DOMAIN DECOMPOSITION-TYPE METHODS

- Back to scientific computing. Introduction – motivation
- Domain partitioning and distributed sparse matrices
- Basic algorithms: distributed Matvec
- Distributed preconditoners: additive Schwarz, multiplicatieve Schwarz.
- Schur complement techniques

Introduction

- Back to scientific computing. So solve: PDE or $Ax = b$
- Thrust of parallel computing techniques in most applications areas.
- Programming model: Message-passing seems (MPI) dominates
- Open MP for small number of processors
- Also: GPUs (CUDA, ...) in most High-performance computers
- Parallel programming has penetrated most ‘applications’ areas [Sciences and Engineering, Data science, industry, ...]

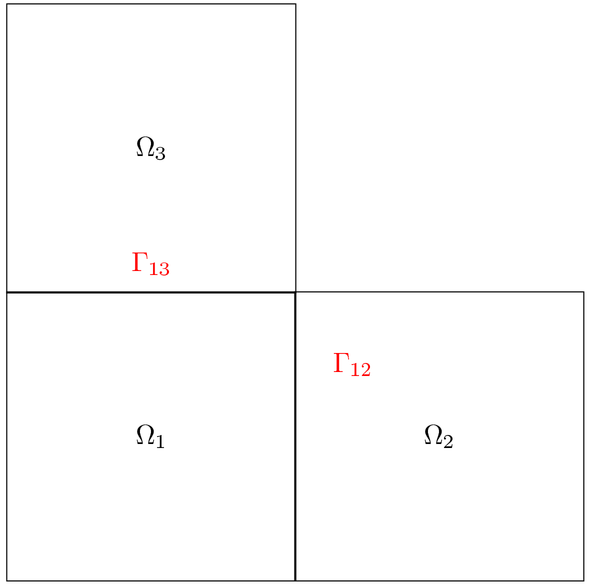
Domain Decomposition: A Model problem

Problem:

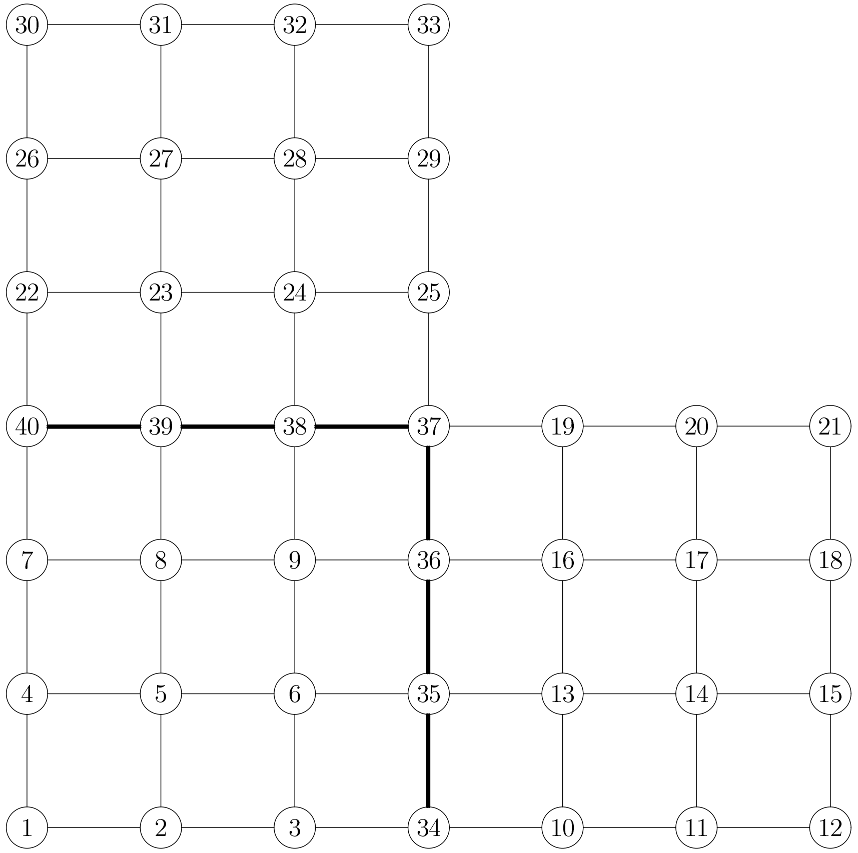
$$\begin{cases} \Delta u = f \text{ in } \Omega \\ u = u_\Gamma \text{ on } \Gamma = \partial\Omega. \end{cases}$$

Domain:

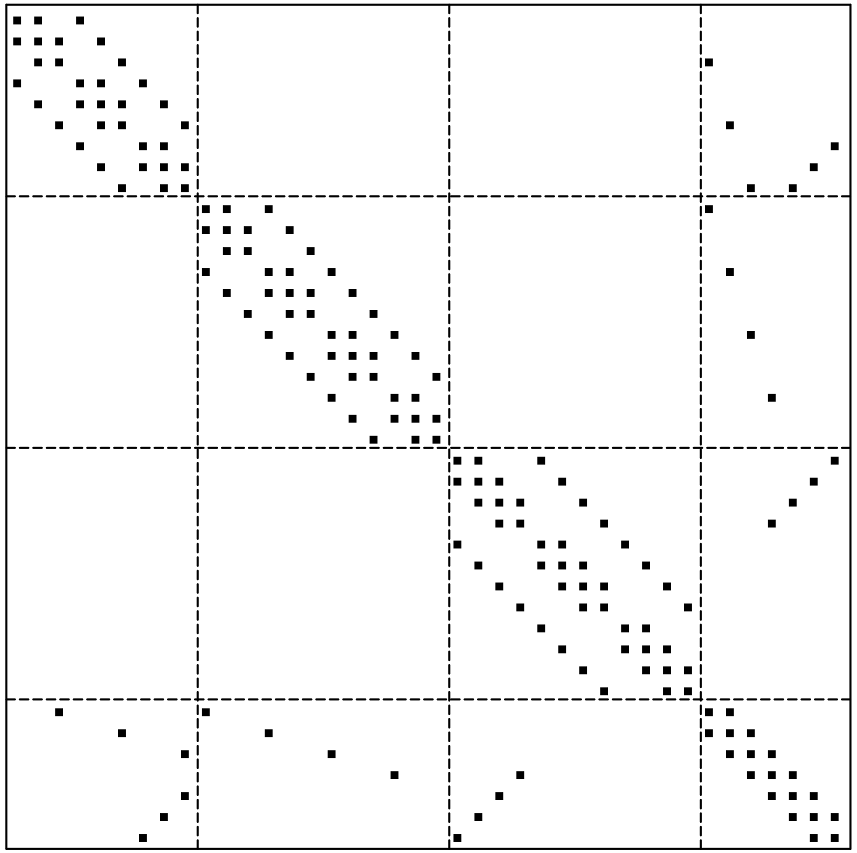
$$\Omega = \bigcup_{i=1}^{s} \Omega_i,$$



➤ 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.

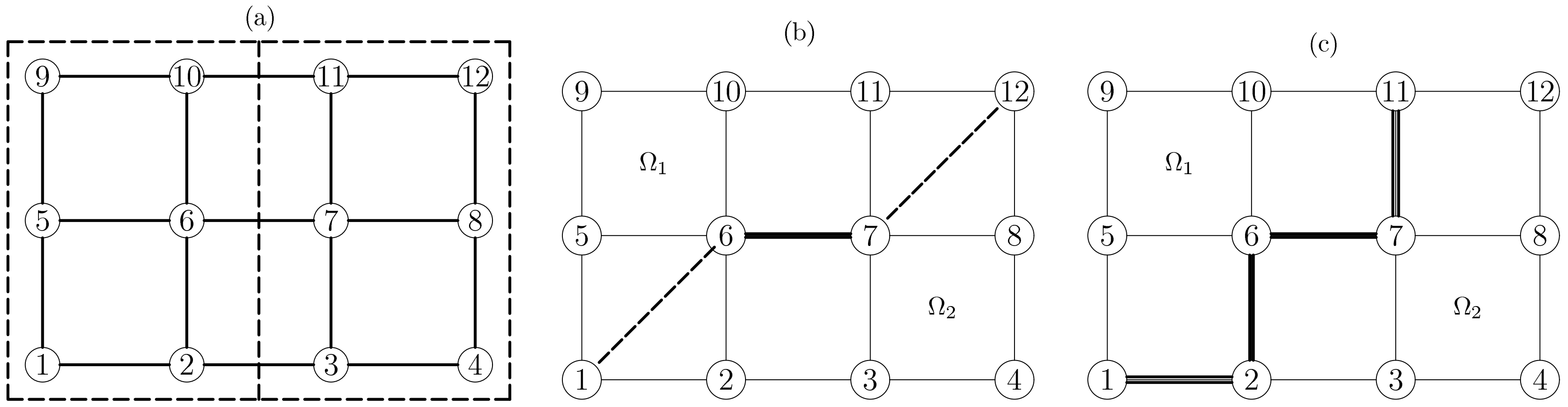


Discretization of domain



Coefficient Matrix

Types of mappings

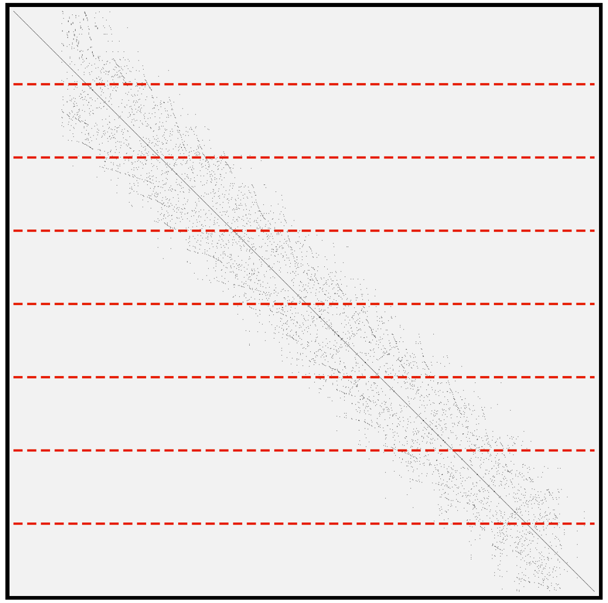


(a) Vertex-based; (b) edge-based; and (c) element-based partitioning

- Can adapt PDE viewpoint to general sparse matrices
- Will use the graph representation and 'vertex-based' viewpoint –

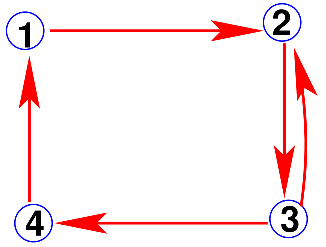
Generalization: Distributed Sparse Systems

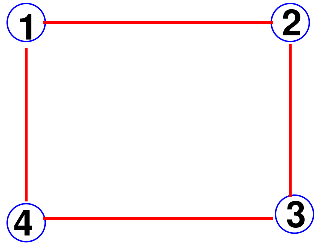
- ➤ Simple illustration: Block assignment. Assign equation i and unknown i to a given 'process'
- ➤ Naive partitioning - won't work well in practice



➤ Best idea is to use the adjacency graph of A:

Vertices = $\{1, 2, \cdots, n\}$;
Edges: $i \to j$ iff $a_{ij} \neq 0$





Graph partitioning problem:

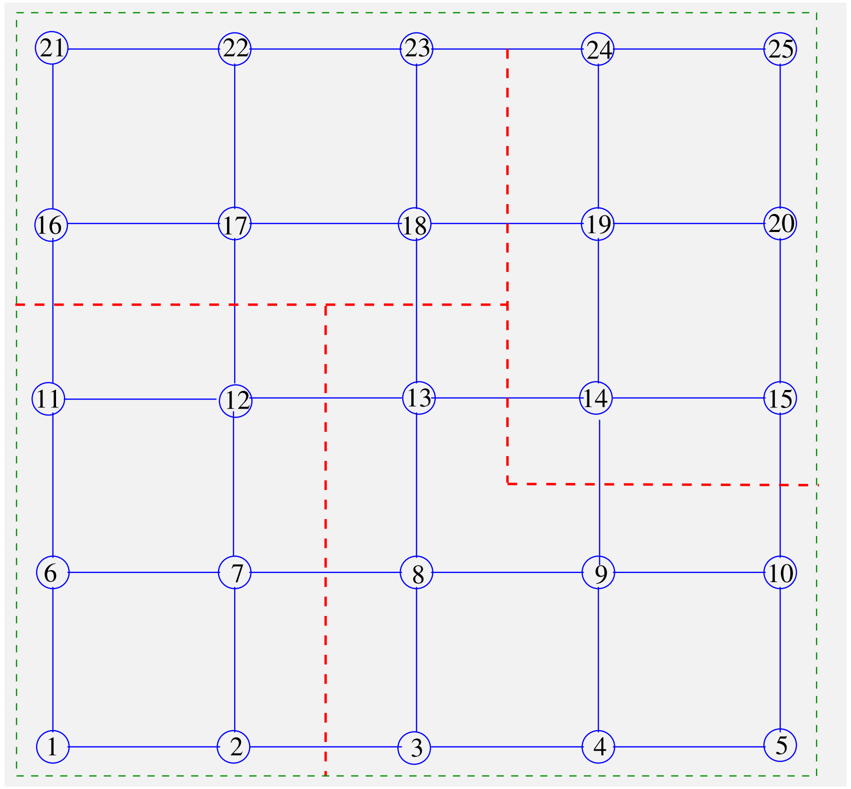
- Want a partition of the vertices of the graph so that

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

(2) interfaces are small in size

➤ Standard dual objective: "minimize" communication + "balance" partition sizes

General Partitioning of a sparse linear system



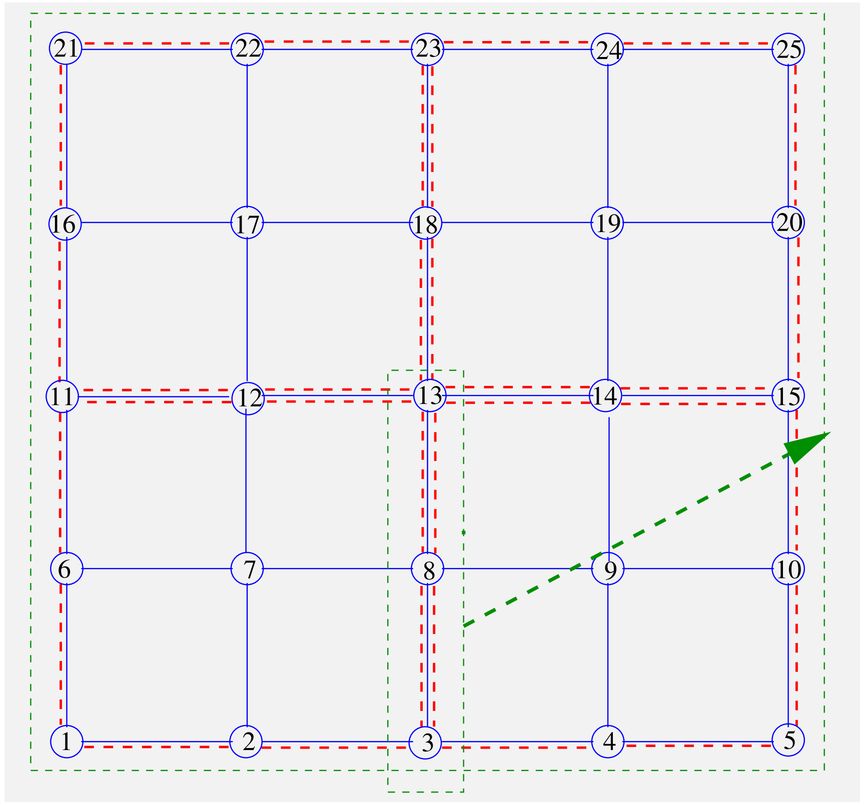
$S_1 = \{1, 2, 6, 7, 11, 12\}$: This means equations and unknowns 1, 2, 3, 6, 7, 11, 12 are assigned to Domain 1.

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

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

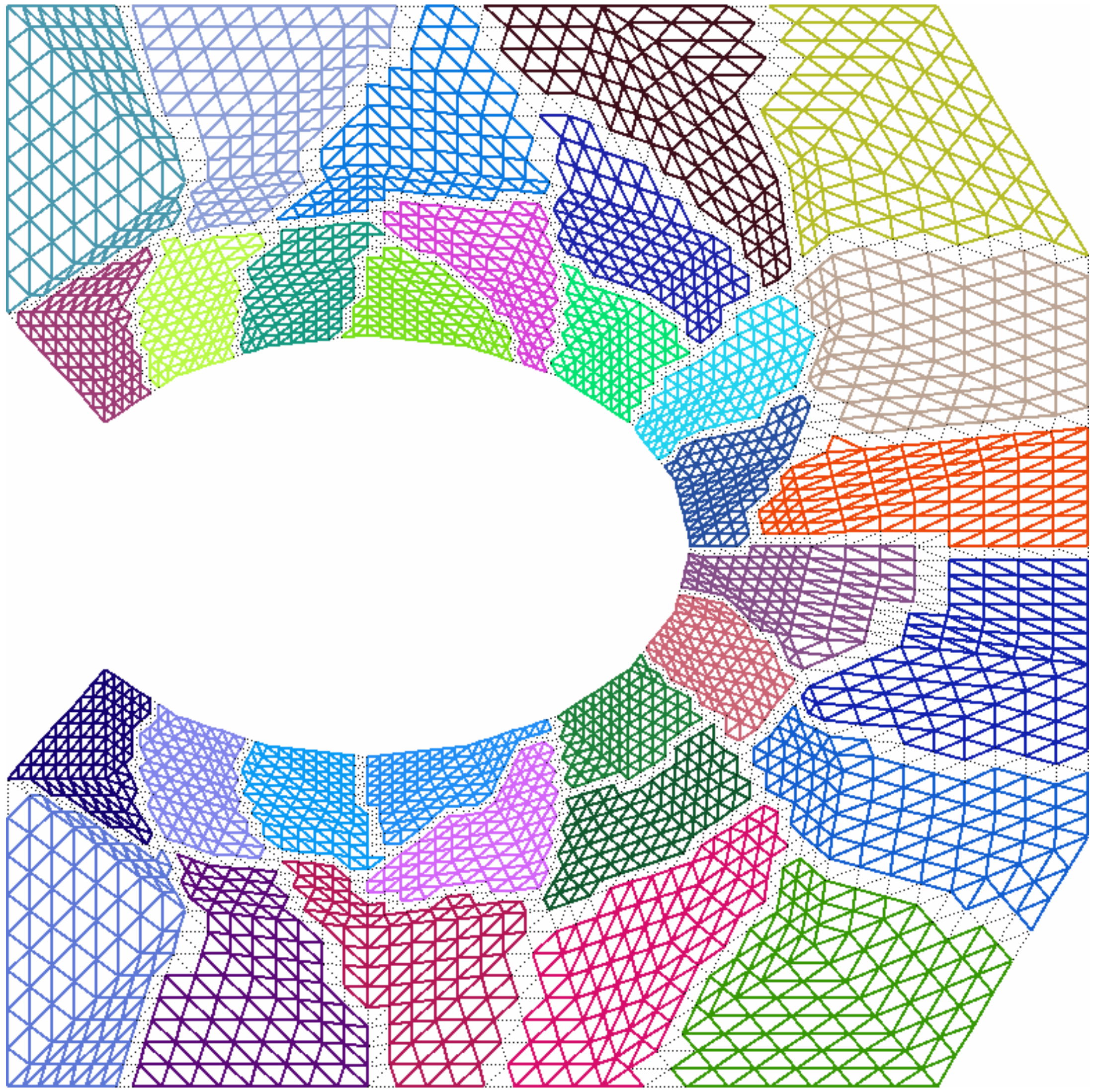
$S_4 = \{14, 15, 19, 20, 24, 25\}$

Alternative: Map elements / edges rather than vertices



Equations/unknowns 3, 8, 13 shared by 2 domains. From distributed sparse matrix viewpoint this is an overlap of one layer

➤ Partitioners : Metis, Chaco, Scotch, Zoltan, H-Metis, PaToH, ..



A few words about hypergraphs

➤ 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

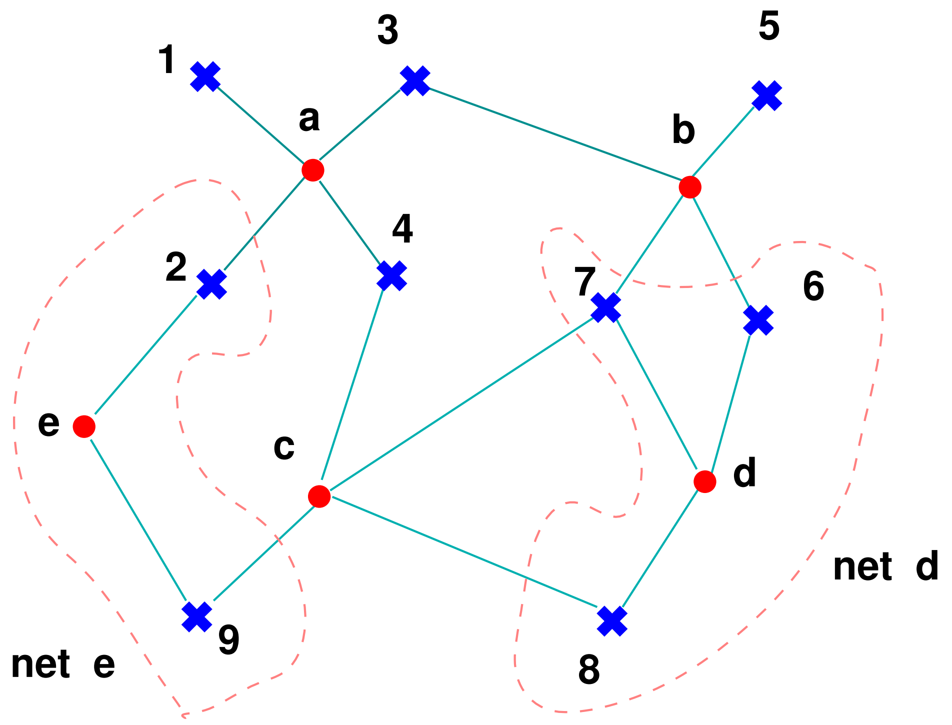
➤ Hypergraphs can better express complex graph partitioning problems and provide better solutions.

➤ Example: completely nonsymmetric patterns ...

➤ .. Even rectangular matrices

Example: $V = \{1, \ldots, 9\}$ and $E = \{a, \ldots, e\}$ with

$a = \{1,2,3,4\}$, $b = \{3,5,6,7\}$, $c = \{4,7,8,9\}$,
$d = \{6,7,8\}$, and $e = \{2,9\}$



Boolean matrix:

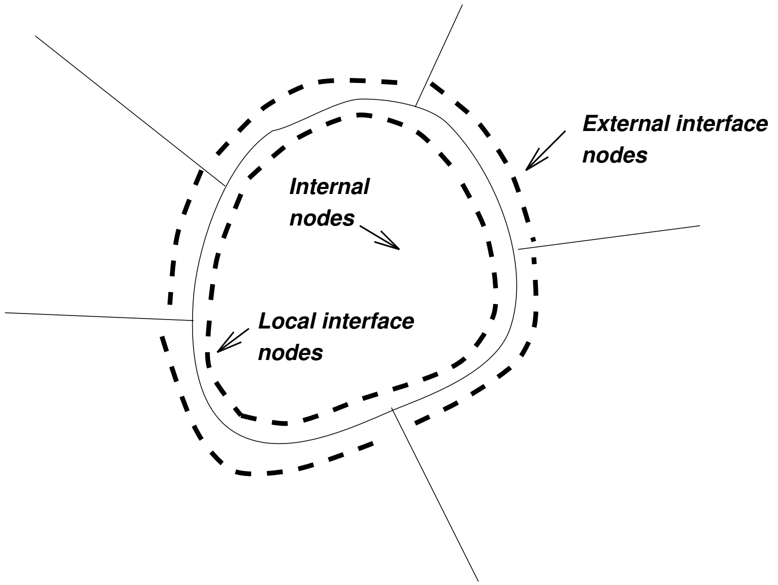
$$A = \begin{array}{ccccccccc|c} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & \\ \hline 1 & 1 & 1 & 1 & & & & & & a \\ & & 1 & & 1 & 1 & 1 & & & b \\ & & & 1 & & & 1 & 1 & 1 & c \\ & & & & & 1 & 1 & 1 & & d \\ & 1 & & & & & & & 1 & e \end{array}$$

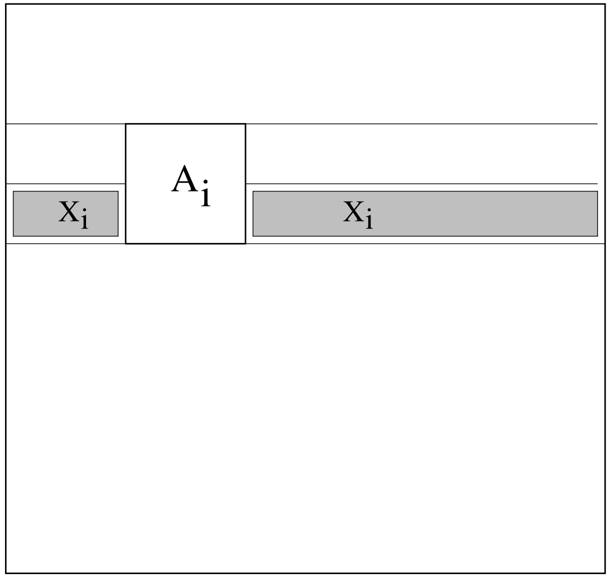
Distributed Sparse matrices (continued)

➤ 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

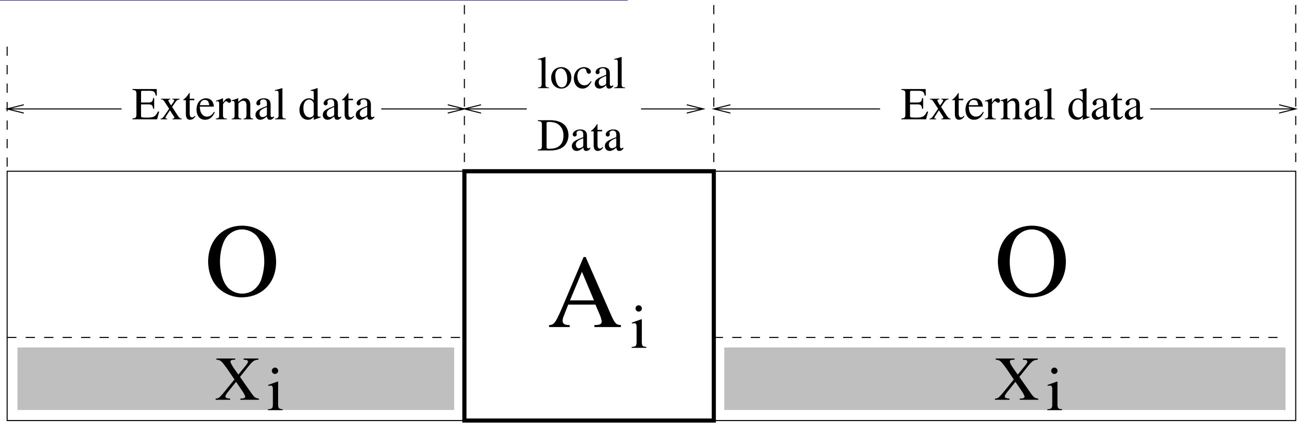




➤ Local interface variables always ordered last.

➤ Need: 1) to set up the various “local objects”. 2) Preprocessing to prepare for communications needed during iteration?

Local view of distributed matrix:

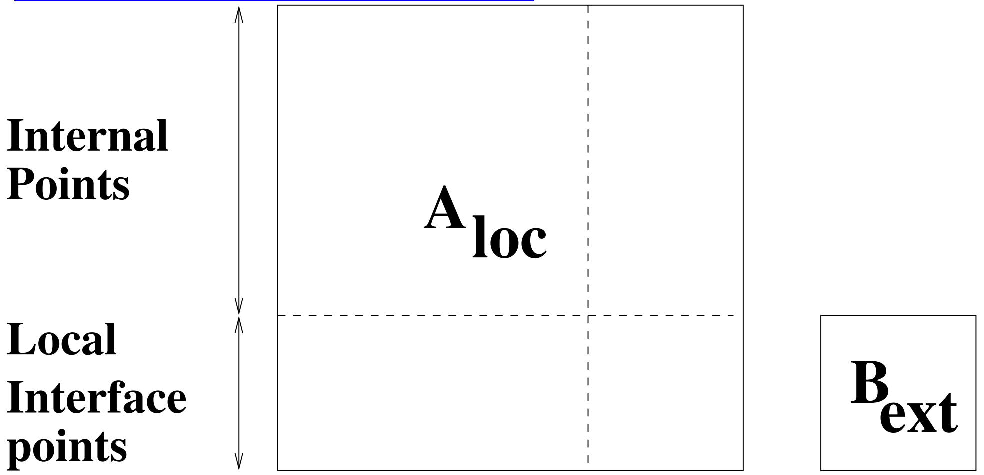


The local system:

$$\underbrace{\begin{pmatrix} B_i & F_i \\ E_i & C_i \end{pmatrix}}_{A_i} \begin{pmatrix} u_i \\ y_i \end{pmatrix} + \underbrace{\begin{pmatrix} 0 \\ \sum_{j \in N_i} E_{ij} y_j \end{pmatrix}}_{y_{ext}} = \begin{pmatrix} f_i \\ g_i \end{pmatrix}$$

➤ u_i : Internal variables; y_i : Interface variables

The local matrix:



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?]

Distributed Sparse Matrix-Vector Product Kernel

Algorithm:

1. Communicate: exchange boundary data.

$$\boxed{\text{Scatter } x_{bound} \text{ to neighbors - Gather } x_{ext} \text{ from neighbors}}$$

2. Local matrix – vector product

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

3. External matrix – vector product

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

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

Distributed Flexible GMRES (FGMRES)

1. **Start:** Choose x_0 and m. Let of the Krylov subspaces. Define $\bar{H}_m \in \mathbb{R}^{(m+1)\times m}$ with $\bar{H}_m \equiv 0$. and initialize all its entries $h_{i,j}$ to zero.

2. **Arnoldi process:**

   (a) Compute $r_0 = b - Ax_0$, $\beta = \|r_0\|_2$ and $v_1 = r_0/\beta$.

   (b) For $j = 1, ..., m$ do

   - Compute $\boxed{z_j := M_j^{-1} v_j}$;Compute $\boxed{w := Az_j}$;
   - For $i = 1, \ldots, j,$ do 1. $h_{i,j} := (w, v_i)$ 2. $w := w - h_{i,j} v_i$ $$\begin{cases} h_{i,j} := (w, v_i) \\ w := w - h_{i,j} v_i \end{cases}$$
   - Compute $h_{j+1,j} = \|w\|_2$ and $v_{j+1} = w/h_{j+1,j}$.

   (c) Define $Z_m := [z_1, ..., z_m]$

3. **Form the approximate solution:** Compute

$y_m = \mathrm{argmin}_y \|\beta e_1 - \bar{H}_m y\|_2$ and $x_m = x_0 + [z_1, z_2, ..., z_m] y_m$ and $e_1 = [1, 0, \ldots, 0]^T$. with $\bar{H}_m = \{h_{i,j}\}_{1 \le i \le j+1; 1 \le j \le m}$.

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

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);
```

A remark: the global viewpoint

$$\left(\begin{array}{ccccc|cccc}
B_1 & & & & & F_1 & & & \\
 & B_2 & & & & & F_2 & & \\
 & & \ddots & & & & & \ddots & \\
 & & & \ddots & & & & & \\
 & & & & B_p & & & & F_p \\
\hline
E_1 & & & & & C_1 & E_{12} & \cdots & E_{1p} \\
 & E_2 & & & & E_{21} & C_2 & \cdots & E_{2p} \\
 & & \ddots & & & \vdots & \vdots & \vdots & \\
 & & & & E_p & E_{p1} & E_{p2} & \cdots & C_p
\end{array}\right)
\begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ \vdots \\ u_p \\ y_1 \\ y_2 \\ \vdots \\ y_p \end{pmatrix}
=
\begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ \vdots \\ f_p \\ g_1 \\ g_2 \\ \vdots \\ g_p \end{pmatrix}$$

$\leftarrow$ $\boxed{\text{Interior variables}}$ $\rightarrow$ $\leftarrow$ $\boxed{\text{Interface variables}}$ $\rightarrow$

Example: Distributed ILU(0)

- Global view of matrix is (for 4 processors):
- A_i = local matrix restricted to internal nodes only

$$A = \left(\begin{array}{cccc|c} A_1 & & & & F_1 \\ & A_2 & & & F_2 \\ & & A_3 & & F_3 \\ & & & A_4 & F_4 \\ \hline E_1 & E_2 & E_3 & E_4 & D \end{array}\right)$$

- 1-st approach: Idea: ILU on this matrix – parallelism available for diagonal blocks. Define an order in which to eliminate interface unknowns.
- 2-nd approach: Multi-color, k-step SOR or SSOR preconditioners.
- 3-rd approach: Solve equations for all interface points [Schur Complement approach] – to precondition, use ideas from DD.

Example: Distributed ILU(0) – cont.

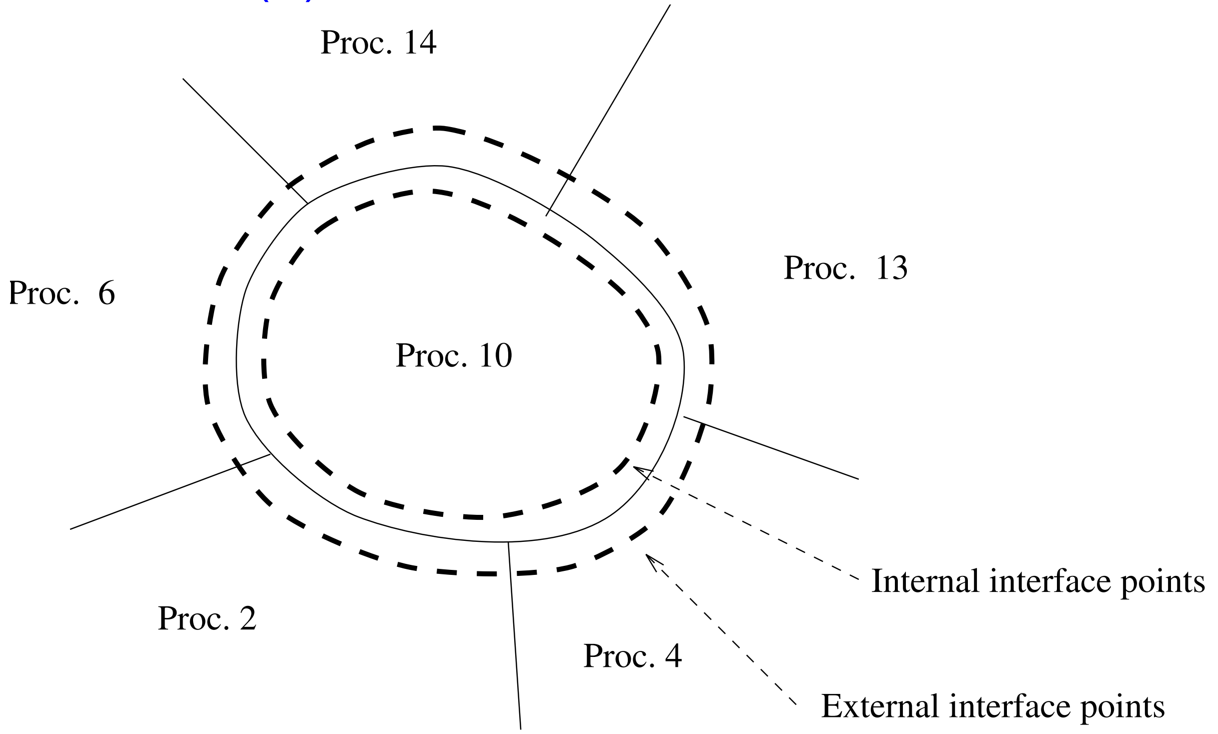
➤ Easy to understand from a local view of distributed matrix

➤ Start by selecting an order [or a “schedule”, or a “priority rule”] in which to process globally

➤ Then locally:

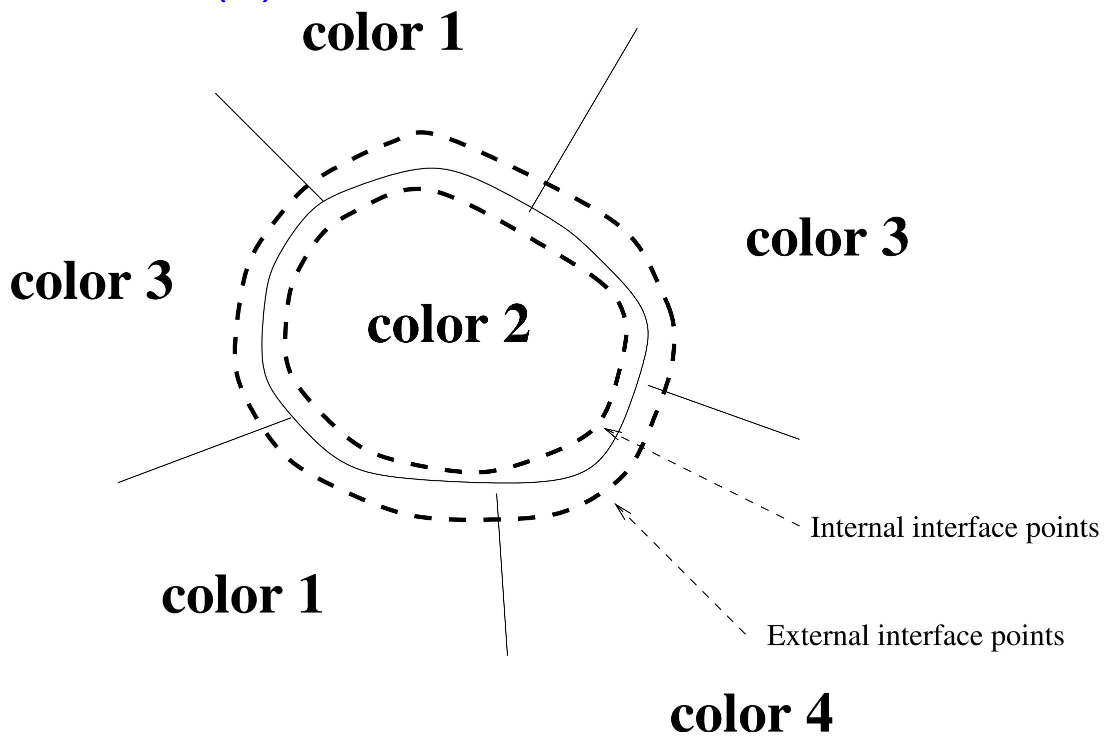
1. Eliminate internal rows
2. Receive rows needed to process local interface rows
3. Process local interface rows
4. Send local interface rows to processors needing them

A distributed view of ILU(0) – schedule based on PE numbers



Note: any schedule can be used provided neighbors have different labels. Example: can use coloring.

A distributed view of ILU(0) – schedule based on PE coloring



- Generalized ILU(k): D. Hysom and A. Pothen ’00.

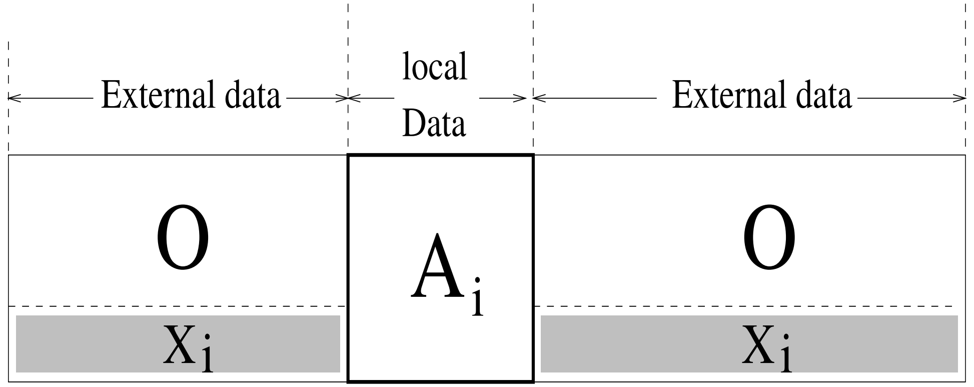
Domain Decomposition–Type preconditoners

- Schwarz Preconditioners
- Schur-complement based Preconditioners
- Multi-level ILU-type Preconditioners

➤ Observation: Often, in practical applications, Schwarz Preconditioners are used : SUB-OPTIMAL

Domain-Decomposition Preconditioners (cont.)

Local view of distributed matrix:



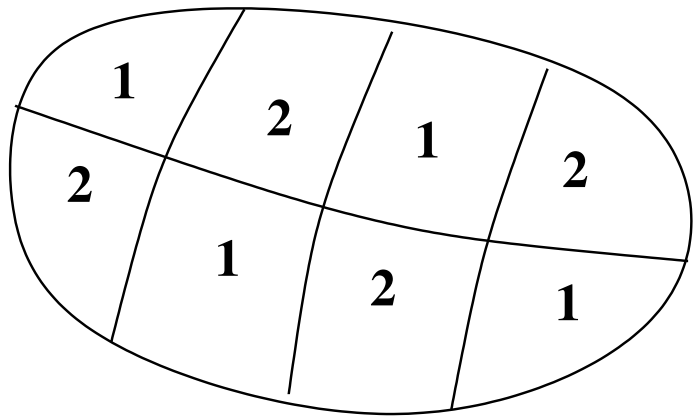
Block Jacobi Iteration (Additive Schwarz):

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 so that:
- No two adjacent subdomains share same color



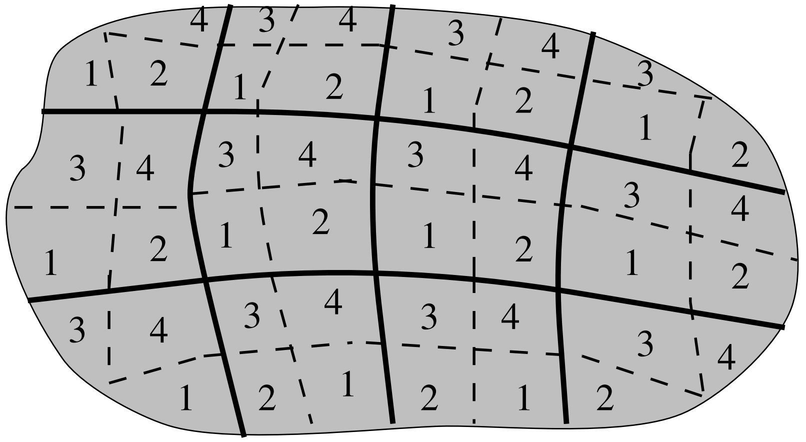
Multicolor Block SOR Iteration (Multiplicative Schwarz):

```
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
```

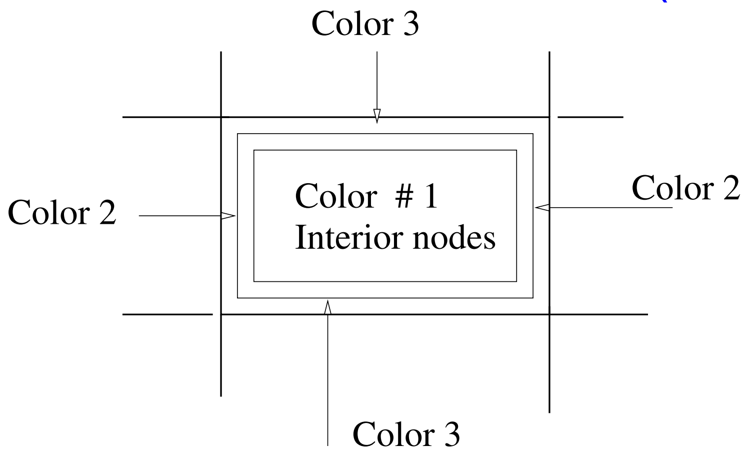
Breaking the sequential color loop

➤ “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 iddle 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) precondioner

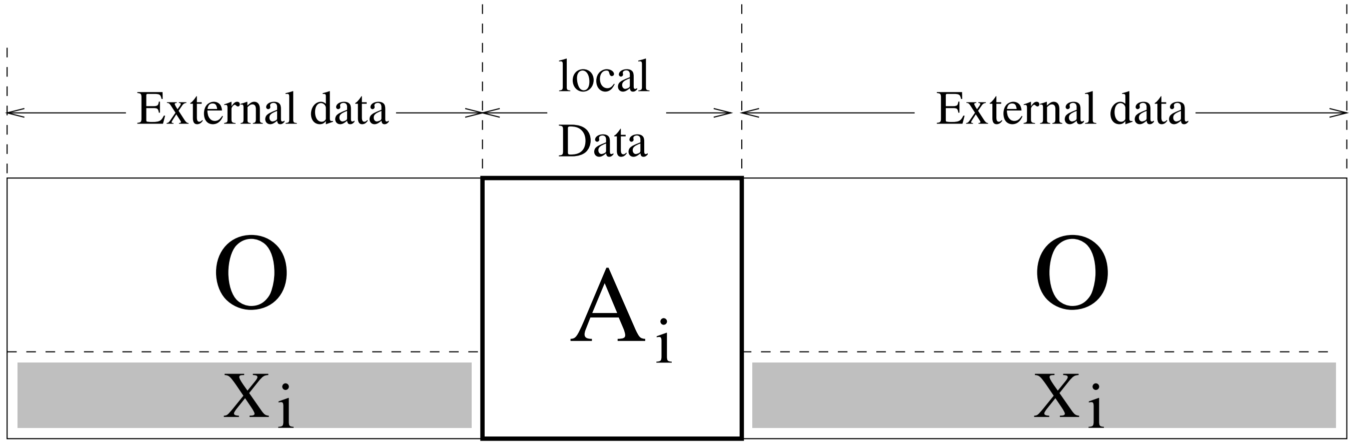
➤ We only use (2) with a small number of inner steps (up to 10) or (3).

SCHUR COMPLEMENT-BASED PRECONDITIONERS

Schur complement system

Local system can be written as

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



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

- Local equations

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

- 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

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

Structure of Schur complement system

Global Schur complement system: $Sy = g'$ with :

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

➤ 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

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

- 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

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

Block LU factorization of A:

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

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:

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

- Need only the (I) LU factorization of the A_i [rest is already available]

- Very easy implementation of (parallel) Schur complement techniques for vertex-based partitioned systems : YS-Sosonkina '97; YS-Sosonkina-Zhang '99.