

A short course on: Preconditioned Krylov subspace methods

Yousef Saad
University of Minnesota
Dept. of Computer Science and
Engineering

Universite du Littoral, Jan 19-30, 2005

Outline

Part 1

- Introd., discretization of PDEs
- Sparse matrices and sparsity
- Basic iterative methods (Relaxation..)

Part 2

- Projection methods
- Krylov subspace methods

Part 3

- Preconditioned iterations
- Preconditioning techniques
- Parallel implementations

Part 4

- Eigenvalue problems
- Applications –

Preconditioning – Basic principles

Basic idea is to use the Krylov subspace method on a modified system such as

$$M^{-1}Ax = M^{-1}b.$$

- The matrix $M^{-1}A$ need not be formed explicitly; only need to solve $Mw = v$ whenever needed.
- Consequence: fundamental requirement is that it should be easy to compute $M^{-1}v$ for an arbitrary vector v .

Left, Right, and Split preconditioning

Left preconditioning

$$M^{-1}Ax = M^{-1}b$$

Right preconditioning

$$AM^{-1}u = b, \text{ with } x = M^{-1}u$$

Split preconditioning . Assume M is factored: $M = M_L M_R$.

$$M_L^{-1}AM_R^{-1}u = M_L^{-1}b, \text{ with } x = M_R^{-1}u$$

Preconditioned CG (PCG)

- ▶ Assume: A and M are both SPD.
- ▶ Applying CG directly to $M^{-1}Ax = M^{-1}b$ or $AM^{-1}u = b$ won't work because coefficient matrices are not symmetric.
- ▶ Alternative: when $M = LL^T$ use split preconditioner option
- ▶ Second alternative: Observe that $M^{-1}A$ is self-adjoint wrt M inner product:

$$(M^{-1}Ax, y)_M = (Ax, y) = (x, Ay) = (x, M^{-1}Ay)_M$$

Note $M^{-1}A$ is also self-adjoint with respect to $(\cdot, \cdot)_A$:

$$(M^{-1}Ax, y)_A = (AM^{-1}Ax, y) = (x, AM^{-1}Ay) = (x, M^{-1}Ay)_A$$

- ▶ Can obtain a similar algorithm
- ▶ Assume that $M =$ Cholesky product $M = LL^T$.

Then, another possibility: Split preconditioning option, which applies CG to the system

$$L^{-1}AL^{-T}u = L^{-1}b, \text{ with } x = L^T u$$

- ▶ Notation: $\hat{A} = L^{-1}AL^{-T}$. All quantities related to the preconditioned system are indicated by $\hat{\cdot}$.

Preconditioned CG (PCG)

ALGORITHM : 1. Preconditioned Conjugate Gradient

1. Compute $r_0 := b - Ax_0$, $z_0 = M^{-1}r_0$, and $p_0 := z_0$
2. For $j = 0, 1, \dots$, until convergence Do:
3. $\alpha_j := (r_j, z_j) / (Ap_j, p_j)$
4. $x_{j+1} := x_j + \alpha_j p_j$
5. $r_{j+1} := r_j - \alpha_j Ap_j$
6. $z_{j+1} := M^{-1}r_{j+1}$
7. $\beta_j := (r_{j+1}, z_{j+1}) / (r_j, z_j)$
8. $p_{j+1} := z_{j+1} + \beta_j p_j$
9. EndDo

ALGORITHM : 2. Conjugate Gradient with Split Preconditioner

1. Compute $r_0 := b - Ax_0$; $\hat{r}_0 = L^{-1}r_0$; and $p_0 := L^{-T}\hat{r}_0$.
2. For $j = 0, 1, \dots$, until convergence Do:
3. $\alpha_j := (\hat{r}_j, \hat{r}_j) / (Ap_j, p_j)$
4. $x_{j+1} := x_j + \alpha_j p_j$
5. $\hat{r}_{j+1} := \hat{r}_j - \alpha_j L^{-1}Ap_j$
6. $\beta_j := (\hat{r}_{j+1}, \hat{r}_{j+1}) / (\hat{r}_j, \hat{r}_j)$
7. $p_{j+1} := L^{-T}\hat{r}_{j+1} + \beta_j p_j$
8. EndDo

- ▶ The x_j 's produced by the above algorithm and PCG are identical (if same initial guess is used).

Question: What can we do in case M is defined only approximately? i.e., if it can vary from one step to the other?

Applications:

- ▶ Iterative techniques as preconditioners: Block-SOR, SSOR, Multigrid, etc..
- ▶ Chaotic relaxation type preconditioners (e.g., in a parallel computing environment)
- ▶ Mixing Preconditioners – mixing coarse mesh / fine mesh preconditioners.

ALGORITHM : 4. GMRES – with (right) Preconditioning

1. Start: Choose x_0 and a dimension m of the Krylov subspaces.

2. Arnoldi process:

- Compute $r_0 = b - Ax_0$, $\beta = \|r_0\|_2$ and $v_1 = r_0/\beta$.
- For $j = 1, \dots, m$ do
 - Compute $z_j := M^{-1}v_j$
 - Compute $w := Az_j$
 - for $i = 1, \dots, j$, do : $\left\{ \begin{array}{l} h_{i,j} := (w, v_i) \\ w := w - h_{i,j}v_i \end{array} \right\}$
 - $h_{j+1,1} = \|w\|_2$; $v_{j+1} = w/h_{j+1,1}$
- Define $V_m := [v_1, \dots, v_m]$ and $\bar{H}_m = \{h_{i,j}\}$.

3. Form the approximate solution: Compute $x_m = x_0 + M^{-1}V_m y_m$ where $y_m = \operatorname{argmin}_y \|\beta e_1 - \bar{H}_m y\|_2$ and $e_1 = [1, 0, \dots, 0]^T$.

4. Restart: If satisfied stop, else set $x_0 \leftarrow x_m$ and goto 2.

ALGORITHM : 3. GMRES – No preconditioning

1. Start: Choose x_0 and a dimension m of the Krylov subspaces.

2. Arnoldi process:

- Compute $r_0 = b - Ax_0$, $\beta = \|r_0\|_2$ and $v_1 = r_0/\beta$.
- For $j = 1, \dots, m$ do
 - Compute $w := Av_j$
 - for $i = 1, \dots, j$, do $\left\{ \begin{array}{l} h_{i,j} := (w, v_i) \\ w := w - h_{i,j}v_i \end{array} \right\}$;
 - $h_{j+1,1} = \|w\|_2$; $v_{j+1} = \frac{w}{h_{j+1,1}}$
- Define $V_m := [v_1, \dots, v_m]$ and $\bar{H}_m = \{h_{i,j}\}$.

3. Form the approximate solution: Compute $x_m = x_0 + V_m y_m$ where $y_m = \operatorname{argmin}_y \|\beta e_1 - \bar{H}_m y\|_2$ and $e_1 = [1, 0, \dots, 0]^T$.

4. Restart: If satisfied stop, else set $x_0 \leftarrow x_m$ and goto 2.

ALGORITHM : 5. GMRES – with variable Preconditioning

1. Start: Choose x_0 and a dimension m of the Krylov subspaces.

2. Arnoldi process:

- Compute $r_0 = b - Ax_0$, $\beta = \|r_0\|_2$ and $v_1 = r_0/\beta$.
- For $j = 1, \dots, m$ do
 - Compute $z_j := M_j^{-1}v_j$; Compute $w := Az_j$;
 - for $i = 1, \dots, j$, do: $\left\{ \begin{array}{l} h_{i,j} := (w, v_i) \\ w := w - h_{i,j}v_i \end{array} \right\}$;
 - $h_{j+1,1} = \|w\|_2$; $v_{j+1} = w/h_{j+1,1}$
- Define $Z_m := [z_1, \dots, z_m]$ and $\bar{H}_m = \{h_{i,j}\}$.

3. Form the approximate solution: Compute $x_m = x_0 + Z_m y_m$ where $y_m = \operatorname{argmin}_y \|\beta e_1 - \bar{H}_m y\|_2$ and $e_1 = [1, 0, \dots, 0]^T$.

4. Restart: If satisfied stop, else set $x_0 \leftarrow x_m$ and goto 2.

Properties

- x_m minimizes $b - Ax_m$ over $\text{Span}\{Z_m\}$.
- If $Az_j = v_j$ (i.e., if preconditioning is 'exact' at step j) then approximation x_j is exact.
- If M_j is constant then method is \equiv to Right-Preconditioned GMRES.

Additional Costs:

- Arithmetic: none.
- Memory: Must save the additional set of vectors $\{z_j\}_{j=1,\dots,m}$

Advantage: Flexibility

Calais February 7, 2005

13

An observation. Introduction to Preconditioning

► Take a look back at basic relaxation methods: Jacobi, Gauss-Seidel, SOR, SSOR, ...

► These are iterations of the form $x^{(k+1)} = Mx^{(k)} + f$ where M is of the form $M = I - P^{-1}A$. For example for SSOR,

$$P_{SSOR} = (D - \omega E)D^{-1}(D - \omega F)$$

Calais February 7, 2005

15

Standard preconditioners

- Simplest preconditioner: $M = \text{Diag}(A)$ ► poor convergence.
- Next to simplest: SSOR.

$$M = (D - \omega E)D^{-1}(D - \omega F)$$

- Still simple but often more efficient: ILU(0).
- ILU(p) – ILU with level of fill p – more complex.
- Class of ILU preconditioners with threshold
- Class of approximate inverse preconditioners
- Class of Multilevel ILU preconditioners
- Algebraic Multigrid Preconditioners

Calais February 7, 2005

14

► SSOR attempts to solve the equivalent system

$$P^{-1}Ax = P^{-1}b$$

where $P \equiv P_{SSOR}$ by the fixed point iteration

$$x^{(k+1)} = \underbrace{(I - P^{-1}A)}_M x^{(k)} + P^{-1}b \quad \text{instead of} \quad x^{(k+1)} = (I - A)x^{(k)} + b$$

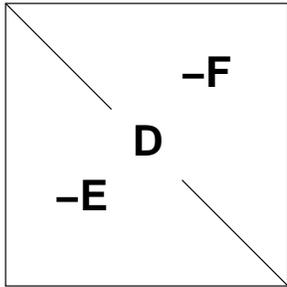
In other words:

Relaxation Scheme \iff Preconditioned Fixed Point Iteration

Calais February 7, 2005

16

The SOR/SSOR preconditioner



- ▶ SOR preconditioning

$$M_{SOR} = (D - \omega E)$$

- ▶ SSOR preconditioning

$$M_{SSOR} = (D - \omega E)D^{-1}(D - \omega F)$$

- ▶ $M_{SSOR} = LU$, L = lower unit matrix, U = upper triangular. One solve with $M_{SSOR} \approx$ same cost as a MAT-VEC.

- ▶ k -step SOR (resp. SSOR) preconditioning:

k steps of SOR (resp. SSOR)

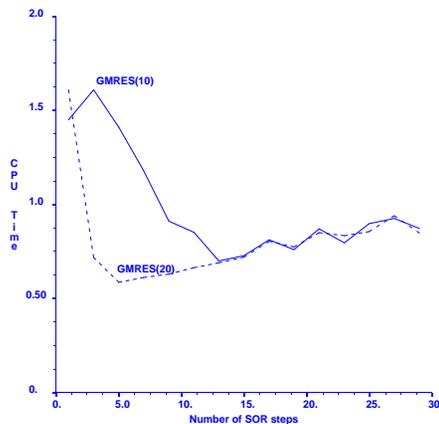
- ▶ Questions: Best ω ? For preconditioning can take $\omega = 1$

$$M = (D - E)D^{-1}(D - F)$$

Observe: $M = LU + R$ with $R = ED^{-1}F$.

- ▶ Best k ? $k = 1$ is rarely the best. Substantial difference in performance.

Iteration times versus k for SOR(k) preconditioned GMRES



ILU(0) and IC(0) preconditioners

- ▶ **Notation:** $NZ(X) = \{(i, j) \mid X_{i,j} \neq 0\}$

- ▶ Formal definition of ILU(0):

$$\begin{aligned} A &= LU + R \\ NZ(L) \cup NZ(U) &= NZ(A) \\ r_{ij} &= 0 \text{ for } (i, j) \in NZ(A) \end{aligned}$$

- ▶ This does not define ILU(0) in a unique way.

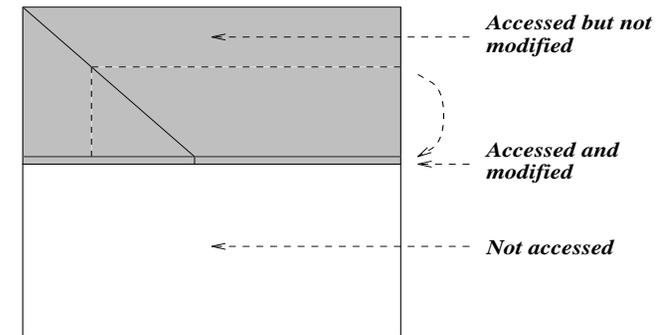
Constructive definition: Compute the LU factorization of A but drop any fill-in in L and U outside of $\text{Struct}(A)$.

- ▶ ILU factorizations are often based on i, k, j version of GE.

What is the IKJ version of GE?

ALGORITHM : 6. Gaussian Elimination – IKJ Variant

1. **For** $i = 2, \dots, n$ **Do**:
2. **For** $k = 1, \dots, i - 1$ **Do**:
3. $a_{ik} := a_{ik}/a_{kk}$
4. **For** $j = k + 1, \dots, n$ **Do**:
5. $a_{ij} := a_{ij} - a_{ik} * a_{kj}$
6. **EndDo**
7. **EndDo**
8. **EndDo**



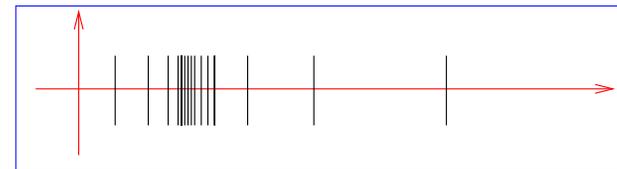
ILU(0) – zero-£ll ILU

ALGORITHM : 7. ILU(0)

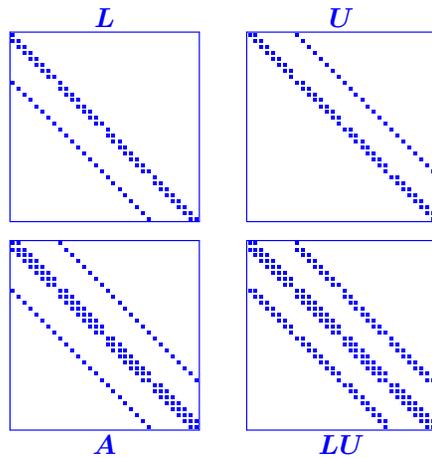
- For** $i = 1, \dots, N$ **Do**:
- For** $k = 1, \dots, i - 1$ **and if** $(i, k) \in NZ(A)$ **Do**:
- Compute** $a_{ik} := a_{ik}/a_{kj}$
- For** $j = k + 1, \dots$ **and if** $(i, j) \in NZ(A)$, **Do**:
- compute** $a_{ij} := a_{ij} - a_{ik}a_{kj}$.
- EndFor**
- EndFor**

► When A is SPD then the ILU factorization = Incomplete Choleski factorization – IC(0). Meijerink and Van der Vorst [1977].

Typical eigenvalue distribution

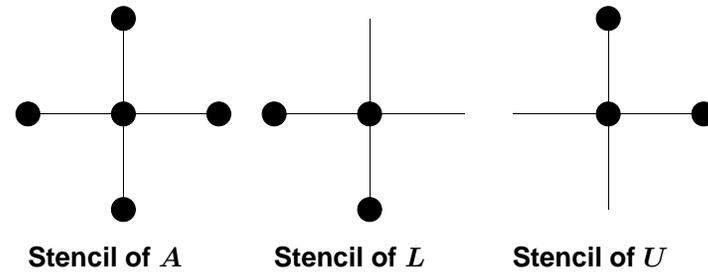


Pattern of ILU(0) for 5-point matrix

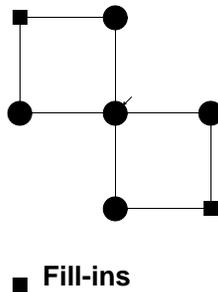


Stencils and ILU factorization

Stencils of A and the L and U parts of A :



Stencil of the product LU :



Higher order ILU factorization

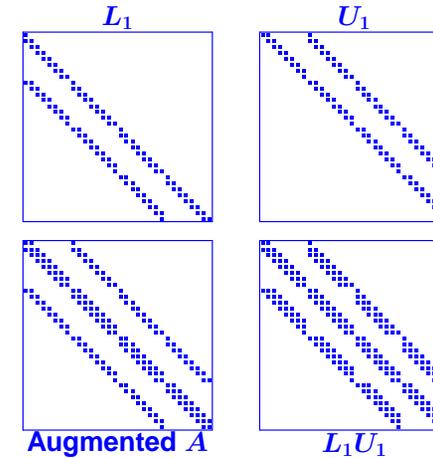
- Higher accuracy incomplete Choleski: for regularly structured problems, IC(p) allows p additional diagonals in L .
- Can be generalized to irregular sparse matrices using the notion of level of fill-in [Watts III, 1979]

- Initially $Lev(a_{ij}) = \begin{cases} 0 & \text{for } a_{ij} \neq 0 \\ \infty & \text{for } a_{ij} = 0 \end{cases}$
- At a given step i of Gaussian elimination:

$$Lev(a_{kj}) = \min\{Lev(a_{kj}); Lev(a_{ki}) + Lev(a_{ij}) + 1\}$$

- ▶ ILU(p) Strategy = drop anything with level of $\mathbb{E}||\text{-in}$ exceeding p .
- * Increasing level of $\mathbb{E}||\text{-in}$ usually results in more accurate ILU and...
- * ...typically in fewer steps and fewer arithmetic operations.

ILU(1)



ALGORITHM : 8. ILU(p)

For $i = 2, N$ Do

For each $k = 1, \dots, i - 1$ and if $a_{ij} \neq 0$ do

Compute $a_{ik} := a_{ik}/a_{kj}$

Compute $a_{i,*} := a_{i,*} - a_{ik}a_{k,*}$

Update the levels of $a_{i,*}$

Replace any element in row i with $lev(a_{ij}) > p$ by zero.

EndFor

EndFor

- ▶ The algorithm can be split into a symbolic and a numerical phase.
- Level-of- $\mathbb{E}||$ ▶ in Symbolic phase

ILU with threshold – generic algorithms

ILU(p) factorizations are based on structure only and not numerical values ▶ potential problems for non M-matrices.

- ▶ One remedy: ILU with threshold – (generic name ILUT.)

Two broad approaches:

First approach [derived from direct solvers]: use any (direct) sparse solver and incorporate a dropping strategy. [Munksgaard (?), Osterby & Zlatev, Sameh & Zlatev[90], D. Young, & al. (Boeing) etc...]

Second approach: [derived from 'iterative solvers' viewpoint]

1. use a (row or column) version of the (i, k, j) version of GE;
2. apply a drop strategy for the element l_{ik} as it is computed;
3. perform the linear combinations to get a_{i*} . Use full row expansion of a_{i*} ;
4. apply a drop strategy to ϵ II-ins.

Restarting methods for linear systems

Motivation: Goal: to use the information generated from current GMRES loop to improve convergence at next GMRES restart.

References:

- ▶ R. A. Nicolaides (87): Deflated CG.
- ▶ R. Morgan (92) Deflated GMRES
- ▶ S. Kharchenko & A. Yeregin (92) pole placement ideas.
- ▶ K. Burrage, J. Ehrel, and B. Pohl (93): Deflated GMRES
- ▶ E de Sturler: use SVD information in GMRES.

ILU with threshold: $ILUT(k, \epsilon)$

- Do the i, k, j version of Gaussian Elimination (GE).
 - During each i -th step in GE, discard any pivot or ϵ II-in whose value is below $\epsilon \|row_i(A)\|$.
 - Once the i -th row of $L + U$, (L-part + U-part) is computed retain only the k largest elements in both parts.
- ▶ Advantages: controlled ϵ II-in. Smaller memory overhead.
 - ▶ Easy to implement – much more so than preconditioners derived from direct solvers.
 - ▶ can be made quite inexpensive.

- ▶ Can help improve convergence and prevent stagnation of GMRES in some cases.

Generally speaking: One should not expect to solve very hard problems with Eigenvalue Deflation Preconditioning alone.

- ▶ Question: Can the same effects be achieved with block-Krylov methods?

Using the Flexible GMRES framework

Method: Deflation can be achieved by 'enriching' the Krylov subspace with approximate eigenvectors obtained from previous runs. We can use Flexible GMRES and append these vectors at end. [See R. Morgan (92), Chapman & YS (95).]

- ▶ Vectors v_1, \dots, v_{m-p} = standard Arnoldi vectors
- ▶ Vectors v_{m-p+1}, \dots, v_m = Computed as in FGMRES where new vectors z_j are previously computed eigenvectors.
- ▶ Storage: we need to store v_1, \dots, v_m and z_{m-p+1}, \dots, z_m . ▶ p additional vectors, with typically $p \ll m$.

Question 1: which eigenvectors to add?

- ▶ Answer: those associated with smallest eigenvalues.

Question 2: How to compute eigenvectors from the Flexible GMRES step? ▶ Answer: use the relation

$$AZ_m = V_{m+1}\bar{H}_m$$

Approximation: $\lambda, \tilde{u} = Z_m y$

Galerkin Condition: $r \perp AZ_m$ gives the generalized problem

$$\bar{H}_m^H \bar{H}_m y = \lambda \bar{H}_m^H V_{m+1}^H Z_m y$$

In Addition in GMRES: $\bar{H}_m = Q_m \bar{R}_m$ so $H_m^H H_m = R_m^H R_m$.

See: Morgan (1993).

GMRES with deflation

1. Deflated Arnoldi process: $r_0 := b - Ax_0, v_1 := r_0 / (\beta := \|r_0\|_2)$.

For $j = 1, \dots, m$ do

If $j < m - p$ then $z_j := v_j$ Else $z_j = u_{j-m}$ (eigenvector)

$w = Az_j$

For $i = 1, \dots, j$, do $\begin{cases} h_{i,j} := (w, v_i) \\ w := w - h_{i,j} v_i \end{cases}$

$h_{j+1,j} = \|w\|_2, v_{j+1} = w / \|w\|_2$.

EndDo

Define $Z_m := [z_1, \dots, z_m]$ and $\bar{H}_m = \{h_{i,j}\}$.

2. Form the approximate solution:

Compute $x_m = x_0 + Z_m y_m$ where $y_m = \operatorname{argmin}_y \|\beta e_1 - \bar{H}_m y\|_2$.

3. Get next eigenvector estimates u_1, \dots, u_p from $\bar{H}_m, V_m, Z_m, \dots$

4. Restart: If satisfied stop, else set $x_0 \leftarrow x_m$ and goto 1.

An example: Shell problems

- ▶ Can be very hard to solve!
- ▶ A matrix of size $N=38,002$, with $Nz = 949,452$ nonzero elements.
- ▶ Actually symmetric. Not exploited in test.
- ▶ Most simplistic methods fail.
- ▶ ILUT(50,0) does not work even with GMRES(80).
- ▶ This is an example when a large subspace is required.

An example

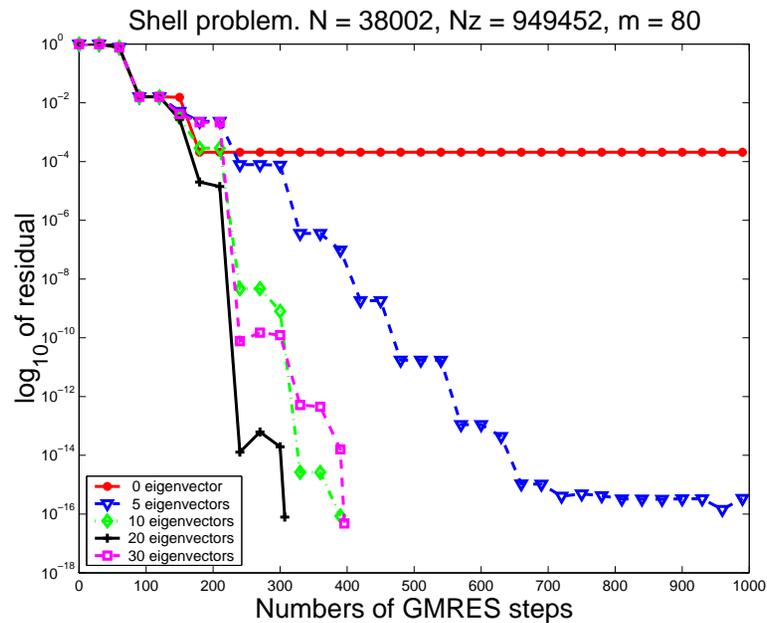
A matrix arising from Euler's equations on unstructured mesh

▶ Contributed by Larry Wigton from Boeing

Size = 3,864. (966 mesh points).

Nonzero elements: 238,252 (about 62 per row).

▶ Difficult to solve in spite of its small size.



▶ Results with $ILUT(lfil, \epsilon)$

$l\&E$	Iterations ($tol = 10^{-8}$)	estimate of $\ (LU)^{-1}\ $
100	*	$0.19E + 56$
110	*	$0.34E + 9$
120	30	$0.70E + 5$
130	25	$0.33E + 7$
140	20	$0.17E + 4$
150	19	$0.69E + 4$

Results with Block Jacobi Preconditioning
with Eigenvalue Deviation

reduction in residual norm in 1200 GMRES steps with $m = 49$		
	4x4 block	16x16 block
$p = 0$	0.8 E 0	0.8 E 0
$p = 4$	0.8 E 0	4.0 E-5
$p = 8$	1.2 E-2	2.9 E-7
$p = 12$	1.9 E-2	3.8 E-6

Theory – (Hermitian case only)

Assume that A is SPD and let $K = K_m + W$, where W is s.t.

$$\text{dist}(AW, U) = \epsilon$$

with $U =$ exact invariant subspace associated with $\lambda_1, \dots, \lambda_s$. Then the residual \tilde{r} obtained from the minimal residual projection process onto the augmented Krylov subspace K satisfies the inequality,

$$\|\tilde{r}\|_2 \leq \|r_0\|_2 \sqrt{\frac{1}{T_m^2(\gamma)} + \epsilon^2}$$

where $\gamma \equiv \frac{\lambda_n + \lambda_{s+1}}{\lambda_n - \lambda_{s+1}}$, $T_m \equiv$ Chebyshev polyn. of deg. m of 1st kind.

► See [YS, SIMAX vol. 4, pp 43-66 (1997)] for other results.

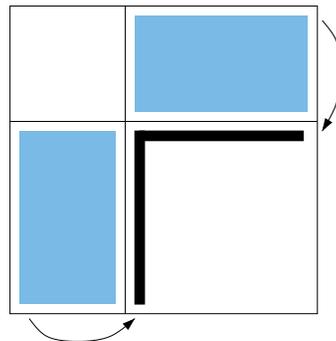
Crout-based ILUT (ILUTC)

Terminology: Crout versions of LU compute the k -th row of U and the k -th column of L at the k -th step.

Computational pattern

Black = part computed at step k

Blue = part accessed



Main advantages:

1. Less expensive than ILUT (avoids sorting)
2. Allows better techniques for dropping

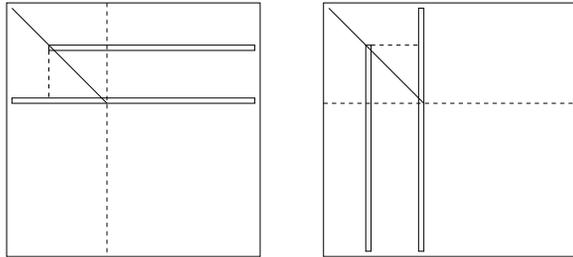
SPECIAL FORMS OF ILUS

References:

- [1] M. Jones and P. Plassman. An improved incomplete Choleski factorization. *ACM Transactions on Mathematical Software*, 21:5–17, 1995.
- [2] S. C. Eisenstat, M. H. Schultz, and A. H. Sherman. Algorithms and data structures for sparse symmetric Gaussian elimination. *SIAM Journal on Scientific Computing*, 2:225–237, 1981.
- [3] M. Bollhöfer. A robust ILU with pivoting based on monitoring the growth of the inverse factors. *Linear Algebra and its Applications*, 338(1–3):201–218, 2001.
- [4] N. Li, Y. Saad, and E. Chow. Crout versions of ILU. MSI technical report, 2002.

Crout LU (dense case)

► Go back to delayed update algorithm (IKJ alg.) and observe: we could do both a column and a row version



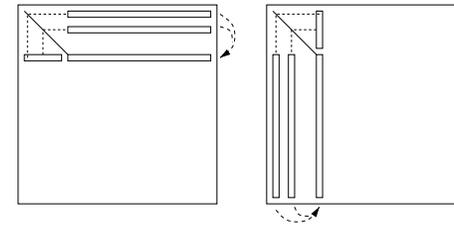
► Left: U computed by rows. Right: L computed by columns

Note: The entries $1 : k - 1$ in the k -th row in left figure need not be computed. Available from already computed columns of L . Similar observation for L (right).

ALGORITHM : 10. ILUC - Crout version of ILU

1. For $k = 1 : n$ Do :
2. Initialize row z : $z_{1:k-1} = 0$, $z_{k:n} = a_{k,k:n}$
3. For $\{i \mid 1 \leq i \leq k - 1 \text{ and } l_{ki} \neq 0\}$ Do :
4. $z_{k:n} = z_{k:n} - l_{ki} * u_{i,k:n}$
5. EndDo
6. Initialize column w : $w_{1:k} = 0$, $w_{k+1:n} = a_{k+1:n,k}$
7. For $\{i \mid 1 \leq i \leq k - 1 \text{ and } u_{ik} \neq 0\}$ Do :
8. $w_{k+1:n} = w_{k+1:n} - u_{ik} * l_{k+1:n,i}$
9. EndDo
10. Apply a dropping rule to row z
11. Apply a dropping rule to column w
12. $u_{k,:} = z$; $l_{:,k} = w / u_{kk}$, $l_{kk} = 1$
13. Enddo

► Notice that the updates to the k -th row of U (resp. the k -th column



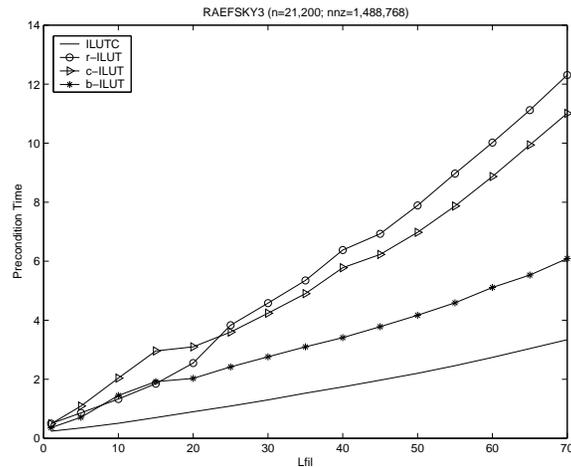
ALGORITHM : 9. Crout LU Factorization (dense case)

1. For $k = 1 : n$ Do :
2. For $i = 1 : k - 1$ and if $a_{ki} \neq 0$ Do :
3. $a_{k,k:n} = a_{k,k:n} - a_{ki} * a_{i,k:n}$
4. EndDo
5. For $i = 1 : k - 1$ and if $a_{ik} \neq 0$ Do :
6. $a_{k+1:n,k} = a_{k+1:n,k} - a_{ik} * a_{k+1:n,i}$
7. EndDo
8. $a_{ik} = a_{ik} / a_{kk}$ for $i = k + 1, \dots, n$
9. EndDo

of L) can be made in any order.

► Operations in Lines 4 and 8 are sparse vector updates (must be done in sparse mode)..

Comparison with standard techniques



Precondition time vs. Lfil for ILUC (solid), row-ILUT (circles), column-ILUT (triangles) and r-ILUT with Binary Search Trees (stars)

Calais February 7, 2005

53

- Can develop ILU versions based on this data structure.
- Advantages: (1) Savings when A has a symmetric structure. (2) Graceful degradation to an incomplete Choleski when A is symmetric (or nearly symmetric). (3) A little more convenient than ILUT for handling 'instability' of factorization.

Let $A_{k+1} = \begin{pmatrix} A_k & v_k \\ w_k & \alpha_k \end{pmatrix}$. If $A_k = L_k D_k U_k$ then

$$A_{k+1} = \begin{pmatrix} L_k & 0 \\ y_k & 1 \end{pmatrix} \begin{pmatrix} D_k & 0 \\ 0 & d_{k+1} \end{pmatrix} \begin{pmatrix} U_k & z_k \\ 0 & 1 \end{pmatrix}$$

$$z_k = D_k^{-1} L_k^{-1} v_k; \quad y_k = w_k U_k^{-1} D_k^{-1}; \quad d_{k+1} = \alpha_{k+1} - y_k D_k z_k$$

- To get next column z_k we need to solve a system with sparse L and sparse RHS v_k . Similarly with y_k .
- How can we approximate such systems inexpensively?

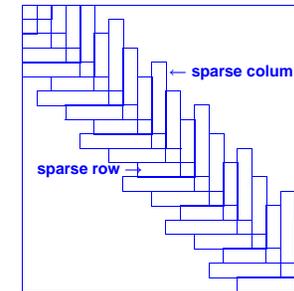
Note: Sparse RHS and sparse L

ILUS – ILU for Sparse Skyline format

- Often in CFD codes the matrices are generated in a format consisting of a sparse row representation of the decomposition

$$A = D + L_1 + L_2^T$$

where D is the strict lower part of A and L_1, L_2 are strict lower triangular.



- Simplest possibility: truncated Neumann series,

$$z_k = D_k^{-1} L_k^{-1} v_k = D_k^{-1} (I + E_k + E_k^2 + \dots + E_k^p) v_k$$

vector z_k gets denser as 'level-of-fill' p increases.

- We also use sparse-sparse mode GMRES.
- Idea of sparse-sparse mode computations is quite useful in developing preconditioners.

Calais February 7, 2005

56

Approximate Inverse preconditioners

Motivation:

- L - U solves in ILU may be 'unstable'
- Parallelism in L-U solves limited

Idea: Approximate the inverse of A directly $M \approx A^{-1}$

► Right preconditioning: Find M such that

$$AM \approx I$$

► Left preconditioning: Find M such that

$$MA \approx I$$

► Factored approximate inverse: Find L and U s.t.

$$LAU \approx D$$

One (of many) options:

try to find M to approximately minimize $\|I - AM\|_F$

Note:

$$\min \|I - AM\|_F^2 = \min \sum_{j=1}^n \|e_j - AMe_j\|_2^2 = \sum_{j=1}^n \min \|e_j - Am_j\|_2^2$$

- Problem decouples into n independent least-squares systems
- In each of these systems the matrix and RHS are sparse

Two paths:

1. Can find a good sparsity pattern for M first then compute M using this pattern.
2. Can find the pattern dynamically [similar to ILUT]

Some references

- Benson and Frederickson ('82): approximate inverse using stencils
- Grote and Simon ('93): Choose M to be banded
- Cosgrove, Díaz and Griewank ('91) : Procedure to add fill-ins to M
- Kolotilina and Yeregin ('93) : Factorized symmetric preconditionings $M = G_L^T G_L$
- Huckle and Grote ('95) : Procedure to find good pattern for M
- Chow and YS ('95): Find pattern dynamically by using dropping.
- M. Benzi & Tuma ('96, '97,..): Factored app. inv.

Approximate inverse with drop-tolerance [Chow & YS, 1994]

$$\text{Find } \min \|e_j - Am_j\|_2^2, \quad 1 \leq j \leq n$$

by solving approximately

$$Am_j = e_j, \quad 1 \leq j \leq n$$

with a few steps of GMRES, starting with a sparse m_j .

- Iterative method works in sparse mode: Krylov vectors are sparse.
- Use sparse-vector by sparse-vector and sparse-matrix by sparse-vector operations.
- Dropping strategy is applied on m_j .
- Exploit 'self-preconditioning'.

Sparse-Krylov MINRES and GMRES

- Dual threshold dropping strategy: drop tolerance and maximum number of nonzeros per column
- In MINRES, dropping performed on solution after each inner iteration
- In GMRES, dropping performed on Krylov basis at each iteration
- Use sparse-vector by sparse-vector operations

A few remarks

- ▶ There is no guarantee that M is nonsingular, unless the accuracy is high enough.
- ▶ There are many cases in which $APINV$ preconditioners work while ILU or ILUTP [with reasonable ϵ] won't
- ▶ The best use of $APINV$ preconditioners may be in combining them with other techniques. For example,

$$\text{Minimize } \|B - AM\|_F$$

where B is some other preconditioner (e.g. block-diagonal).

- ▶ Preconditioner for $A \rightarrow MB^{-1}$.

Self-preconditioning The system

$$Am_j = e_j$$

may be preconditioned with the current M . This is even more effective if the columns are computed in sequence.

- Actually use FGMRES
- Leads to *inner* and *outer* iteration approach
- Quadratic convergence if no dropping is done
- Effect of reordering?

Approximate inverses for block-partitioned matrices

Motivation. Domain Decomposition

$$\begin{pmatrix} B_1 & & & F_1 \\ & B_2 & & F_2 \\ & & \dots & \vdots \\ & & & B_n & F_n \\ E_1 & E_2 & \dots & E_n & C \end{pmatrix} \equiv \begin{pmatrix} B & F \\ E & D \end{pmatrix}$$

Note the factorization:
$$\begin{pmatrix} B & F \\ E & C \end{pmatrix} = \begin{pmatrix} B & 0 \\ E & S \end{pmatrix} \begin{pmatrix} I & B^{-1}F \\ 0 & I \end{pmatrix}$$

in which S is the Schur complement,

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

One idea: Compute $M = LU$ in which

$$L = \begin{pmatrix} B & 0 \\ E & M_S \end{pmatrix} \quad \text{and} \quad U = \begin{pmatrix} I & B^{-1}F \\ 0 & I \end{pmatrix}$$

► $M_S =$ some preconditioner to S .

One option: $M_S = \tilde{S} =$ sparse approximation to S

$$\tilde{S} = C - EY \quad \text{where} \quad Y \approx B^{-1}F$$

► Need to find a sparse matrix Y such that

$$BY \approx F$$

where F and B are sparse.

Incomplete Cholesky and SSOR for Normal Equations

► First Observation: IC(0) does not necessarily exist for SPD matrices.

► Can shift matrix: perform IC(0) on $AA^T + \alpha I$ for example. Hard to find good values of α for general matrices.

► Can modify dropping strategy: exploit relation between IC(0) and Incomplete Modified Gram Schmidt on $A \rightarrow$ ICMGS, [Wang & Gallivan, 1993]

Preconditioning the Normal Equations

► Why not solve

$$A^T A x = A^T b \quad \text{or} \quad A A^T y = b ?$$

► Advantage: symmetric positive definite systems

► Disadvantages:

- Worse conditioning
- Not easy to precondition.

► Generally speaking, the disadvantages outweigh the advantages.

► Can also get L from Incomplete LQ factorization [Saad, 1989]. Advantage: arbitrary accuracy. Disadvantage: need the Q factor.

► We never need to form the matrix $B = A^T A$ or $B = A A^T$ in implementation.

► Alternative: use SSOR [equivalent to Kacmarz algorithm]. No difficulties with shifts [Take $\omega = 1$], trivial to implement, no additional storage required.

Independent set orderings permute a matrix into the form

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

where B is a diagonal matrix.

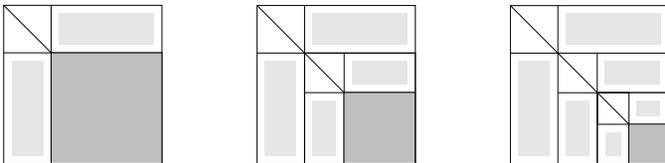
- ▶ Unknowns associated with the B block form an independent set (IS).
- ▶ IS is maximal if it cannot be augmented by other nodes to form another IS.
- ▶ IS ordering can be viewed as a “simplification” of multicoloring

ILUM AND ARMS

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
- ▶ Can devise an ILU factorization based on this strategy.



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

ALGORITHM : 11. ILUM

For $lev = 1, nlev$ *Do*

- a. *Get an independent set for A.*
- b. *Form the reduced system associated with this set;*
- c. *Apply a dropping strategy to this system;*
- d. *Set A := current reduced matrix and go back to (a).*

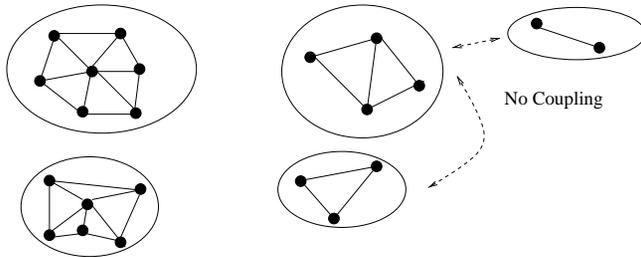
EndDo

Group Independent Sets / Aggregates

► Generalizes (common) Independent Sets

Main goal: to improve robustness

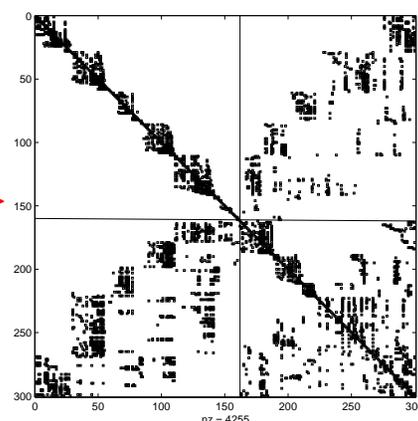
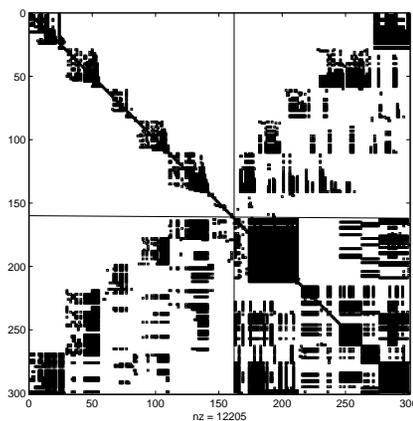
Main idea: use independent sets of “cliques”, or “aggregates”. There is no coupling between the aggregates.



► Reorder equations so nodes of independent sets come first

Problem: Fill-in

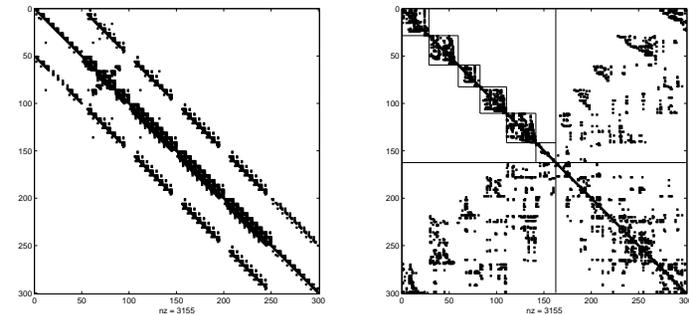
Remedy: dropping strategy



► Next step: treat the Schur complement recursively

Algebraic Recursive Multilevel Solver (ARMS)

Original matrix, A , and reordered matrix, $A_0 = P_0^T A P_0$.



► Block ILU factorizations (Diagonal blocks treated as sparse):

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

Algebraic Recursive Multilevel Solver (ARMS)

$$\begin{pmatrix} B & F \\ E & C \end{pmatrix} \begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix} \rightarrow \begin{pmatrix} L & 0 \\ E U^{-1} & I \end{pmatrix} \times \begin{pmatrix} U & L^{-1} F \\ 0 & S \end{pmatrix} \begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix}$$

where $S = C - E B^{-1} F =$ Schur complement.

- Idea: perform above block factorization recursively on S
- Blocks in B treated as sparse. Can be as large/small as desired.
- Algorithm is fully recursive
- Incorporates so-called W-cycles
- stability criterion added to block independent sets algorithm

Factorization:

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

- ▶ L-solve ~ restriction operation. U-solve ~ prolongation.
- ▶ Solve Last level system with, e.g., ILUT+GMRES

ALGORITHM : 12. ARMS(A_{lev}) factorization

1. If $lev = last_lev$ then
2. **Compute $A_{lev} \approx L_{lev} U_{lev}$**
3. Else:
4. **Find an independent set permutation P_{lev}**
5. Apply permutation $A_{lev} := P_{lev}^T A_{lev} P$
6. **Compute factorization**
7. Call ARMS(A_{lev+1})
8. Endif

Three options for inner-outer inter-level iterations

(VARMS) Descend, using the level-structure. At last level use GMRES-ILUT. Ascend back to the current level.

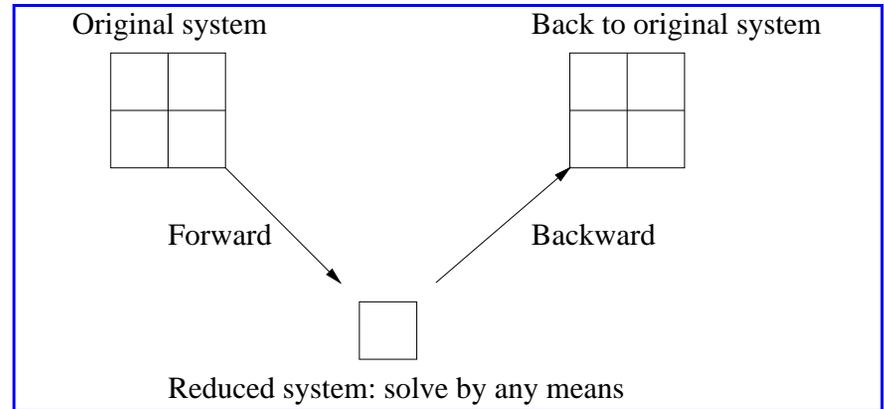
(WARMS) Use a few steps of GMRES+VARMS to solve the reduced system. At last level: use GMRES-ILUT.

(WARMS*) Use a few steps of FGMRES to solve the reduced system - Preconditioner: WARMS* (recursive). Last level: use ILUT-GMRES

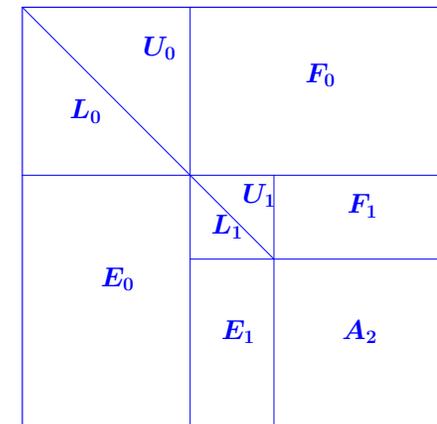
- ▶ WARMS* can be expensive! use with a small number of levels.
- ▶ Iterating allows to use less costly factorizations [memory]

Inner-Outer inter-level iterations

Idea: Use an iteration at level l to reduce residual norm by tol. τ



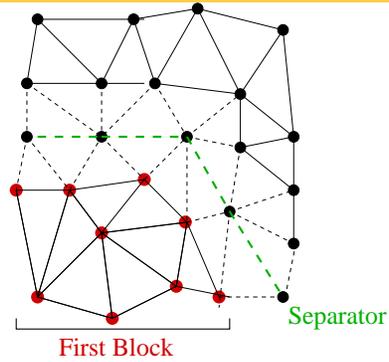
- ▶ Many possible variants.



Storage: ▶ At each level - except last, store: L_i, U_i, F_i, E_i

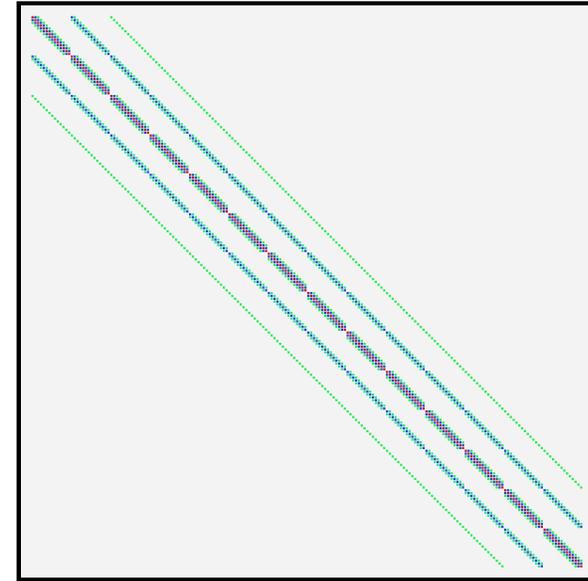
- ▶ For WARMS: need to multiply by intermediate A_i 's
- ▶ $A_{l+1} \times w$ computed as $(C_l - E_l U_l^{-1} L_l^{-1} F) \times w$ ▶ Need to store above 4 matrices + C_i .

Group Independent Set reordering

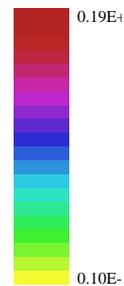
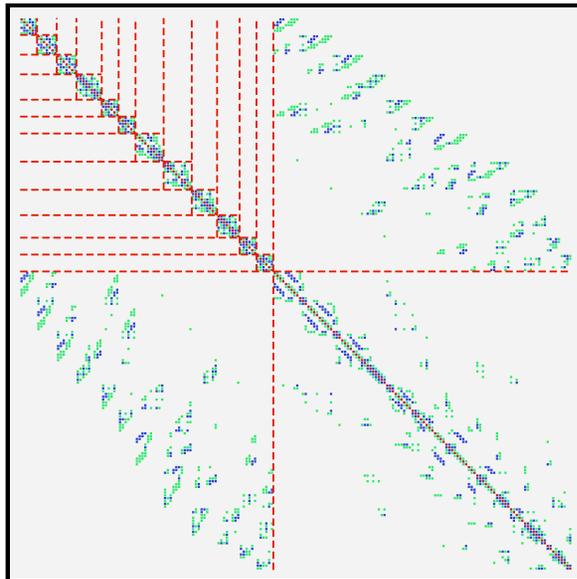


Simple strategy used: Do a Cuthill-MKee ordering until there are enough points to make a block. Reverse ordering. Start a new block from a non-visited node. Continue until all points are visited. Add criterion for rejecting "not sufficiently diagonally dominant rows."

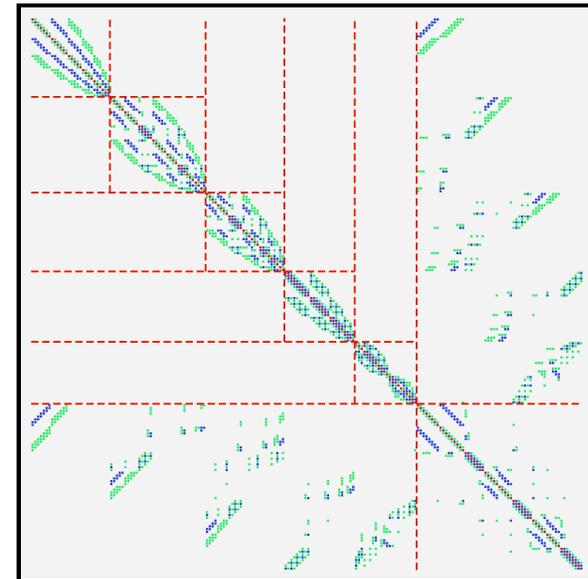
Original matrix



Block size of 6



Block size of 20



PARALLEL PRECONDITIONERS

Parallel preconditioners: A few approaches

“Parallel matrix computation” viewpoint:

- Local preconditioners: Polynomial (in the 80s), Sparse Approximate Inverses, [M. Benzi-Tuma & al '99., E. Chow '00]
- Distributed versions of ILU [Ma & YS '94, Hysom & Pothen '00]
- Use of multicoloring to unravel parallelism

Introduction

- ▶ In recent years: Big thrust of parallel computing techniques in applications areas. Problems becoming larger and harder
- ▶ In general: very large machines (e.g. Cray T3E) are gone. Exception: big US government labs.
- ▶ Replaced by ‘medium’ size machine (e.g. IBM SP2, SGI Origin)
- ▶ Programming model: Message-passing seems to be King (MPI)
- ▶ Open MP and threads for small number of processors
- ▶ Important new reality: parallel programming has penetrated the ‘applications’ areas [Sciences and Engineering + industry]

Domain Decomposition ideas:

- Schwarz-type Preconditioners [e.g. Widlund, Bramble-Pasciak-Xu, X. Cai, D. Keyes, Smith, ...]
- Schur-complement techniques [Gropp & Smith, Ferhat et al. (FETI), T.F. Chan et al., YS and Sosonkina '97, J. Zhang '00, ...]

Multigrid / AMG viewpoint:

- Multi-level Multigrid-like preconditioners [e.g., Shadid-Tuminaro et al (Aztec project), ...]
- ▶ In practice: Variants of additive Schwarz very common (simplicity)

Intrinsically parallel preconditioners

Some alternatives

- (1) Polynomial preconditioners;
- (2) Approximate inverse preconditioners;
- (3) Multi-coloring + independent set ordering;
- (4) Domain decomposition approach.

POLYNOMIAL PRECONDITIONING

Principle: $M^{-1} = s(A)$ where s is a (low) degree polynomial:

$$s(A)Ax = s(A)b$$

Problem: how to obtain s ? Note: $s(A) \approx A^{-1}$

- * Chebyshev polynomials
 - * Least squares polynomials
 - * Others
- Several approaches.
- Polynomial preconditioners are seldom used in practice.

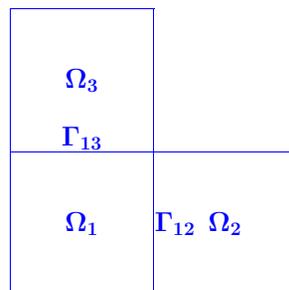
Domain Decomposition

Problem:

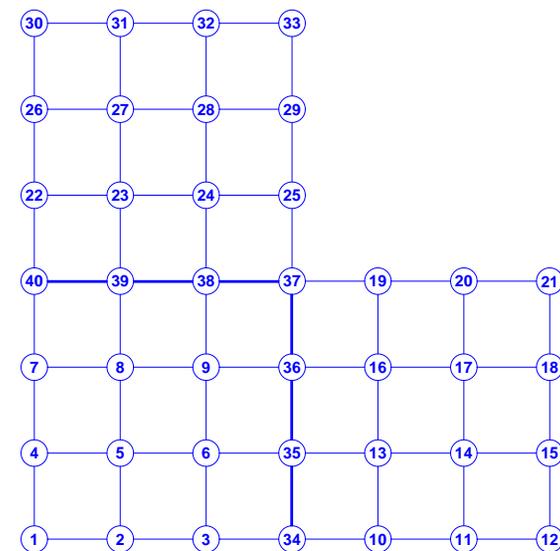
$$\begin{cases} \Delta u = f & \text{in } \Omega \\ u = u_\Gamma & \text{on } \Gamma = \partial\Omega. \end{cases}$$

Domain:

$$\Omega = \bigcup_{i=1}^s \Omega_i,$$

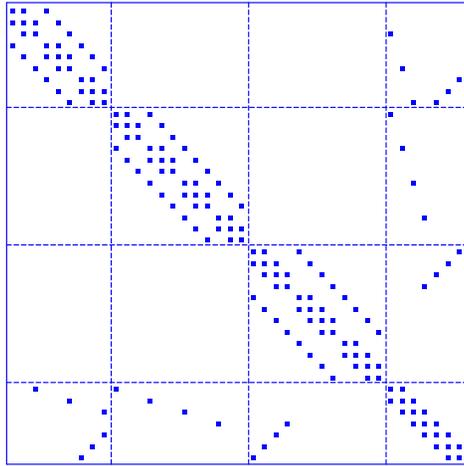


- Domain decomposition or substructuring methods attempt to solve a PDE problem (e.g.) on the entire domain from problem solutions on the subdomains Ω_i .

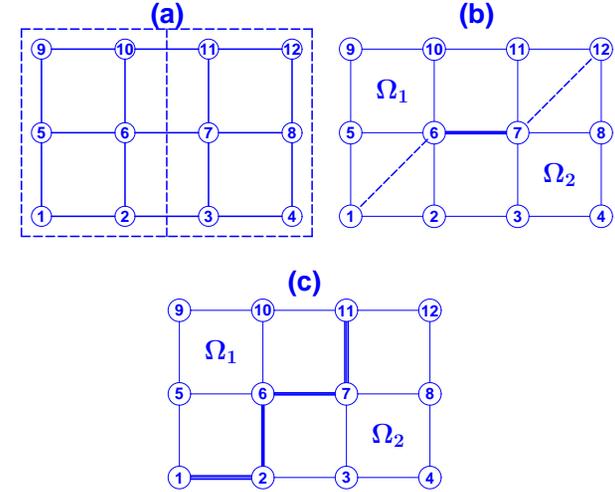


Discretization of domain

Types of mappings



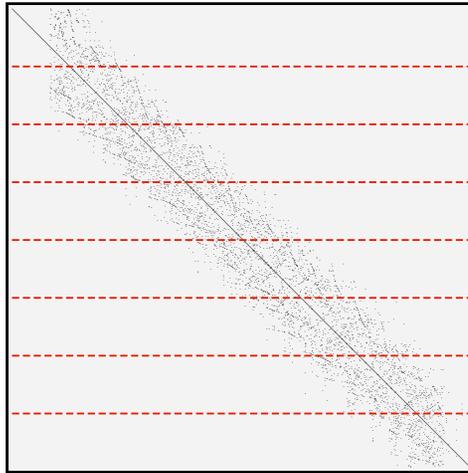
CoefEcient Matrix



(a) Vertex-based, (b) edge-based, and (c) element-based partitioning

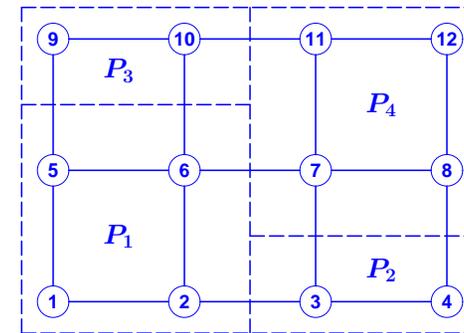
Generalization: Distributed Sparse Systems

- ▶ Simple illustration:
Block assignment
Assign equation i and unknown i to a given processor.



Partitioning a sparse matrix

- ▶ Use a graph partitioner to partition the adjacency graph:



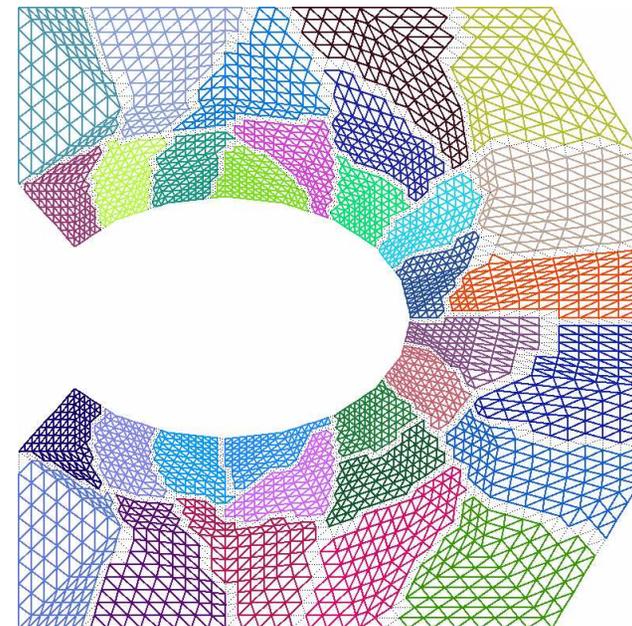
- ▶ Can allow overlap.
- ▶ Partition can be very general.

Given: a general mapping of pairs equation/unknown to processors. = { set of p subsets of variables }.

- ▶ Subsets can be arbitrary + allow 'overlap'. Mapping can be obtained by graph partitioners.

Problem: build local data structures needed for iteration phase.

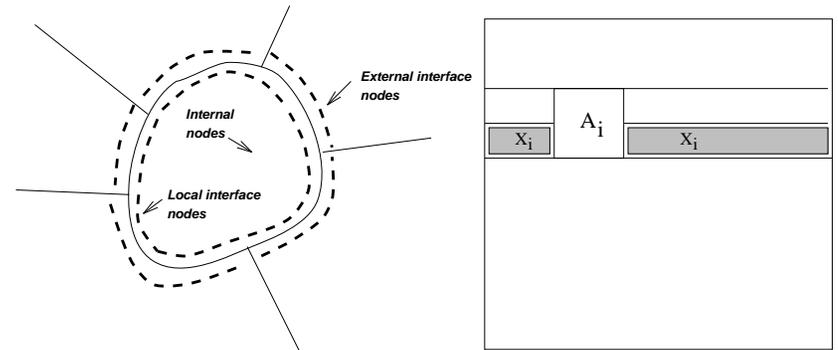
- ▶ Several graph partitioners available: Metis, Chaco, Scotch, ...



Distributed Sparse matrices (continued)

- Once a good partitioning is found, questions are:
1. How to represent this partitioning?
 2. What is a good data structure for representing distributed sparse matrices?
 3. How to set up the various “local objects” (matrices, vectors, ..)
 4. What can be done to prepare for communication that will be required during execution?

Two views of a distributed sparse matrix

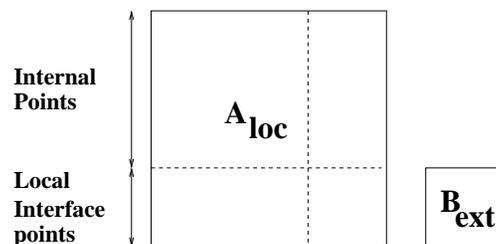


- Local interface variables always ordered last.

Local view of distributed matrix:



The local matrix:



Distributed Sparse Matrix-Vector Product Kernel

Algorithm:

1. Communicate: exchange boundary data.

Scatter x_{bound} to neighbors - Gather x_{ext} from neighbors

2. Local matrix – vector product

$$y = A_{loc}x_{loc}$$

3. External matrix – vector product

$$y = y + B_{ext}x_{ext}$$

NOTE: 1 and 2 are independent and can be overlapped.

Distributed Sparse Matrix-Vector Product

Main part of the code:

```
call MSG_bdx_send(nloc,x,y,nproc,proc,ix,ipr,ptrn,ierr)
c
c do local matrix-vector product for local points
c
c call amux(nloc,x,y,aloc,jaloc,ialoc)
c
c receive the boundary information
c
c call MSG_bdx_receive(nloc,x,y,nproc,proc,ix,ipr,ptrn,ie
c
c do local matrix-vector product for external points
c
c nrow = nloc - nbnd + 1
c call amux1(nrow,x,y(nbnd),aloc,jaloc,ialoc(nloc+1))
c
c return
```

The local exchange information

- ▶ List of adjacent processors (or subdomains)
- ▶ For each of these processors, lists of boundary nodes to be sent / received to /from adj. PE's.
- ▶ The receiving processor must have a matrix ordered consistently with the order in which data is received.

Requirements

- ▶ The 'set-up' routines should handle overlapping
- ▶ Should use minimal storage (only arrays of size nloc allowed).

Main Operations in (F) GMRES :

1. Saxpy's – local operation – no communication
2. Dot products – global operation
3. Matrix-vector products – local operation – local communication
4. Preconditioning operations – locality varies.

Distributed Dot Product

```
/*----- call blas function
   tloc = DDOT(n, x, incx, y, incy);
/*----- call global reduction
   MPI_Allreduce(&tloc,&ro,1,MPI_DOUBLE,MPI_SUM,comm);
```

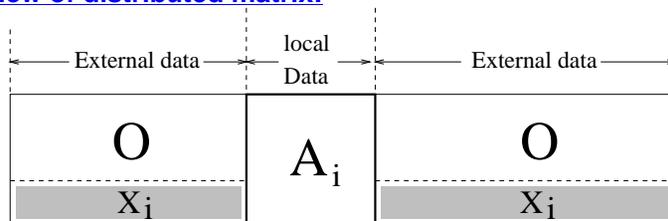
PARALLEL PRECONDITIONERS

Three approaches:

- Schwarz Preconditioners
 - Schur-complement based Preconditioners
 - Multi-level ILU-type Preconditioners
- **Observation:** Often, in practical applications, only Schwarz Preconditioners are used

Domain-Decomposition-Type Preconditioners

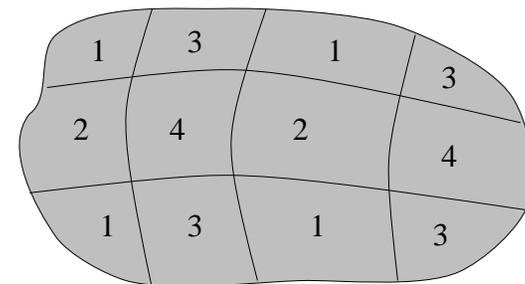
Local view of distributed matrix:



Block Jacobi Iteration (Additive Schwarz):

1. Obtain external data y_i
2. Compute (update) local residual $r_i = (b - Ax)_i = b_i - A_i x_i - B_i y_i$
3. Solve $A_i \delta_i = r_i$
4. Update solution $x_i = x_i + \delta_i$

- **Multiplicative Schwarz.** Need a coloring of the subdomains.



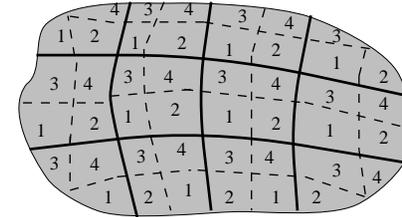
Multicolor Block SOR Iteration (Multiplicative Schwarz):

1. Do $col = 1, \dots, numcols$
2. If ($col.eq.mycol$) Then
3. Obtain external data y_i
4. Update local residual $r_i = (b - Ax)_i$
5. Solve $A_i \delta_i = r_i$
6. Update solution $x_i = x_i + \delta_i$
7. EndIf
8. EndDo

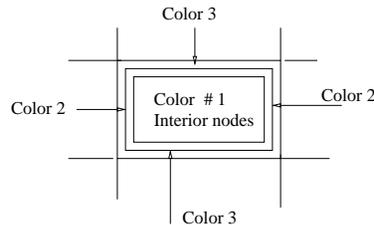
Breaking the sequential color loop

► “Color” loop is sequential. Can be broken in several different ways.

(1) Have a few subdomains per processors



(2) Separate interior nodes from interface nodes (2-level blocking)



(3) Use a block-GMRES algorithm - with Block-size = number of colors. SOR step targets a different color on each column of the block ► no idle time.

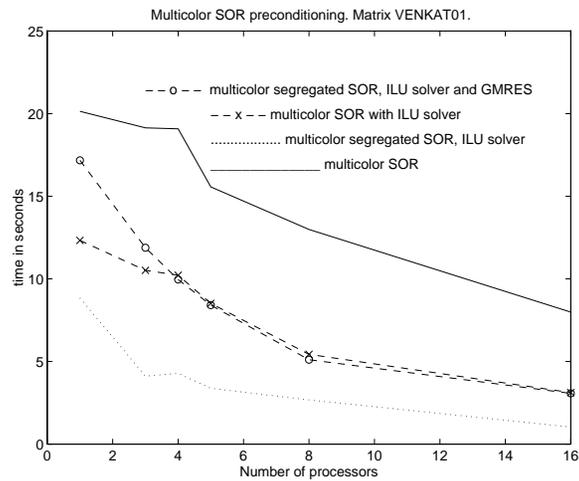
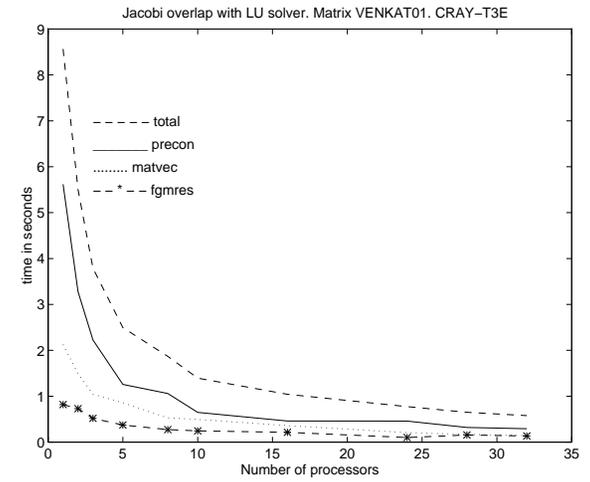
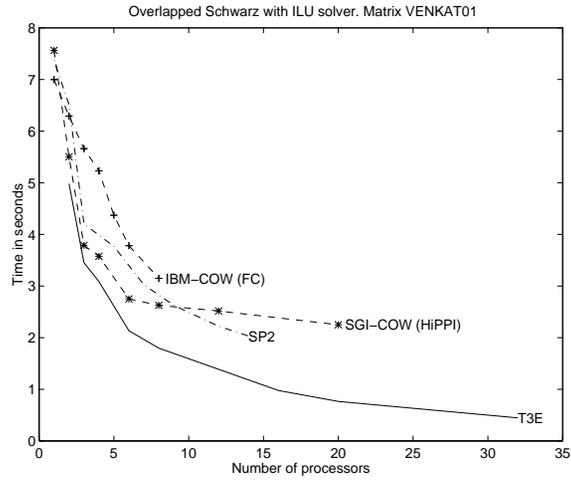
Local Solves

► Each local system $A_i \delta_i = r_i$ can be solved in three ways:

1. By a (sparse) direct solver
2. Using a standard preconditioned Krylov solver
3. Doing a backward-forward solution associated with an accurate ILU (e.g. ILUT) preconditioner

► We only use (2) with a small number of inner steps (up to 10) or (3).

Performance comparison for different machines



SCHUR COMPLEMENT-BASED PRECONDITIONERS

Schur complement system

Local system can be written as

$$A_i x_i + X_i y_{i,ext} = b_i. \quad (1)$$

x_i = vector of local unknowns, $y_{i,ext}$ = external interface variables, and b_i = local part of RHS.

► Local equations

$$\begin{pmatrix} B_i & F_i \\ E_i & C_i \end{pmatrix} \begin{pmatrix} u_i \\ y_i \end{pmatrix} + \begin{pmatrix} 0 \\ \sum_{j \in N_i} E_{ij} y_j \end{pmatrix} = \begin{pmatrix} f_i \\ g_i \end{pmatrix}. \quad (2)$$

► eliminate u_i from the above system:

$$S_i y_i + \sum_{j \in N_i} E_{ij} y_j = g_i - E_i B_i^{-1} f_i \equiv g'_i,$$

where S_i is the “local” Schur complement

$$S_i = C_i - E_i B_i^{-1} F_i. \quad (3)$$

Structure of Schur complement system

► Schur complement system

$$S y = g'$$

with

$$S = \begin{pmatrix} S_1 & E_{12} & \dots & E_{1p} \\ E_{21} & S_2 & \dots & E_{2p} \\ \vdots & & \ddots & \vdots \\ E_{p1} & E_{p-1,2} & \dots & S_p \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{pmatrix} = \begin{pmatrix} g'_1 \\ g'_2 \\ \vdots \\ g'_p \end{pmatrix}.$$

Simplest idea: Schur Complement Iterations

$$\begin{pmatrix} u_i \\ y_i \end{pmatrix} \begin{matrix} \text{Internal variables} \\ \text{Interface variables} \end{matrix}$$

► Do a global primary iteration (e.g., block-Jacobi)

► Then accelerate only the y variables (with a Krylov method)

Still need to precondition..

Approximate Schur-LU

► Two-level method based on induced preconditioner. Global system can also be viewed as

$$\begin{pmatrix} B & F \\ E & C \end{pmatrix} \begin{pmatrix} u \\ y \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix}, \quad \text{with } B = \left(\begin{array}{ccc|c} B_1 & & & F_1 \\ & B_2 & & F_2 \\ & & \cdots & \vdots \\ & & & B_p & F_p \\ \hline E_1 & E_2 & \cdots & E_p & C \end{array} \right)$$

Block LU factorization of A :

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

Two approaches:

(1) can compute approximation $\tilde{S}_i \approx S_i$ using approximate inverse techniques (M. Sosonkina)

(2) we can simply use LU factorization of A_i . Exploit the property:

$$\text{If } A_i = \begin{pmatrix} L_{B_i} & 0 \\ E_i U_{B_i}^{-1} & L_{S_i} \end{pmatrix} \begin{pmatrix} U_{B_i} & L_{B_i}^{-1} F_i \\ 0 & U_{S_i} \end{pmatrix} \quad \text{Then } L_{S_i} U_{S_i} = S_i$$

Preconditioning:

$$L = \begin{pmatrix} B & 0 \\ E & M_S \end{pmatrix} \quad \text{and} \quad U = \begin{pmatrix} I & B^{-1}F \\ 0 & I \end{pmatrix}$$

with $M_S =$ some approximation to S .

► Preconditioning to global system can be induced from any preconditioning on Schur complement.

Rewrite local Schur system as

$$y_i + S_i^{-1} \sum_{j \in N_i} E_{ij} y_j = S_i^{-1} [g_i - E_i B_i^{-1} f_i].$$

► equivalent to Block-Jacobi preconditioner for Schur complement.

► Solve with, e.g., a few steps (e.g., 5) of GMRES

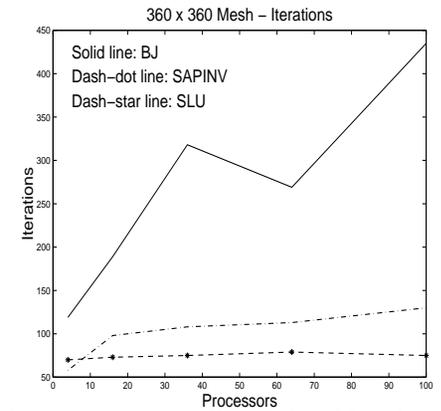
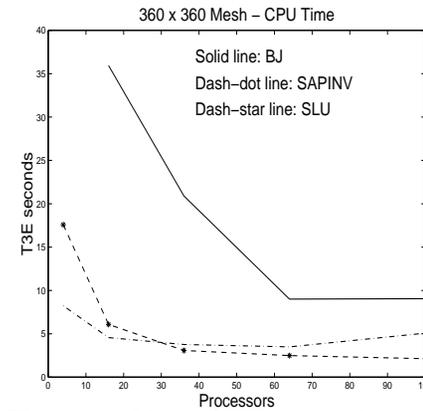
► Question: How to solve with S_i ?

Name	Precon	lfil	4	8	16	24	36	40
raefsky1	SAPINV	10	14	13	10	11	8	8
		20	12	11	9	9	8	8
	SAPINVS	10	16	13	10	11	8	8
		20	13	11	9	9	8	8
	SLU	10	215	197	198	194	166	171
		20	48	50	40	42	41	41
	BJ	10	85	171	173	273	252	263
		20	82	170	173	271	259	259

Number of FGMRES(20) iterations for the RAEFSKY1 problem.

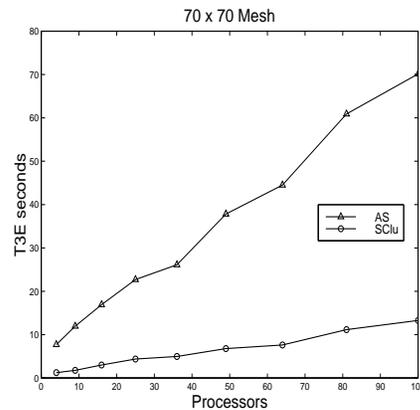
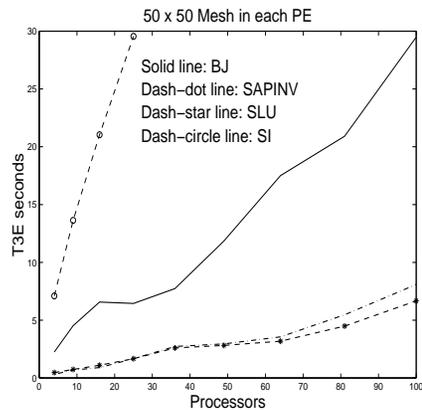
Name	Precon	lfil	16	24	32	40	56	64	80	96
af23560	SAPINV	20	32	36	27	29	73	35	71	61
		30	32	35	23	29	46	60	33	52
	SAPINVS	20	32	35	24	29	55	35	37	59
		30	32	34	23	28	43	45	23	35
	SLU	20	81	105	94	88	90	76	85	71
		30	38	34	37	39	38	39	38	35
	BJ	20	37	153	53	60	77	80	95	*
		30	36	41	53	57	81	87	97	115

Number of FGMRES(20) iterations for the AF23560 problem.



Times and iteration counts for solving a 360×360 discretized Laplacean problem with 3 different preconditioners using α exible GMRES(10).

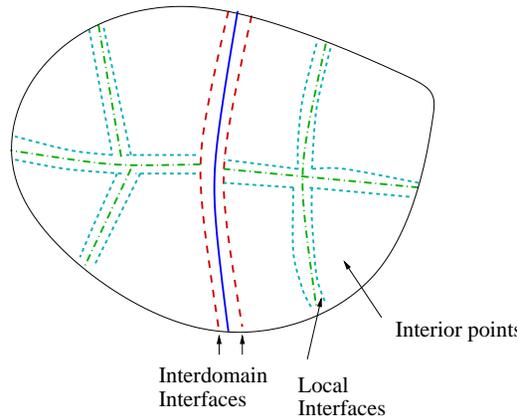
► Solution times for a Laplacean problem with various local sub-problem sizes using FGMRES(10) with 3 different preconditioners (BJ, SAPINV, SLU) and the Schur complement iteration (SI).



PARALLEL ARMS

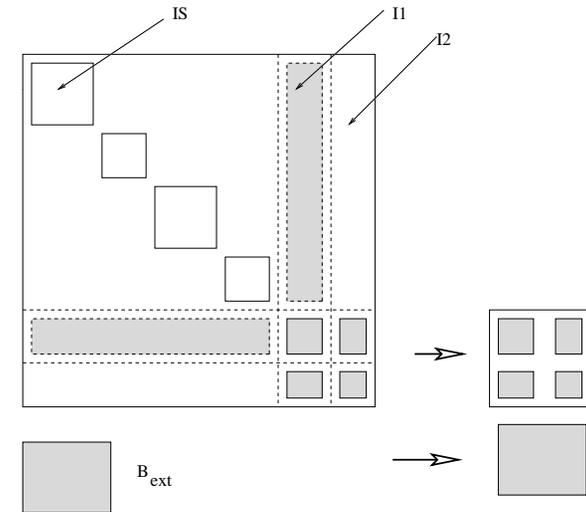
Parallel implementation of ARMS

Three types of points: interior (independent sets), local interfaces, and global interfaces



Main ideas: (1) exploit recursivity (2) distinguish two phases: elimination of interior points and then interface points.

Result: 2-part Schur complement: one corresponding to local interfaces and the other to inter-domain interfaces.



Three approaches

Method 1: Simple additive Schwarz using ILUT or ARMS locally

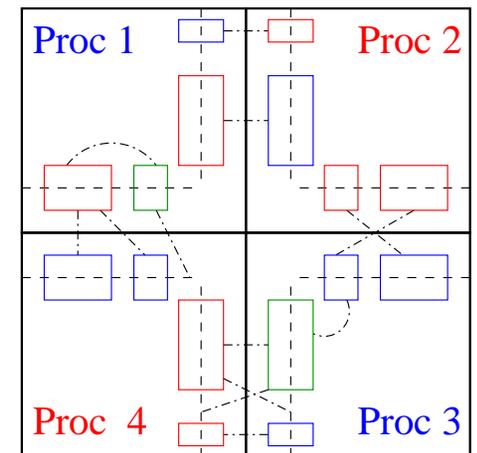
Method 2: Schur complement approach. Solve Schur complement system (both I1 and I2) with either a block Jacobi (M. Sosonkina and YS, '99) or multicolor ILU(0).

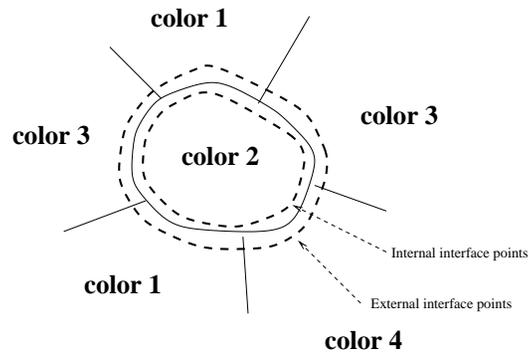
Method 3: Do independent set reduction across subdomains. Requires construction of global group independent sets.

► Current status Methods 1 and 2.

Construction of global group independent sets A two level strategy

1. Color subdomains
2. Find group independent sets locally
3. Color groups consistently





Algorithm: Multicolor Distributed ILU(0)

1. Eliminate local rows,
2. Receive external interf. rows from PEs s.t. $color(PE) < MyColor$
3. Process local interface rows
4. Send local interface rows to PEs s.t. $color(PE) > MyColor$

Test problem

1. Scalability experiment: sample finite difference problem.

$$-\Delta u + \gamma \left(e^{xy} \frac{\partial u}{\partial x} + e^{-xy} \frac{\partial u}{\partial y} \right) + \alpha u = f,$$

Dirichlet Boundary Conditions ; $\gamma = 100, \alpha = -10$; centered differences discretization.

► Keep size constant on each processor [100 × 100] ► Global linear system with 10,000 * *nproc* unknowns.

2. Comparison with a parallel direct solver – symmetric problems
3. Large irregular matrix example arising from magneto hydrodynamics.

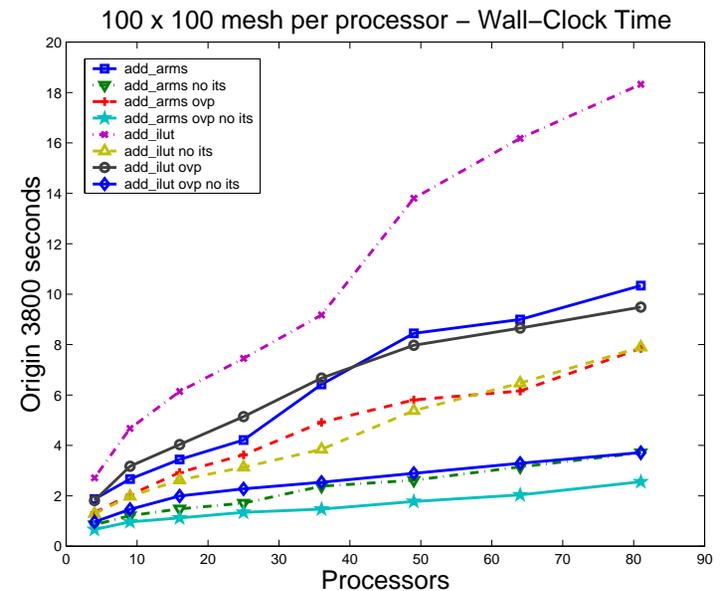
Methods implemented in pARMS:

add_x Additive Schwarz procedure, with method **x** for subdomains. With/out overlap. **x** is one of ILUT, ILUK, ARMS.

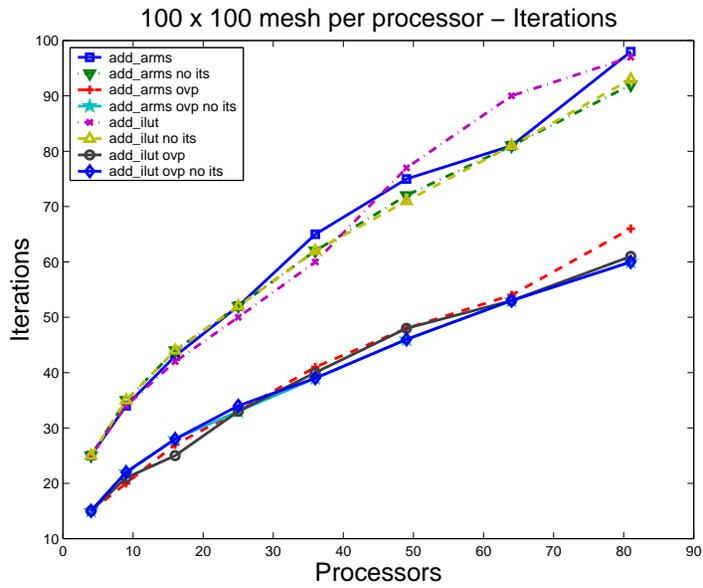
sch_x Schur complement technique, with method **x** = factorization used for local submatrix = {ILUT, ILUK, ARMS}. Equiv. to Additive Schwarz preconditioner on Schur complement.

sch_sgs Multicolor Multiplicative Schwarz (block Gauss-Seidel) preconditioning is used instead of additive Schwarz for Schur complement.

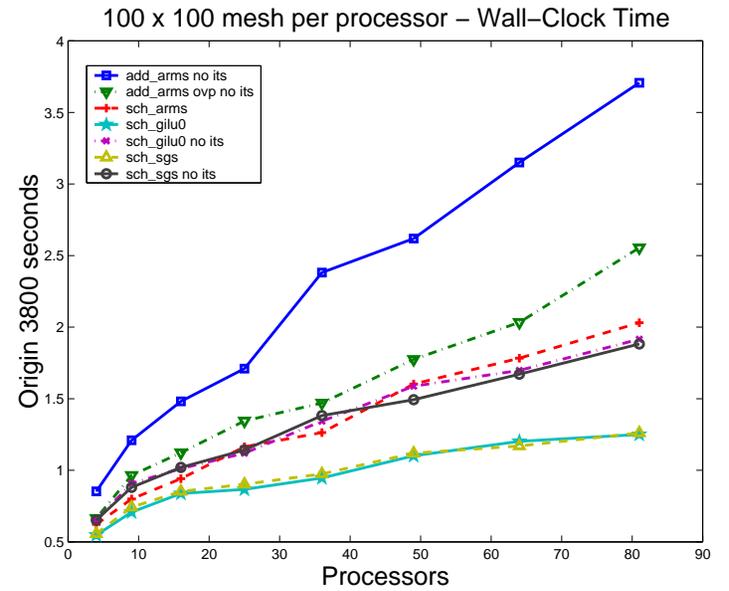
sch_gilu0 ILU(0) preconditioning is used for solving the global Schur complement system obtained from the ARMS reduction.



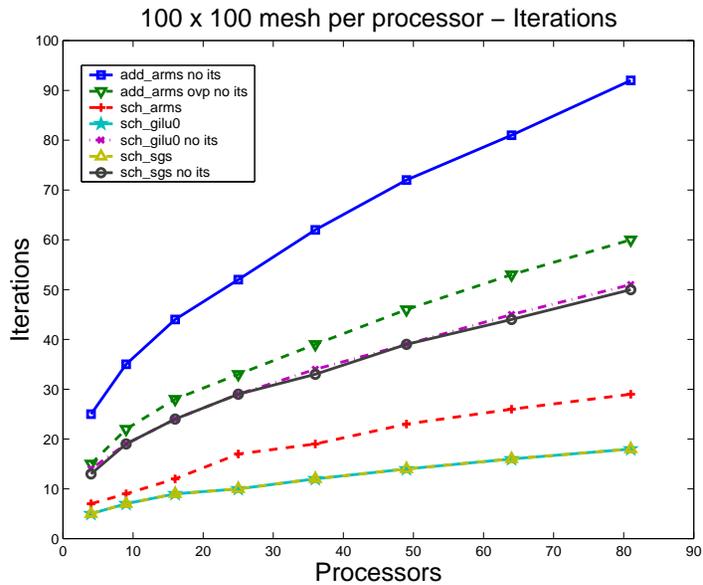
Solution times for 2D PDE problem with fixed subproblem size



Iterations for 2D PDE problem with £xed subproblem size



Solution times for a 2D PDE problem with the £xed subproblem size using different preconditioners.



Iterations for a 2D PDE problem with the £xed subproblem size using different preconditioners.

