A tutorial on:
Iterative methods for Sparse Matrix Problems

Yousef Saad
University of Minnesota
Computer Science and Engineering

CRM Montreal - May 2, 2008
Outline

Part 1
- Sparse matrices and sparsity
- Basic iterative techniques
- Projection methods
- Krylov subspace methods

Part 2
- Preconditioned iterations
- Preconditioning techniques

Part 3
- Parallel implementations
- Multigrid methods

Part 4
- Eigenvalue problems
- Applications
MULTILEVEL PRECONDITIONING
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.
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, ..]
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
Original matrix, $A$, and reordered matrix, $A_0 = P_0^T A P_0$.

Block ILU factorization of $A_l$

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

Problem: Fill-in

Remedy: dropping strategy

Next step: treat the Schur complement recursively
Basic step:

\[
\begin{pmatrix}
B & F \\
E & C
\end{pmatrix}
\begin{pmatrix}
y \\
z
\end{pmatrix} =
\begin{pmatrix}
f \\
g
\end{pmatrix} \quad \rightarrow
\]

\[
\begin{pmatrix}
L & 0 \\
EU^{-1} & I
\end{pmatrix}
\times
\begin{pmatrix}
U & L^{-1}F \\
0 & S
\end{pmatrix}
\begin{pmatrix}
y \\
z
\end{pmatrix} =
\begin{pmatrix}
f \\
g
\end{pmatrix}
\]

where \( S = C - EB^{-1}F = \text{Schur complement} \).

- Perform block factorization recursively on \( S \)
- \( L, U \) Blocks: sparse
- Exploit recursivity
Factorization: at level $l$ \[ P_l^T A_l P_l = \]
\[
\begin{pmatrix}
B_l & F_l \\
E_l & C_l
\end{pmatrix}
\approx
\begin{pmatrix}
L_l & 0 \\
E_l U_l^{-1} & I
\end{pmatrix}
\begin{pmatrix}
I & 0 \\
0 & A_{l+1}
\end{pmatrix}
\begin{pmatrix}
U_l & L_l^{-1} F_l \\
0 & I
\end{pmatrix}
\]

- L-solve $\sim$ restriction. U-solve $\sim$ prolongation.

- Solve Last level system with, e.g., ILUT+GMRES
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
**ARMS with permutations for 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

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

$P, Q$ 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).
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 \(M\).

Matrix after preselection

Matrix after Matching perm.
### Numerical illustration

<table>
<thead>
<tr>
<th>Matrix</th>
<th>order</th>
<th>nonzeros</th>
<th>Application (Origin)</th>
</tr>
</thead>
<tbody>
<tr>
<td>barrier2-9</td>
<td>115,625</td>
<td>3,897,557</td>
<td>Device simul. (Schenk)</td>
</tr>
<tr>
<td>matrix_9</td>
<td>103,430</td>
<td>2,121,550</td>
<td>Device simul. (Schenk)</td>
</tr>
<tr>
<td>mat-n_3*</td>
<td>125,329</td>
<td>2,678,750</td>
<td>Device simul. (Schenk)</td>
</tr>
<tr>
<td>ohne2</td>
<td>181,343</td>
<td>11,063,545</td>
<td>Device simul. (Schenk)</td>
</tr>
<tr>
<td>para-4</td>
<td>153,226</td>
<td>5,326,228</td>
<td>Device simul. (Schenk)</td>
</tr>
<tr>
<td>cir2a</td>
<td>482,969</td>
<td>3,912,413</td>
<td>circuit simul.</td>
</tr>
<tr>
<td>scircuit</td>
<td>170998</td>
<td>958936</td>
<td>circuit simul. (Hamm)</td>
</tr>
<tr>
<td>circuit_4</td>
<td>80209</td>
<td>307604</td>
<td>Circuit simul. (Bomhof)</td>
</tr>
<tr>
<td>wang3.rua</td>
<td>26064</td>
<td>177168</td>
<td>Device simul. (Wang)</td>
</tr>
<tr>
<td>wang4.rua</td>
<td>26068</td>
<td>177196</td>
<td>Device simul. (Wang)</td>
</tr>
</tbody>
</table>
## Parameters

<table>
<thead>
<tr>
<th>$n_{lev_{max}}$</th>
<th>$tol_{DD}$</th>
<th>Drop tolerance</th>
<th>$Fill_{max}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>0.1</td>
<td>LU-B 0.01 0.01 0.01</td>
<td>LU-B 3</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>GW S 1.e-05</td>
<td>GW S 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LU-S 1.e-05</td>
<td>LU-S 20</td>
</tr>
<tr>
<td>Matrix</td>
<td>Fill Factor</td>
<td>Set-up Time</td>
<td>GMRES(60)</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>-------------</td>
<td>-----------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Its.</td>
</tr>
<tr>
<td>barr2-9</td>
<td>0.62</td>
<td>4.01e+00</td>
<td>113</td>
</tr>
<tr>
<td>mat-n_3</td>
<td>0.89</td>
<td>7.53e+00</td>
<td>40</td>
</tr>
<tr>
<td>matrix_9</td>
<td>1.77</td>
<td>5.53e+00</td>
<td>160</td>
</tr>
<tr>
<td>ohne2</td>
<td>0.62</td>
<td>4.34e+01</td>
<td>99</td>
</tr>
<tr>
<td>para-4</td>
<td>0.62</td>
<td>5.70e+00</td>
<td>49</td>
</tr>
<tr>
<td>wang3</td>
<td>2.33</td>
<td>8.90e-01</td>
<td>45</td>
</tr>
<tr>
<td>wang4</td>
<td>1.86</td>
<td>5.10e-01</td>
<td>31</td>
</tr>
<tr>
<td>scircuit</td>
<td>0.90</td>
<td>1.86e+00</td>
<td>Fail</td>
</tr>
<tr>
<td>circuit_4</td>
<td>0.75</td>
<td>1.60e+00</td>
<td>199</td>
</tr>
<tr>
<td>circ2a</td>
<td>0.76</td>
<td>2.19e+02</td>
<td>18</td>
</tr>
</tbody>
</table>

Results for the 10 systems - ARMS-ddPQ + GMRES(60) & GMRES(100)
<table>
<thead>
<tr>
<th></th>
<th>Fill Factor</th>
<th>Set-up Time</th>
<th>GMRES(60) Its.</th>
<th>GMRES(60) Time</th>
<th>GMRES(100) Its.</th>
<th>GMRES(100) Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same param’s</td>
<td>0.89</td>
<td>1.81</td>
<td>400</td>
<td>9.13e+01</td>
<td>297</td>
<td>8.79e+01</td>
</tr>
<tr>
<td>Droptol = .001</td>
<td>1.00</td>
<td>1.89</td>
<td>98</td>
<td>2.23e+01</td>
<td>82</td>
<td>2.27e+01</td>
</tr>
</tbody>
</table>

Solution of the system scircuit – no scaling + two different sets of parameters.
PARALLEL IMPLEMENTATION
Introduction

- Thrust of parallel computing techniques in most applications areas.

- Programming model: Message-passing seems (MPI) dominates

- Open MP and threads for small number of processors

- Important new reality: parallel programming has penetrated the ‘applications’ areas [Sciences and Engineering + industry]

- Problem 1: algorithms lagging behind somewhat

- Problem 2: Message passing is painful for large applications. ‘Time to solution’ up.
“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
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)
Standard 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 \(\Omega_i\).
Discretization of domain
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Coefficient Matrix**
Types of mappings

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

- Can adapt PDE viewpoint to general sparse matrices
- Will use the graph representation and 'vertex-based' viewpoint
DISTRIBUTED SPARSE MATRICES
Simple illustration: Block assignment. Assign equation \( i \) and unknown \( i \) to a given 'process'.

Naive partitioning - won’t work well in practice.
Best idea is to use the adjacency graph of $A$:

Vertices = $\{1, 2, \cdots, n\}$;

Edges: $i \rightarrow j$ iff $a_{ij} \neq 0$

Graph partitioning problem:

- Want a partition of the vertices of the graph so that

  (1) partitions have $\sim$ the same sizes

  (2) interfaces are small in size
General Partitioning of a sparse linear system

\[ S_1 = \{1, 2, 6, 7, 11, 12\} \]: This means equations and unknowns 1, 2, 3, 6, 7, 11, 12 are assigned to Domain 1.

\[ S_2 = \{3, 4, 5, 8, 9, 10, 13\} \]

\[ S_3 = \{16, 17, 18, 21, 22, 23\} \]

\[ S_4 = \{14, 15, 19, 20, 24, 25\} \]
Alternative: Map elements / edges rather than vertices

Equations/unknowns 3, 8, 12 shared by 2 domains. From distributed sparse matrix viewpoint this is an overlap of one layer

Partitioners: Metis, Chaco, Scotch, ..

More recent: Zoltan, H-Metis, PaToH
Standard dual objective: “minimize” communication + “balance” partition sizes

Recent trend: use of hypergraphs [PaToh, Hmetis,...]

Hypergraphs are very general. Ideas borrowed from VLSI work

Main motivation: to better represent communication volumes when partitioning a graph. Standard models face many limitations

Hypergraphs can better express complex graph partitioning problems and provide better solutions. Example: completely nonsymmetric patterns.
Local interface variables always ordered last.

Need: 1) to set up the various “local objects”. 2) Preprocessing to prepare for communications needed during iteration?
Local view of distributed matrix:

The local system:

\[
\begin{pmatrix}
B_i & F_i \\
E_i & C_i
\end{pmatrix}
\begin{pmatrix}
u_i \\
y_i
\end{pmatrix}
+ 
\begin{pmatrix}
0 \\
\sum_{j \in N_i} E_{ij} y_j
\end{pmatrix}
= 
\begin{pmatrix}
f_i \\
g_i
\end{pmatrix}
\]

\( u_i \): Internal variables; \( y_i \): Interface variables
The local matrix consists of 2 parts: a part (\(A_{loc}\)) which acts on local data and another (\(B_{ext}\)) which acts on remote data.

- Once the partitioning is available these parts must be identified and built locally.
- In finite elements, assembly is a local process.
- How to perform a matrix vector product? [needed by iterative schemes?]
Algorithm:

1. Communicate: exchange boundary data.
   
   **Scatter** $x_{\text{bound}}$ to neighbors - **Gather** $x_{\text{ext}}$ from neighbors

2. Local matrix – vector product
   
   \[ y = A_{\text{loc}} x_{\text{loc}} \]

3. External matrix – vector product
   
   \[ y = y + B_{\text{ext}} x_{\text{ext}} \]

**NOTE:** 1 and 2 are independent and can be overlapped.
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.
/*------------------------ call blas1 function

    tloc = DDOT(n, x, incx, y, incy);

/*------------------------ call global reduction

    MPI_Allreduce(&tloc,&ro,1,MPI_DOUBLE,MPI_SUM,comm);
A remark: the global viewpoint

\[
\begin{pmatrix}
B_1 & F_1 \\
B_2 & F_2 \\
\vdots & \vdots \\
B_p & F_p \\
\end{pmatrix}
\begin{pmatrix}
E_1 \\
E_2 \\
\vdots \\
E_p \\
\end{pmatrix}
\begin{pmatrix}
C_1 & E_{12} & \cdots & E_{1p} \\
E_{21} & C_2 & \cdots & E_{2p} \\
\vdots & \vdots & \ddots & \vdots \\
E_{p1} & E_{p2} & \cdots & C_p \\
\end{pmatrix}
\begin{pmatrix}
u_1 \\
u_2 \\
\vdots \\
u_p \\
y_1 \\
y_2 \\
\vdots \\
y_p \\
\end{pmatrix}
= 
\begin{pmatrix}
f_1 \\
f_2 \\
\vdots \\
f_p \\
g_1 \\
g_2 \\
\vdots \\
g_p \\
\end{pmatrix}
\]

← Interior variables → ← Interface variables →
SCHUR COMPLEMENT-BASED PRECONDITIONERS
Schur complement system

Local system can be written as

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

\(x_i\) = vector of local unknowns, \(y_{i,ext}\) = external interface variables, and \(b_i\) = local part of RHS.
Local equations

\[
\begin{pmatrix}
B_i & F_i \\
E_i & C_i
\end{pmatrix}
\begin{pmatrix}
u_i \\
y_i
\end{pmatrix}
+
\begin{pmatrix}
0 \\
\sum_{j \in N_i} E_{ij} y_j
\end{pmatrix}
=
\begin{pmatrix}
f_i \\
g_i
\end{pmatrix}
\]  

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

\[
S_i = C_i - E_i B_i^{-1} F_i.
\]
**Structure of Schur complement system**

Global Schur complement system:

\[ Sy = g' \]

with:

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

- \( E_{ij} \)'s are sparse = same as in the original matrix
- Can solve global Schur complement system iteratively. Back-substitute to recover rest of variables (internal).
- Can use the procedure as a preconditining to global system.
Simplest idea: Schur Complement Iterations

\[
\begin{pmatrix}
  u_i \\ y_i
\end{pmatrix}
\]

- Internal variables
- Interface variables

- 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

\[
\begin{pmatrix}
B & F \\
E & C
\end{pmatrix}
\begin{pmatrix}
u \\
y
\end{pmatrix}
=
\begin{pmatrix}
f \\
g
\end{pmatrix},
\]

\[
B =
\begin{pmatrix}
B_1 & F_1 \\
& B_2 & F_2 \\
& & \ddots & \vdots \\
& & & B_p & F_p \\
E_1 & E_2 & \cdots & E_p & C
\end{pmatrix}
\]

Block LU factorization of \( A \):

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

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

- equivalent to Block-Jacobi preconditioner for Schur complement.

- Solve with, e.g., a few \( s \) (e.g., 5) of GMRES
Question: How to solve with $S_i$?

Can use LU factorization of local matrix $A_i = \begin{pmatrix} B_i & F_i \\ E_i & C_i \end{pmatrix}$

and exploit the relation:

\[
A_i = \begin{pmatrix} L_{B_i} & 0 \\ E_i U_{B_i}^{-1} & L_{S_i} \end{pmatrix} \begin{pmatrix} U_{B_i} & L_{B_i}^{-1} F_i \\ 0 & U_{S_i} \end{pmatrix} \rightarrow L_{S_i} U_{S_i} = S_i
\]

Need only the (l) LU factorization of the $A_i$ [rest is already available]

PARALLEL ARMS
Parallel implementation of ARMS

Three types of points:
- interior (independent sets)
- local interfaces
- 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.

Algorithm: Multicolor Distributed ILU(0)

1. Eliminate local rows,
2. Receive external interf. rows from PEs s.t. $\text{color}(PE) < \text{MyColor}$
3. Process local interface rows
4. Send local interface rows to PEs s.t. $\text{color}(PE) > \text{MyColor}$
Methods implemented in pARMS:

- **add_x**: Additive Schwarz with method $x$ for subdomains. With/out overlap. $x$ = one of ILUT, ILUK, ARMS.

- **sch_x**: Schur complement technique, with method $x$ = factorization used for local submatrix. Same $x$ as above. 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) preconditions to solve global Schur complement system obtained from ARMS reduction.
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 \([100 \times 100]\) - Global linear system with \(10,000 \times nproc\) unknowns.

2. Comparison with a parallel direct solver – symmetric problems

3. Large irregular matrix example arising from magneto hydrodynamics.
100 x 100 mesh per processor – Wall–Clock Time

Times for 2D PDE problem with fixed subproblem size
100 x 100 mesh per processor – Iterations

Iterations for 2D PDE problem with fixed subproblem size
Times for 2D PDE problem with fixed subproblem size
100 x 100 mesh per processor – Iterations

Processors

Iterations

- add_arms no its
- add_arms ovp no its
- sch_arms
- sch_gilu0
- sch_gilu0 no its
- sch_sgs
- sch_sgs no its

Iterations

Processors
Direct solvers:

- **SUPERLU**
  

- **MUMPS: [cerfacs]**

- **Univ. Minn. / IBM’s PSPASES [SPD matrices]**
  

- **UMFPACK**
Iterative solvers:

- **PETSc**
  
  http://acts.nersc.gov/petsc/

  and Trilinos (more recent)
  
  http://trilinos.sandia.gov/

  ... are very comprehensive packages..

- **PETSc includes few preconditioners...**

- **Hypre, ML, ...**, all include interfaces to PETSc or trilinos

- **pARMS:**
  
  http://www.cs.umn.edu/~saad/software

  is a more modest effort -