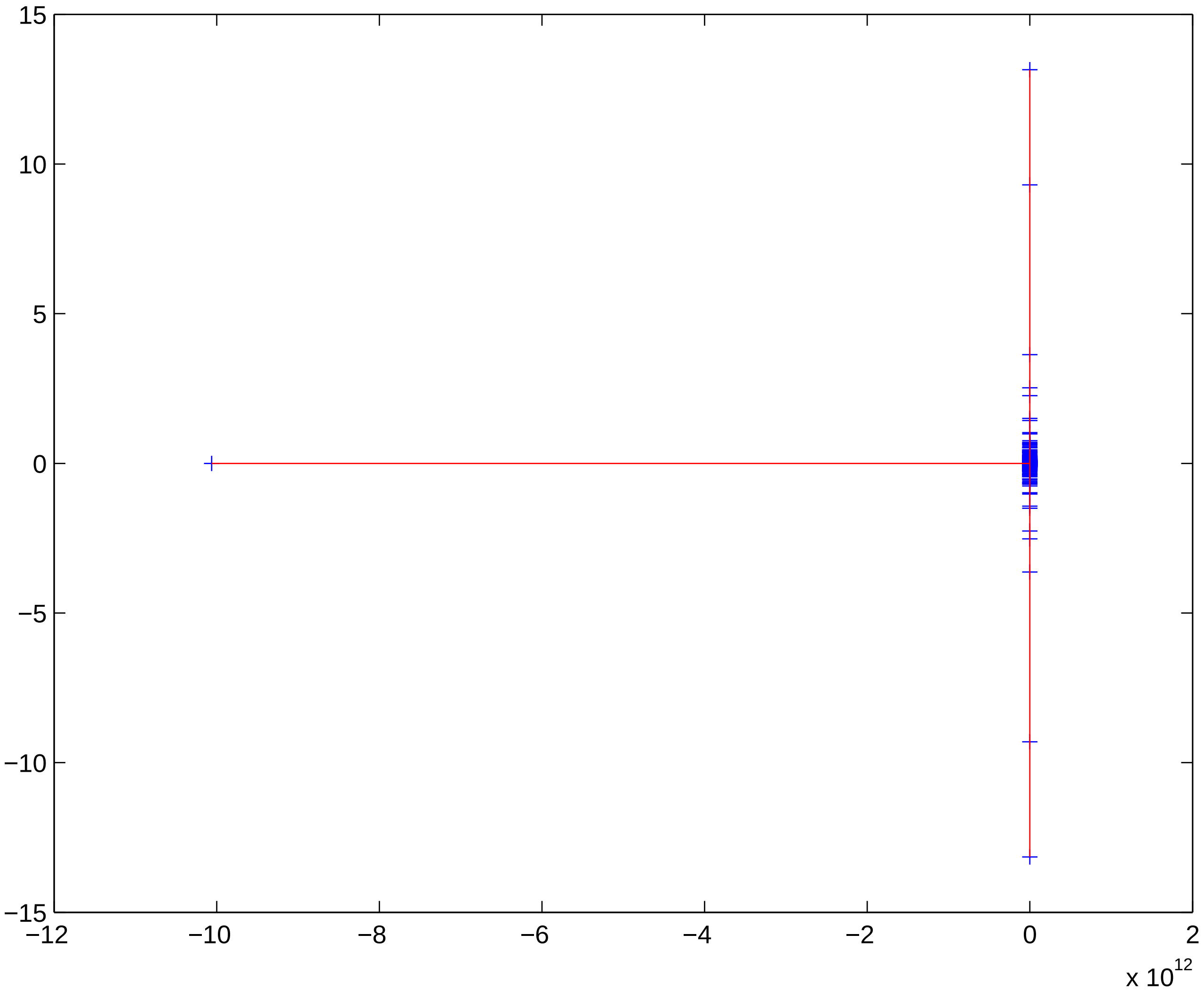
[2] Y.A. Erlangga, C.W. Oosterlee and C. Vuik, SIAM J. Sci. Comput.,27, pp. 1471-1492, 2006

[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.

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

- Terrible spectrum:

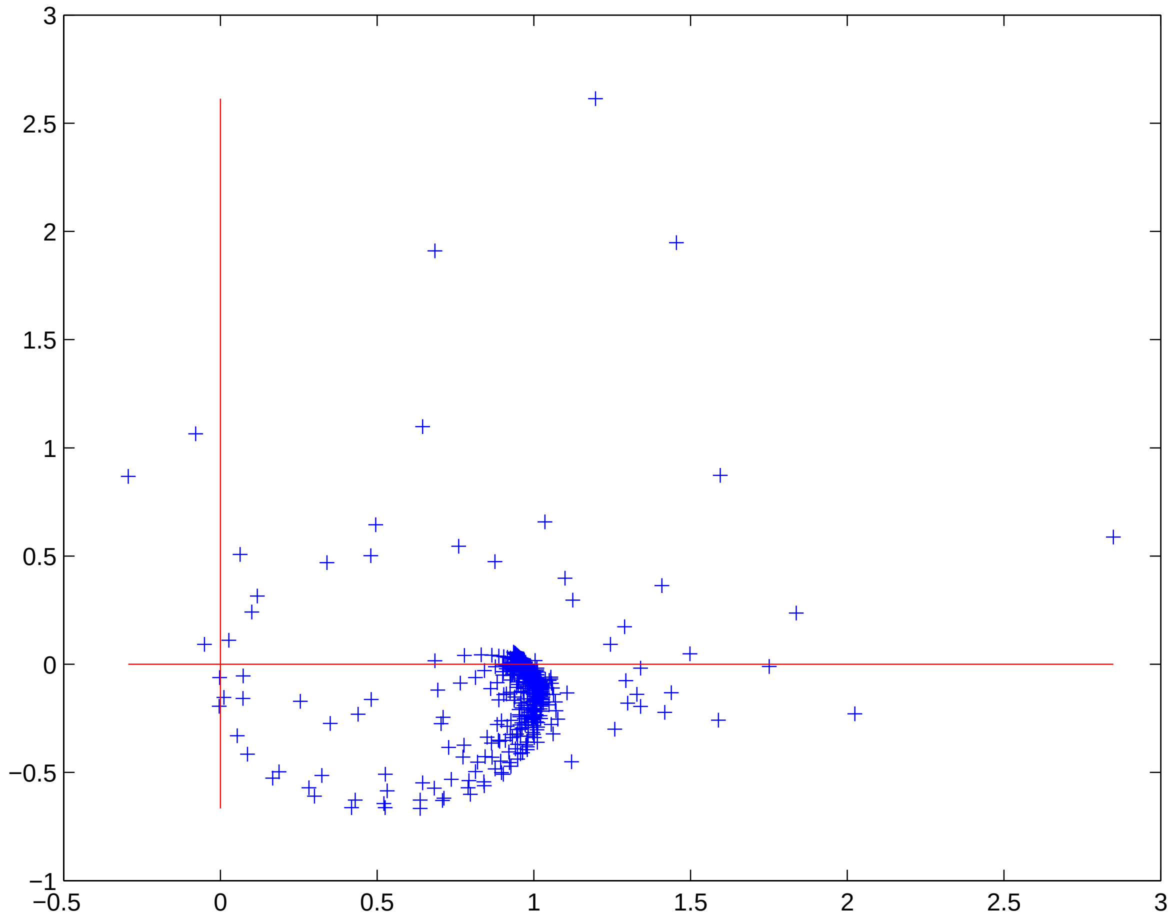


➤ Now plot eigs of $L^{-1}AU^{-1}$ where L, U are inc. LU factors of $B = A + 0.25 * i$

➤ Much better! Observed by many [PDE viewpoint]

Idea:

Adapt technique to ILU:
Add complex shifts before ILU



Question:

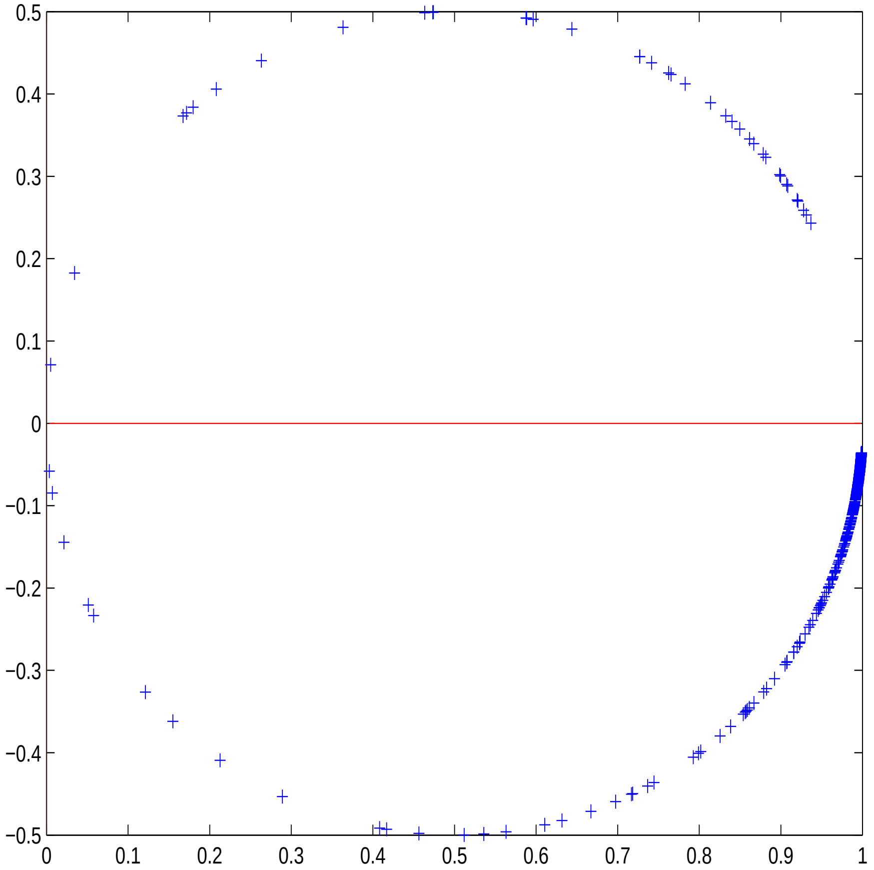
What if we do an exact factorization [droptol = 0]?

➤ $\Lambda(L^{-1}AU^{-1}) = \Lambda[(A+\alpha iI)^{-1}A]$

➤ $\Lambda = \left\{\frac{\lambda_j}{\lambda_j + i\alpha}\right\}$

➤ Located on a circle – with a cluster at one.

➤ Figure shows situation on the same example



Recent comparisons

** Joint work with Daniel Osei-Kuffuor

- Test problem seen earlier. Mesh size $1/h = 160 \to n = 28,980$, $nnz = 260,280$
- Wavenumber varied [until convergence fails]

ILUT with $droptol = 0.02$					
k	$\frac{\lambda}{h}$	**No. iters**	**Setup Time (s)**	**Iter. Time (s)**	**Fill Factor**
2π	160	191	0.1	6.03	1.35
4π	80	214	0.1	6.86	1.37
8π	40	317	0.11	9.67	1.42
16π	20	**	**	**	**

ILUT – with complex shifts – $droptol = 0.02$					
k	$\frac{\lambda}{h}$	No. iters	Setup Time (s)	Iter. Time (s)	Fill Factor
2π	160	191	0.1	5.34	1.35
4π	80	211	0.1	5.90	1.36
8π	40	280	0.11	7.89	1.41
16π	20	273	0.11	7.90	1.60
32π	10	163	0.18	5.41	2.5
64π	5	107	0.33	4.25	3.84
ARMS-ddPQ					
k	$\frac{\lambda}{h}$	No. iters	Setup Time (s)	Iter. Time (s)	Fill Factor
2π	160	180	0.68	9.20	2.07
4π	80	224	0.71	11.5	2.09
8π	40	261	0.54	11.8	2.17
16π	20	127	0.58	5.71	2.39
32π	10	187	0.69	8.61	3.15
64π	5	231	0.39	8.89	3.50

DIAGONAL ESTIMATORS

Application: Computing Diag[Inv[A]] **

- Many problems lead to the computation of Diag[Inv[A]] or (easier) Trace[Inv[A]]

Examples:

- In Density Functional Theory (DFT): charge density is nothing but $Diag[f(H)]$, where f = step function. Approximating f by a rational function leads to evaluating Diag[Inv[A]]

- In Stastistics: $Trace[Inv[A]]$ is stochastically estimated to get parameters in Cross-Validation techniques. [Huntchinson '90]

** Joint work with J. Tang

- In Dynamic Mean Field Theory (DMFT), we look for the diagonal of "Green's function" to solve Dyson's equation.. [see J. Freericks 2005]

- In uncertainty quantification, the diagonal of the inverse of a covariance matrix is needed [Bekas, Curioni, Fedulova '09]

- Stochastic estimations of Trace(f(A)) extensively used by quantum chemists to estimate Density of States[1]

1.Ref: H. Röder, R. N. Silver, D. A. Drabold, J. J. Dong, Phys. Rev. B. 55, 15392 (1997)

Stochastic Estimator

Notation:

- A = original matrix, $B = A^{-1}$.
- $\delta(B) = \text{diag}(B)$ [matlab notation]
- $\mathcal{D}(B)$ = diagonal matrix with diagonal $\delta(B)$
- $\odot$ and $\oslash$: Elementwise multiplication and division of vectors
- $\{v_j\}$: Sequence of s random vectors

Result:

$$\delta(B) \approx \left[\sum_{j=1}^{s} v_j \odot B v_j\right] \oslash \left[\sum_{j=1}^{s} v_j \odot v_j\right]$$

Refs: C. Bekas , E. Kokiopoulou & YS ('05), Recent: C. Bekas, A. Curioni, I. Fedulova '09.

➤ Let $V_s = [v_1, v_2, \ldots, v_s]$. Then, alternative expression:

$$\mathcal{D}(B) \approx \mathcal{D}(BV_sV_s^\top)\mathcal{D}^{-1}(V_sV_s^\top)$$

Question: When is this result exact?

Main Proposition

- Let $V_s \in \mathbb{R}^{n\times n}$ with rows $\{v_{j,:}\}$; and $B \in \mathbb{C}^{n\times n}$ with elements $\{b_{jk}\}$
- Assume that: $\langle v_{j,:}, v_{k,:}\rangle = 0, \forall j \neq k$, s.t. $b_{jk} \neq 0$

Then:

$$\mathcal{D}(B) = \mathcal{D}(BV_sV_s^\top)\mathcal{D}^{-1}(V_sV_s^\top)$$

➤ Approximation to b_{ij} exact when rows i and j of V_s are $\perp$

Probing

Goal: Find V_s such that (1) s is small and (2) V_s satisfies Proposition (rows i & j orthgonoal for any nonzero b_{ij})

Difficulty: Can work only for sparse matrices but $B = A^{-1}$ is usually dense

- B can sometimes be approximated by a sparse matrix.
- Consider for some ϵ : $$(B_\epsilon)_{ij} = \begin{cases} b_{ij}, & |b_{ij}| > \epsilon \\ 0, & |b_{ij}| \le \epsilon \end{cases}$$
- B_ϵ will be sparse under certain conditions, e.g., when A is diagonally dominant
- In what follows we assume B_ϵ is sparse and set $B := B_\epsilon$.
- Pattern will be required by standard probing methods.

Generic Probing Algorithm

ALGORITHM : 1 . *Probing*

Input: A, s

Output: Matrix $\mathcal{D}(B)$

Determine $V_s := [v_1, v_2, \ldots, v_s]$

for $j \leftarrow 1$ **to** s

  Solve $Ax_j = v_j$

end

Construct $X_s := [x_1, x_2, \ldots, x_s]$

Compute $\mathcal{D}(B) := \mathcal{D}\left(X_s V_s^\top\right) \mathcal{D}^{-1}(V_s V_s^\top)$

➤ Note: rows of V_s are typically scaled to have unit 2-norm =1., so $\mathcal{D}^{-1}(V_s V_s^\top) = I$.

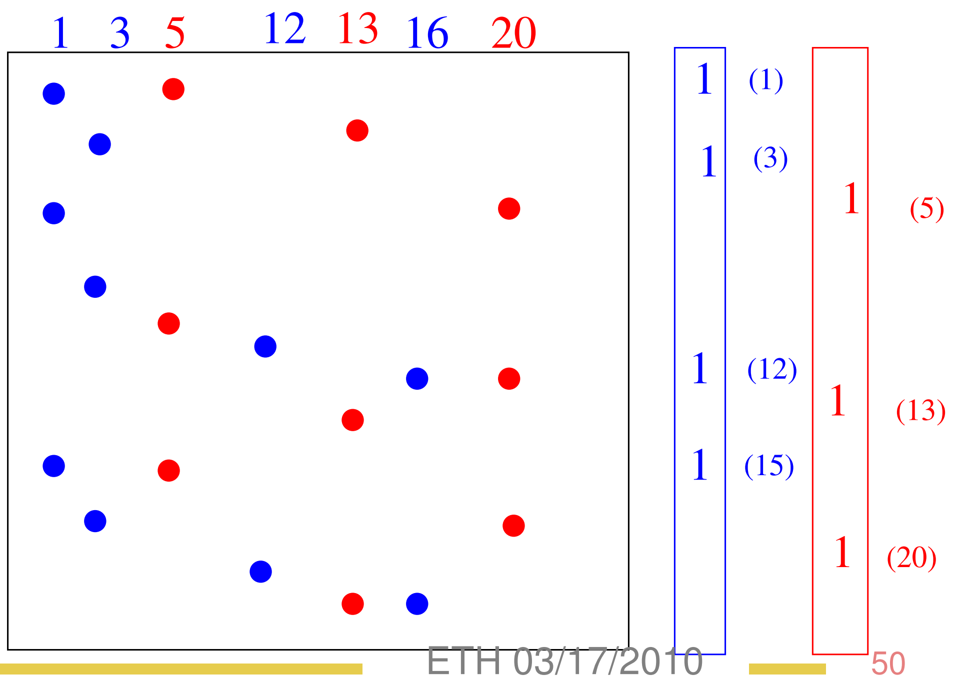
Standard probing (e.g. to compute a Jacobian)

➤ Several names for same method: “probing”; “CPR”, “Sparse Jacobian estimators”,..

Basis of the method: can compute Jacobian if a coloring of the columns is known so that no two columns in the same color overlap.

All entries of same color can be computed with one mat-vec.

Example: For all blue entries multiply B by the blue vector on right.



What about Diag(inv(A))?

- Define v_i - probing vector associated with color i:

$$[v_i]_k = \begin{cases} 1 & \text{if } color(k) == i \\ 0 & \text{otherwise} \end{cases}$$

- Will satisfy requirement of Proposition.... but
- ... this coloring is not what is needed! [It is an overkill]

Alternative:

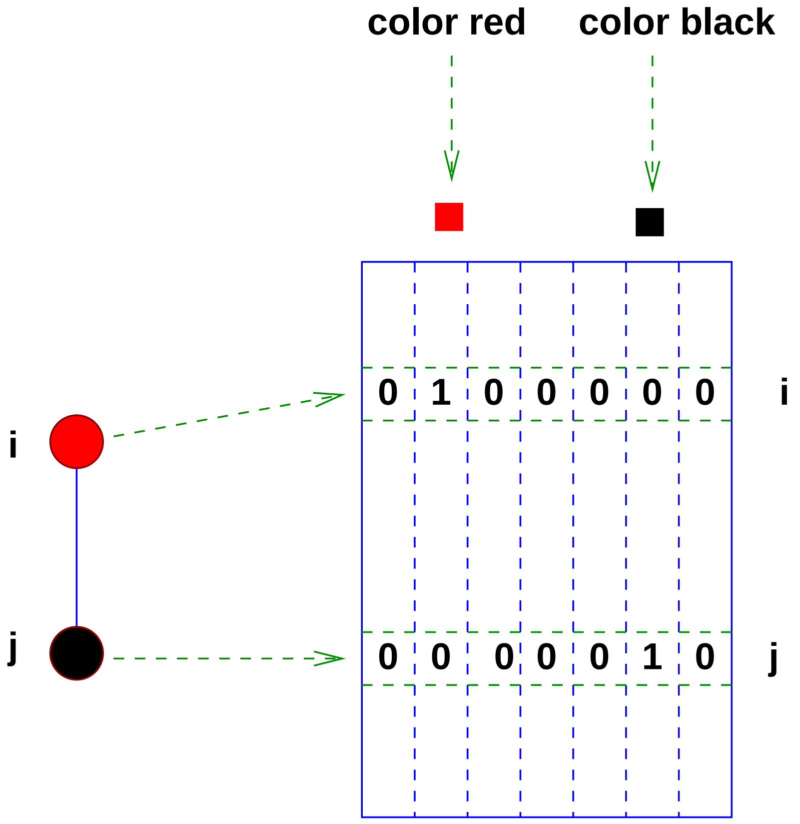
- Color the graph of B in the standard graph coloring algorithm [Adjacency graph, not graph of column-overlaps]

Result: Graph coloring yields a valid set of probing vectors for $\mathcal{D}(B)$.

Proof:

➤ Column v_c: one for each node i whose color is c, zero elsewhere.

➤ Row i of V_s: has a '1' in column c, where $c = color(i)$, zero elsewhere.

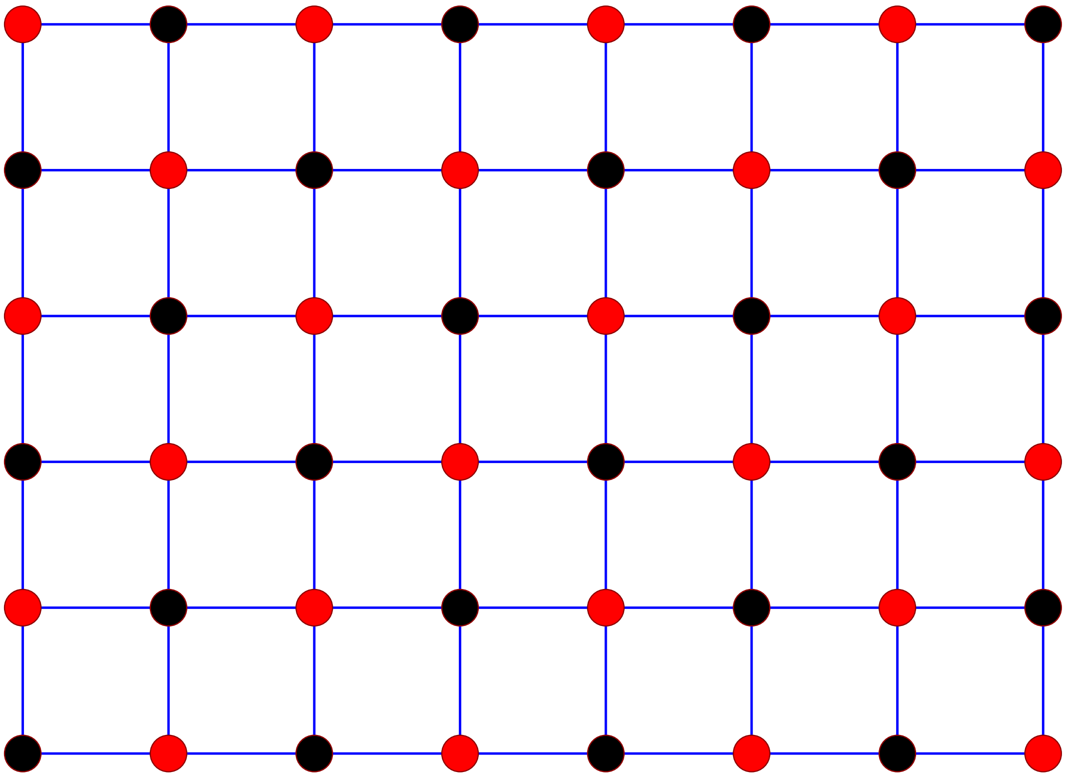


➤ If $b_{ij} \neq 0$ then in matrix V_s:

- i-th row has a '1' in column $color(i)$, '0' elsewhere.
- j-th row has a '1' in column $color(j)$, '0' elsewhere.

➤ The 2 rows are orthogonal.

Example:



- Two colors required for this graph $\longrightarrow$ two probing vectors
- Standard method: 6 colors [graph of $B^T B$]

Next Issue: Guessing the pattern of B

- Recall that we are dealing with $B := B_\epsilon$ ['pruned' B]
- Assume A diagonally dominant
- Write $\boxed{A = D - E}$, with $D = \mathcal{D}(A)$. Then :

$$A = D(I - F) \quad \text{with} \quad F \equiv D^{-1}E \quad \rightarrow$$

$$\boxed{A^{-1} \approx \underbrace{(I + F + F^2 + \cdots + F^k)D^{-1}}_{B^{(k)}}}$$

- When A is D.D. $\|F^k\|$ decreases rapidly.
- Can approximate pattern of B by that of $B^{(k)}$ for some k.
- Interpretation in terms of paths of length k in graph of A.

Q: How to select k?

A: Inspect $A^{-1}e_j$ for some j

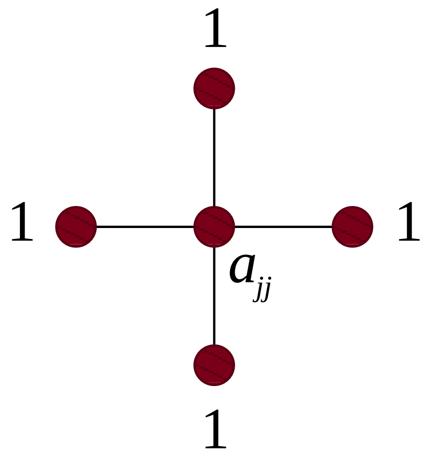
- Values of solution outside pattern of $(A^k e_j)$ should be small.
- If during calculations we get larger than expected errors – then redo with larger k, more colors, etc..
- Can we salvage what was done? Question still open.

Preliminary experiments

Problem Setup

- **DMFT**: Calculate the imaginary time Green's function
- **DMFT Parameters**: Set of physical parameters is provided
- **DMFT loop**: At most 10 outer iterations, each consisting of 62 inner iterations

- **Each inner iteration**: Find $\mathcal{D}(B)$
- **Each inner iteration**: Find $\mathcal{D}(B)$
- **Matrix**: Based on a five-point stencil with $a_{jj} = \mu + i\omega - V - s(j)$



Probing Setup

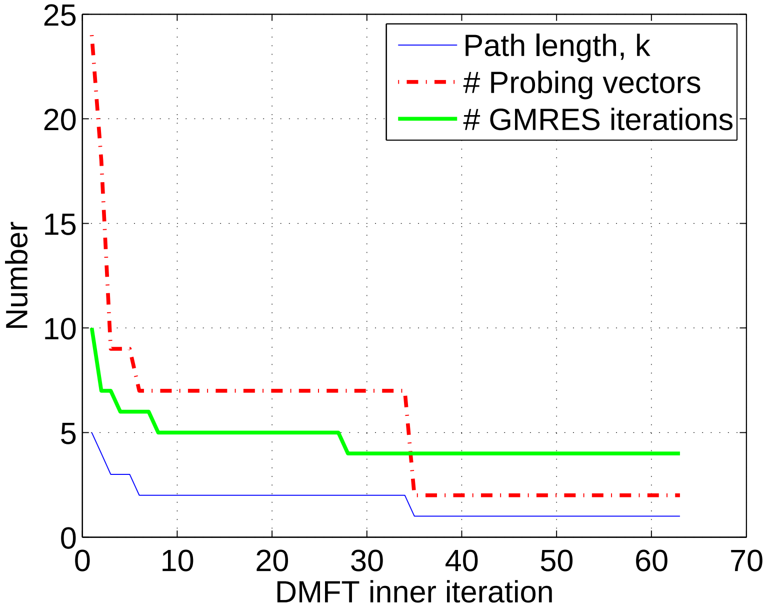
- **Probing tolerance**: $\epsilon = 10^{-10}$
- **GMRES tolerance**: $\delta = 10^{-12}$

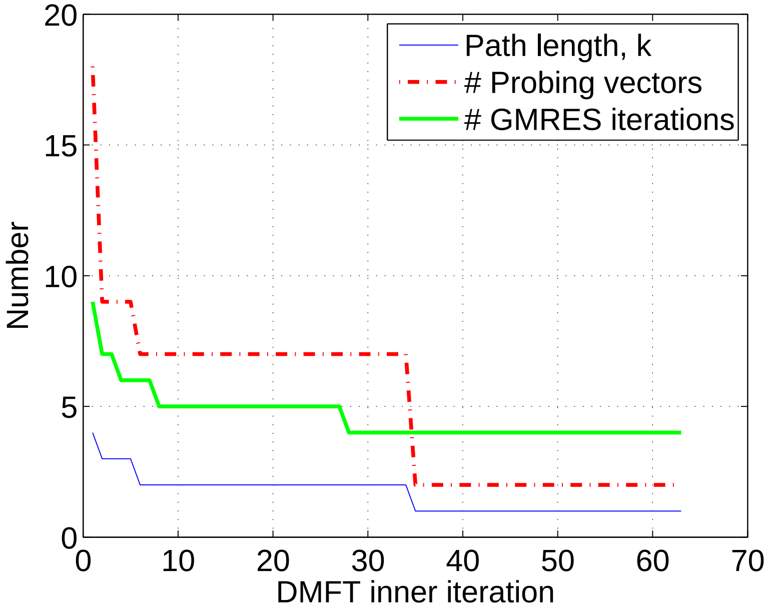
Results

CPU times (sec) for one inner iteration of DMFT

$n \longrightarrow$	21^2	41^2	61^2	81^2
LAPACK	0.5	26	282	> 1000
Lanczos	0.2	9.9	115	838
Probing	0.02	0.19	0.79	2.0

$n = 21 \times 21$ $\qquad$ $n = 81 \times 81$





Statistics for two mesh sizes

Challenge: The indefinite case

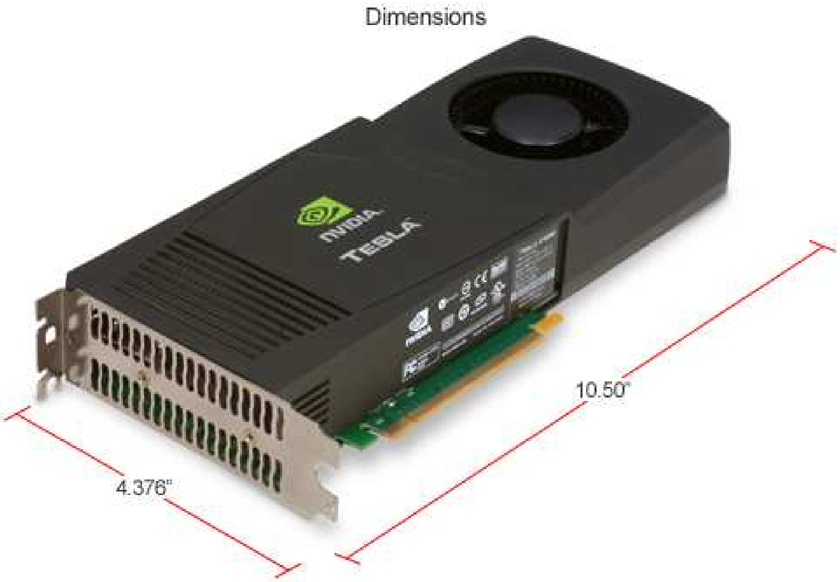
- The DMFT code deals with a separate case which uses a “real axis” sampling..
- Matrix A is no longer diagonally dominant – Far from it.
- This is a much more challenging case.
- Plan for now: solve $Ax_j = e_j$ FOR ALL j’s - with the ARMS solver using ddPQ ordering.

SPARSE MATRIX COMPUTATIONS ON GPUS

Sparse matrix computations with GPUs **

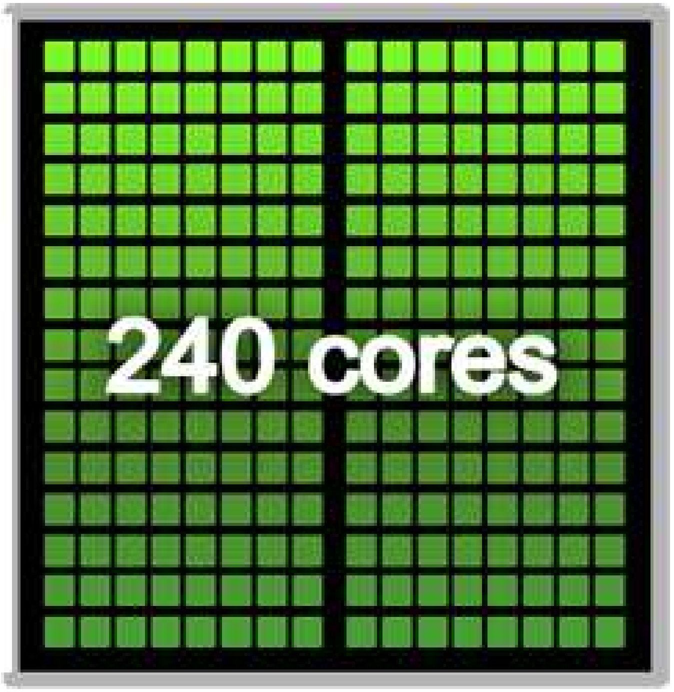
- GPUs Currently a very popular approach to: inexpensive supercomputing
- Can buy $\sim$ one Teraflop peak power for around $1,350.

Tesla C1060



** Joint work with Ruipeng Li

Tesla:



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

➤ Fermi promises to be more impressive

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

Typical program:

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

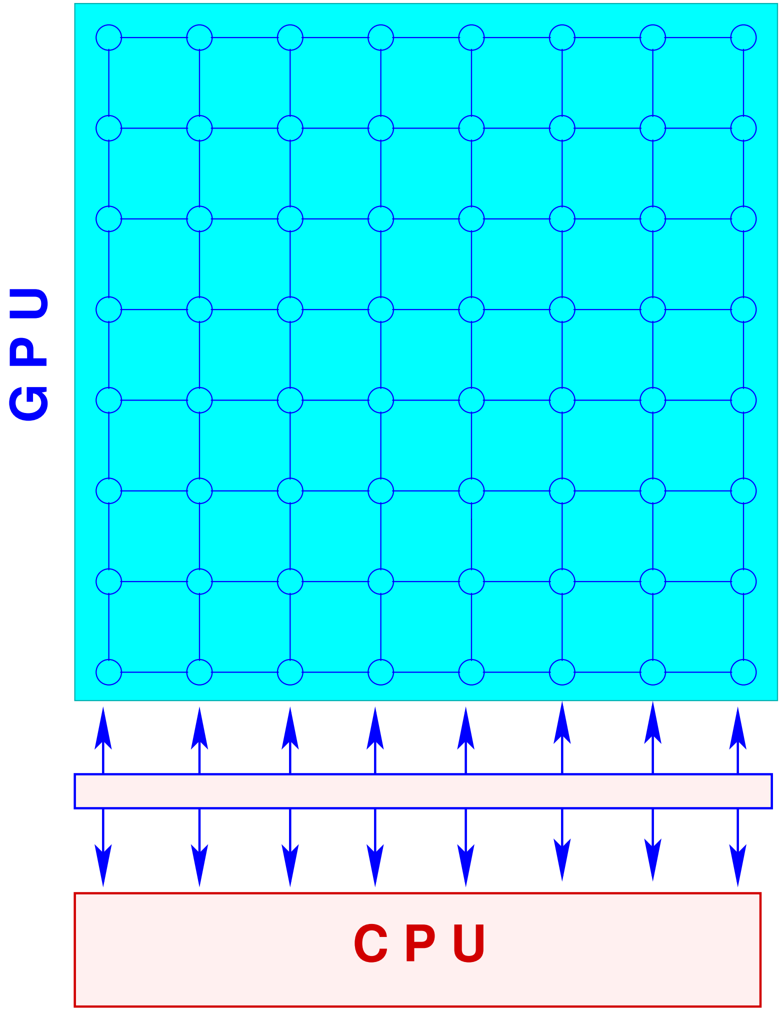
   `cudaMalloc(...)`
3. Send data Host → GPU

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

`kernel <<<(...)>>>(..)`

5. Copy data GPU →CPU

   `cudaMemcpy(...)`



Sparse Matvecs on the Tesla

- Preliminary results are mixed [high programming cost, very good performance for some calculations]
- Performance of matvec [GLOPS] on a Tesla C1060
- Matrices:

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

	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

ILU: Sparse Forward/Backward Sweeps

- Exploit Level-Scheduling.. [Topological sort]
- Poor performance relative to CPU
- Extremely poor when #levs is large
- In the worst case, #levs=n, ≈ 2 Mflops

Matrix	N	CPU Mflops	GPU-Lev	
			#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

GPU Sparse Triangular Solve with Level Scheduling

Alternative: Polynomial Preconditioners

- $M^{-1} = s(A)$, where $s(t)$ is a polynomial of low degree
- Solve: $s(A) \cdot Ax = s(A) \cdot b$
- $s(A)$ need not be formed explicitly
- $s(A) \cdot Av$: Preconditioning Operation: a sequence of matrix-by-vector product to exploit high performance Spmv kernel
- Inner product on space $\mathbb{P}_k$ ($\omega \geq 0$ is a weight on (α, β))

$$\langle p, q \rangle_\omega = \int_\alpha^\beta p(\lambda) q(\lambda) \omega(\lambda)\, d\lambda$$

- Seek polynomial s_{k-1} of degree $\leq k - 1$ which minimizes

$$\|1 - \lambda s(\lambda)\|_\omega$$

L-S Polynomial Preconditioning

Tol=1.0e-6; MaxIts=1,000; *:MD reordering applied

Matrix	ITSOL-ILU(3)		GPU-ILU(3)		L-S Polyn		
	iter.	sec.	iter.	sec.	iter.	sec.	Deg
bcsstk36	FAILED		**351***	**10.58***	31	1.34	100
ct20stif	27	9.4	**21***	**2.22***	16	0.70	50
ship_003	27	25.8	27	21.1	10	2.90	100
msc23052	181	18.5	181	6.0	37	1.28	80
bcsstk17	46	1.8	46	2.8	22	0.55	120

ILU(3) & L-S Polynomial Preconditioning

Preconditioner Time

- High level fill-in ILU preconditioner can be very expensive to build
- L-S Polynomial preconditioner set-up time $\approx$ very low
- Example: ILU(3) and L-S Poly with 20-step Lanczos procedure (for estimating interval bounds).

Matrix	N	ILU(3) sec.	LS-Poly sec.
Boeing/ct20stif	23,052	15.63	0.26

Preconditioner Construction Time

Conclusion

- General rule: ILU-based preconditioners not meant to replace tailored preconditioners. Can be very useful as parts of other techniques.
- Recent work on generalizing nonsymmetric permutations to symmetric matrices [Duff-Pralet, 2006].
- Complex shifting strategy quite useful even for real matrices
- Diag(inv(A)) problem - fairly easy for D.D case. Very challenging in indefinite case: B is dense and 'equimodular'
- GPUs for irregular sparse matrix computations: Much remains to be done both in hardware and in algorithms/software

Software:

`http://www.cs.umn.edu/~saad/software`

- ➤ ARMS-C [C-code] - available from ITSOL package..
- ➤ Parallel version of ARMS available. pARMS3 released recently
- ➤ See also: ILUPACK – developed mainly by Matthias Bollhoefer and his team

`http://www.tu-berlin.de/ilupack/.`