PARALLEL PRECONDITIONERS
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]
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
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.
Multicoloring

- General technique that can be exploited in many different ways to introduce parallelism – generally of order $N$.
- Constitutes one of the most successful techniques (when applicable) on (vector) supercomputers.

**Simple example:** Red-Black ordering.
Observe: L-U solves (or SOR sweeps) will require only diagonal scalings + matrix-vector products with matrices of size $\frac{N}{2}$.
Red-Black ordering leads to equations of the form:

\[
\begin{pmatrix}
D_1 & E \\
F & D_2
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2
\end{pmatrix} =
\begin{pmatrix}
b_1 \\
b_2
\end{pmatrix}
\]

\(D_1\) and \(D_2\) are diagonal.

**Question:** How to solve such a system?

**Method 1:** use ILU(0) on this block system. \(O(N)\) Parallelism.

- Often, the number of iterations is higher than with the natural ordering. But still competitive for easy problems.

- Could use a more accurate preconditioner [e.g., ILUT]. However: \(O(N)\) parallelism lost.
Method 2: SOR/SSOR(k) preconditioner.

- No such difficulty with SOR/SSOR...

- A 'more accurate preconditioner’ means more SOR/SSOR steps. [SOR(k) → k steps of SOR]. Can perform quite well.

Iteration times versus $k$ for SOR($k$) preconditioned GMRES.
Method 3: Eliminate the red unknowns from the system

Reduced system: \[(D_2 - FD_1^{-1}E)x_2 = b_2 - FD_1^{-1}b_1.\]

- Again a sparse linear system with half as many unknowns.
- Can often be efficiently solved with only diagonal preconditioning.

**Question:** Can we recursively color the reduced system and get a “second-level” reduced system?

**Answer:** Yes but need to generalize red-black ordering.
How to generalize Red-Black ordering?

Answer: Multicoloring & independent sets

A greedy multicoloring technique:

- Initially assign color number zero (uncolored) to every node.
- Choose an order in which to traverse the nodes.
- Scan all nodes in the chosen order and at every node $i$ do
  
  $$\text{Color}(i) = \min\{k \neq 0 | k \neq \text{Color}(j), \forall j \in \text{Adj}(i)\}$$

$\text{Adj}(i) =$ set of nearest neighbors of $i = \{k | a_{ik} \neq 0\}$. 
**Independent Sets**

An independent set (IS) is a set of nodes that are not coupled by an equation. The set is maximal if all other nodes in the graph are coupled to a node of IS. If the unknowns of the IS are labeled first, then the matrix will have the form:

\[
\begin{pmatrix}
B & F \\
E & C
\end{pmatrix}
\]

in which $B$ is a diagonal matrix, and $E$, $F$, and $C$ are sparse.

**Greedy algorithm:** Scan all nodes in a certain order and at every node $i$ do: if $i$ is not colored color it Red and color all its neighbors Black. **Independent set:** set of red nodes. **Complexity:** $O(|E| + |V|)$. 
Domain Decomposition

Problem:

\[ \begin{align*}
\Delta u &= f \text{ in } \Omega \\
u &= u_\Gamma \text{ on } \Gamma = \partial \Omega.
\end{align*} \]

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
Coefficient Matrix
Types of mappings

(a) Vertex-based, (b) edge-based, and (c) element-based partitioning
DISTRIBUTED SPARSE MATRICES
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. \( = \{ \text{set of } p \text{ subsets of variables} \} \).

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, ...
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 $x_{\text{bound}}$ 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 part of the code:

call MSG_bdx_send(nloc,x,y,nproc,proc,ix,ipr,ptrn,ierr)

do local matrix-vector product for local points

call amux(nloc,x,y,aloc,jaloc,ialoc)

receive the boundary information

call MSG_bdx_receive(nloc,x,y,nproc,proc,ix,ipr,ptrn,ierr)

do local matrix-vector product for external points

nrow = nloc - nbnd + 1

call amux1(nrow,x,y(nbnd),aloc,jaloc,ialoc(nloc+1))

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 blas1 function */
    tloc = DDOT(n, x, incx, y, incy);
/*------------------------ call global reduction */
    MPI_Allreduce(&tloc, &ro, 1, MPI_DOUBLE, MPI_SUM, comm);
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:

![Diagram showing the local view of a distributed matrix with local data and external data]

Block Jacobi Iteration (Additive Schwarz):

1. Obtain external data $y_i$
2. Compute (update) local residual $r_i = (b - Ax)_i = b_i - A_ix_i - B_iy_i$
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 $col = 1, \ldots, numcols$
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.
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

Overlapped Schwarz with ILU solver. Matrix VENKAT01

Time in seconds versus Number of processors for different machines:
- IBM-COW (FC)
- SGI-COW (HiPPI)
- SP2
- T3E
Multicolor SOR preconditioning. Matrix VENKAT01.

- - o - - multicolor segregated SOR, ILU solver and GMRES
- - x - - multicolor SOR with ILU solver
------------------------ multicolor segregated SOR, ILU solver
---------------------- multicolor SOR

Number of processors

Time in seconds

February 7, 2005
SCHUR COMPLEMENT-BASED PRECONDITIONERS
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**

**Schur complement system**

\[ S y = g' \]

with

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

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

with

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

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 \( M_S = \text{some approximation to } S \).

- 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 \( S_i \)?
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:

$$\begin{bmatrix}
L_{B_i} & 0 \\
E_iU^{-1}_{B_i} & L_{S_i}
\end{bmatrix} \begin{bmatrix}
U_{B_i} & L_{B_i}^{-1}F_i \\
0 & U_{S_i}
\end{bmatrix} \Rightarrow L_{S_i}U_{S_i} = S_i$$
<table>
<thead>
<tr>
<th>Name</th>
<th>Precon</th>
<th>lfil</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>24</th>
<th>36</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>raeffsky1</td>
<td>SAPINV</td>
<td>10</td>
<td>14</td>
<td>13</td>
<td>10</td>
<td>11</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
<td>12</td>
<td>11</td>
<td>9</td>
<td>9</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>SAPINVS</td>
<td>10</td>
<td>16</td>
<td>13</td>
<td>10</td>
<td>11</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
<td>13</td>
<td>11</td>
<td>9</td>
<td>9</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>SLU</td>
<td>10</td>
<td>215</td>
<td>197</td>
<td>198</td>
<td>194</td>
<td>166</td>
<td>171</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
<td>48</td>
<td>50</td>
<td>40</td>
<td>42</td>
<td>41</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>BJ</td>
<td>10</td>
<td>85</td>
<td>171</td>
<td>173</td>
<td>273</td>
<td>252</td>
<td>263</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
<td>82</td>
<td>170</td>
<td>173</td>
<td>271</td>
<td>259</td>
<td>259</td>
</tr>
</tbody>
</table>

Number of FGMRES(20) iterations for the RAEFSKY1 problem.
<table>
<thead>
<tr>
<th>Name</th>
<th>Precon</th>
<th>$l_{fil}$</th>
<th>16</th>
<th>24</th>
<th>32</th>
<th>40</th>
<th>56</th>
<th>64</th>
<th>80</th>
<th>96</th>
</tr>
</thead>
<tbody>
<tr>
<td>af23560</td>
<td>SAPINV</td>
<td>20</td>
<td>32</td>
<td>36</td>
<td>27</td>
<td>29</td>
<td>73</td>
<td>35</td>
<td>71</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30</td>
<td>32</td>
<td>35</td>
<td>23</td>
<td>29</td>
<td>46</td>
<td>60</td>
<td>33</td>
<td>52</td>
</tr>
<tr>
<td>SAPINVS</td>
<td></td>
<td>20</td>
<td>32</td>
<td>35</td>
<td>24</td>
<td>29</td>
<td>55</td>
<td>35</td>
<td>37</td>
<td>59</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30</td>
<td>32</td>
<td>34</td>
<td>23</td>
<td>28</td>
<td>43</td>
<td>45</td>
<td>23</td>
<td>35</td>
</tr>
<tr>
<td>SLU</td>
<td></td>
<td>20</td>
<td>81</td>
<td>105</td>
<td>94</td>
<td>88</td>
<td>90</td>
<td>76</td>
<td>85</td>
<td>71</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30</td>
<td>38</td>
<td>34</td>
<td>37</td>
<td>39</td>
<td>38</td>
<td>39</td>
<td>38</td>
<td>35</td>
</tr>
<tr>
<td>BJ</td>
<td></td>
<td>20</td>
<td>37</td>
<td>153</td>
<td>53</td>
<td>60</td>
<td>77</td>
<td>80</td>
<td>95</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30</td>
<td>36</td>
<td>41</td>
<td>53</td>
<td>57</td>
<td>81</td>
<td>87</td>
<td>97</td>
<td>115</td>
</tr>
</tbody>
</table>

Number of FGMRES(20) iterations for the AF23560 problem.
Times and iteration counts for solving a $360 \times 360$ discretized Laplacean problem with 3 different preconditioners using flexible 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).
Three approaches to graph partitioning:

1. Spectral methods (Recursive Spectral Bisection)

2. Geometric techniques [Houstis & Rice et al., Miller, Vavasis, Teng et al.] Coordinates are required.

3. Graph Theory techniques [use graph, but no coordinates]
   - Currently best known technique is Metis (multi-level algorithm)
   - Simplest idea: Recursive Graph Bisection; Nested dissection (George & Liu, 1980; Liu 1992)
   - Advantages: simplicity – no coordinates required
The Level Set Expansion Algorithm

- Given: \( p \) nodes ‘uniformly’ spread in the graph (roughly same distance from one another).

Do a level-set traversal from each node simultaneously.

Best described for an example on a 15 \( \times \) 15 Finite Difference grid.

- See [Goehring-Saad ’94, See Cai-Saad ’95]
Alternative criteria

Criterion commonly used: Partition so that

1. each subdomain has about the same number of vertices and
2. The total number of edge cuts is ‘minimized’

Question: Is this the best that can be done?

- Hard to predict the effect of partitioning on the number of iterations.
Example: A test problem with discontinuous coefficient.

\[ a(x, y) = b(x, y) = 1 \]

\[ a(x,y)=10^2 \]
\[ b(x,y)=1 \]
A macro-partitioning of the problem with discontinuities
Partitioning techniques compared:

1. Double-Stripe algorithm ((a) do a BFSf traversal - (b) get subdomains along the way (c) repeat for each subdomain)

2. Same as (1) but do in two phases: first assign low (shaded) region to $3p/4$ processors. Then the rest to $p/4$ processors.

3. Straightforward regular partitioning (“partition by hand”)

4. Same as in (3) but separate again low and high regions.
<table>
<thead>
<tr>
<th>PEs</th>
<th>Time</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>24</td>
</tr>
<tr>
<td>DoubleStripe</td>
<td>9.66</td>
<td>14.34</td>
</tr>
<tr>
<td></td>
<td>81</td>
<td>132</td>
</tr>
<tr>
<td>DoubleStripe_m</td>
<td>5.58</td>
<td>6.33</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>61</td>
</tr>
<tr>
<td>ByHand</td>
<td>6.39</td>
<td>12.73</td>
</tr>
<tr>
<td></td>
<td>52</td>
<td>120</td>
</tr>
<tr>
<td>ByHand_m</td>
<td>6.3</td>
<td>7.34</td>
</tr>
<tr>
<td></td>
<td>53</td>
<td>62</td>
</tr>
</tbody>
</table>

Behavior of the Schur-LU preconditioner with four different partitioning strategies.
**Example 2: Using a shortest path strategy**

**Key idea:** Take into account matrix values when partitioning.

- **Heuristics must be used because there is no way of predicting behavior of preconditioners for different partitionings.**

A first observation: for load-balancing (Matvecs) – a better criterion is to try to equalize “volume” (i.e., number of edges in a subdomain) instead of “number of nodes”.

- **We can go one step further by adding weights to the edges.**
Weights used:

\[ w_{ij} = \frac{1}{|a_{ij}| + |a_{ji}|} \]

Criterion used: “group together matrix entries whose added weights are small ..”

- Employ a shortest path algorithm to find paths with smallest length
- Do level-set expansion accordingly
- The cost is high: \( \mathcal{O}((|V| + |E|) \log |V|) \) with a standard implementation.
Result: impact on Schur-LU preconditioner

Test with RAEFSKY3 - a Harwell-Boeing matrix (extended set).

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard</td>
<td>175.27</td>
<td>161</td>
</tr>
<tr>
<td>Shortest-Path</td>
<td>122.95</td>
<td>106</td>
</tr>
</tbody>
</table>

- Results are fairly consistent on other tests.
- Only drawback: cost of shortest path alg.