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 unaravel 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:** $s(A) \approx A^{-1}$

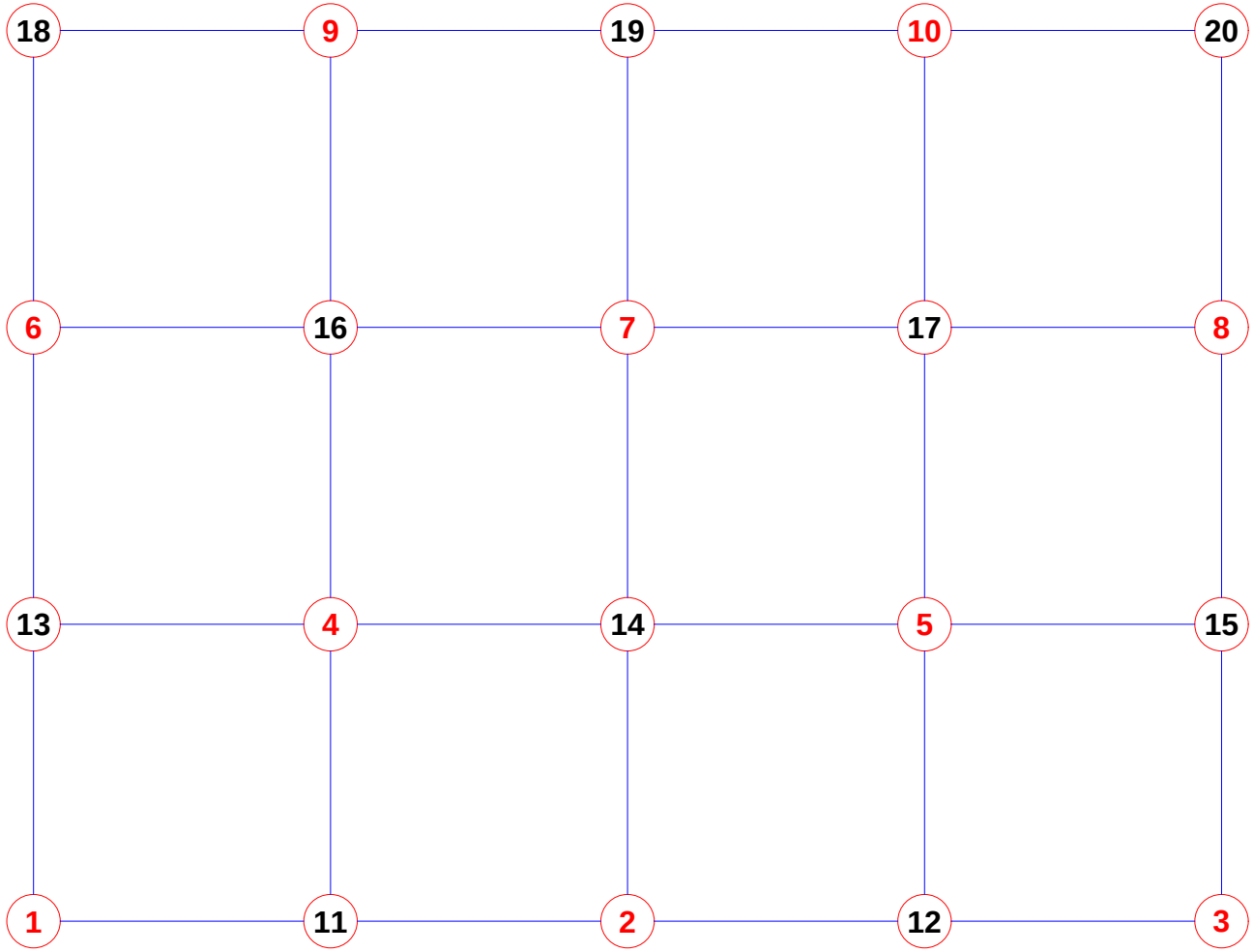
- **Several approaches.**
  - *** Chebyshev polynomials**
  - *** Least squares polynomials**
  - *** Others**
- **Polynomial preconditioners are seldom used in practice.**

Multicoloring

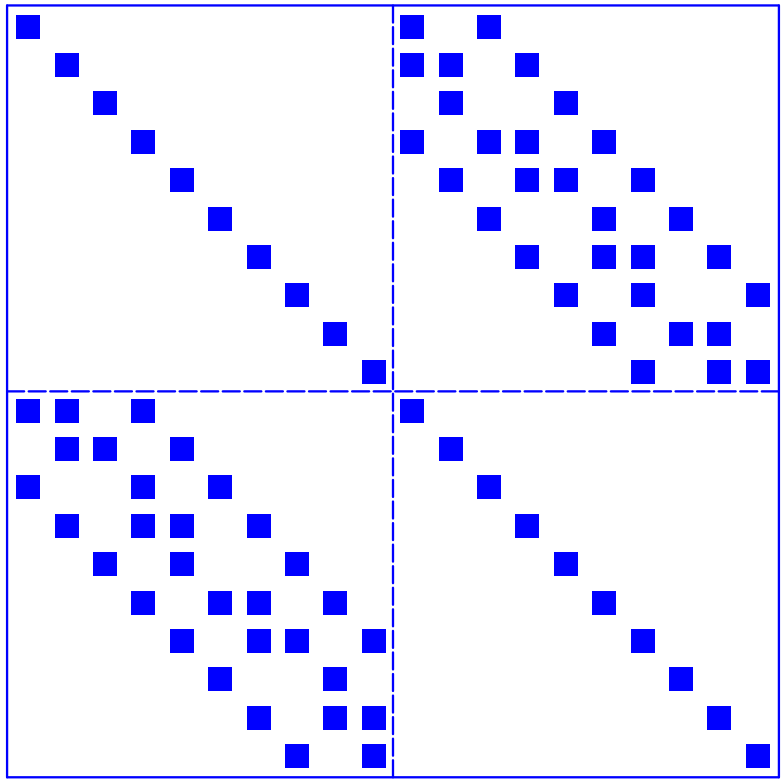
- General technique that can be exploited in many different ways to introduce parallelism – generally of order N.

- Constitutes one of the most successful techniques (when applicable) on (vector) supercomputers.

Simple example: Red-Black ordering.


18
9
19
10
20
6
16
7
17
8
13
4
14
5
15
1
11
2
12
3

Corresponding matrix



➨ **Observe: L-U solves (or SOR sweeps) will require only diagonal scalings + matrix-vector products with matrices of size $N/2$.**

Solution of Red-black systems

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.**

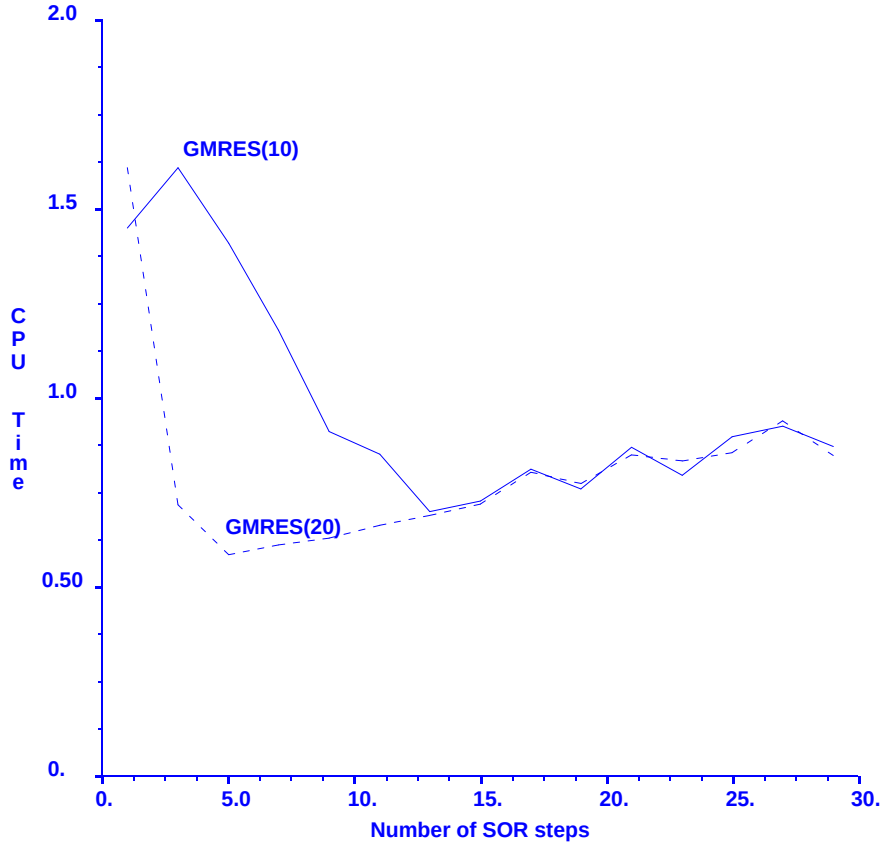
Method 2: SOR/SSOR(k) preconditioner.

- No such difficulty with SOR/SSOR...

- A 'more accurate preconditioner' means more SOR/ SSOR steps.

[SOR(k) $\longrightarrow$ 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:** $$(D_2 - F D_1^{-1} E) x_2 = b_2 - F D_1^{-1} b_1.$$

- 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?

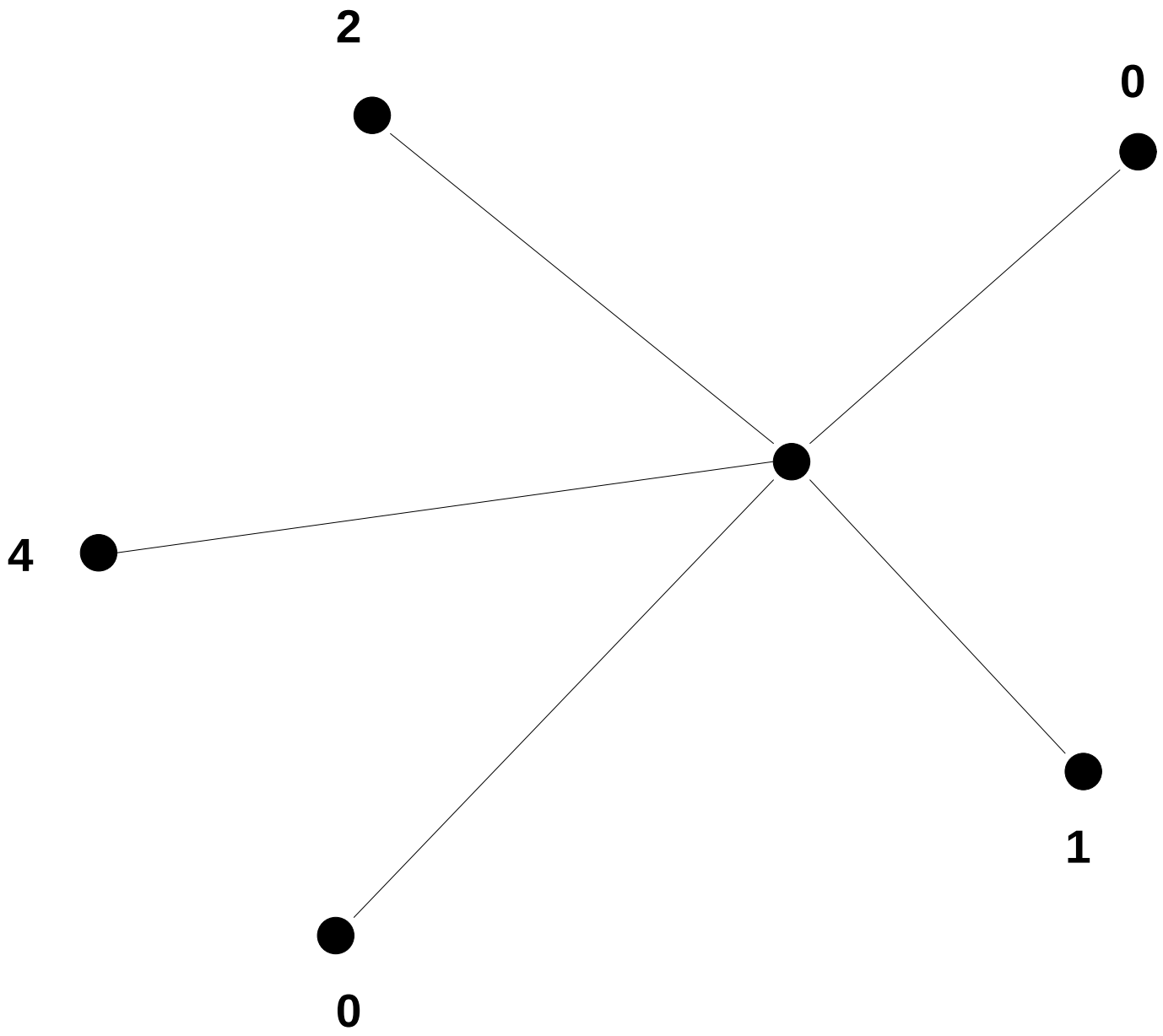
Answer: **Multicoloring** **&** **independent sets**

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**

$$Color(i) = \min\{k \neq 0 | k \neq Color(j), \forall\, j \in \text{Adj (i)}\}$$

Adj(i) = set of nearest neighbors of i = $\{k \mid a_{ik} \neq 0\}$.


2
0
4
1
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: $O(|E|+|V|)$.

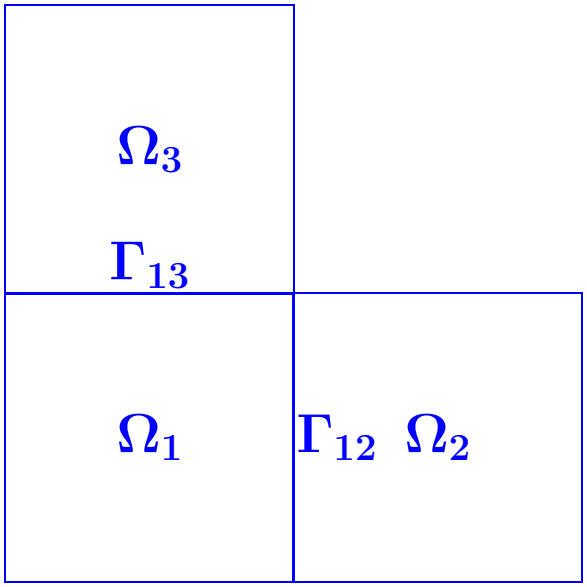
Domain Decomposition

Problem:

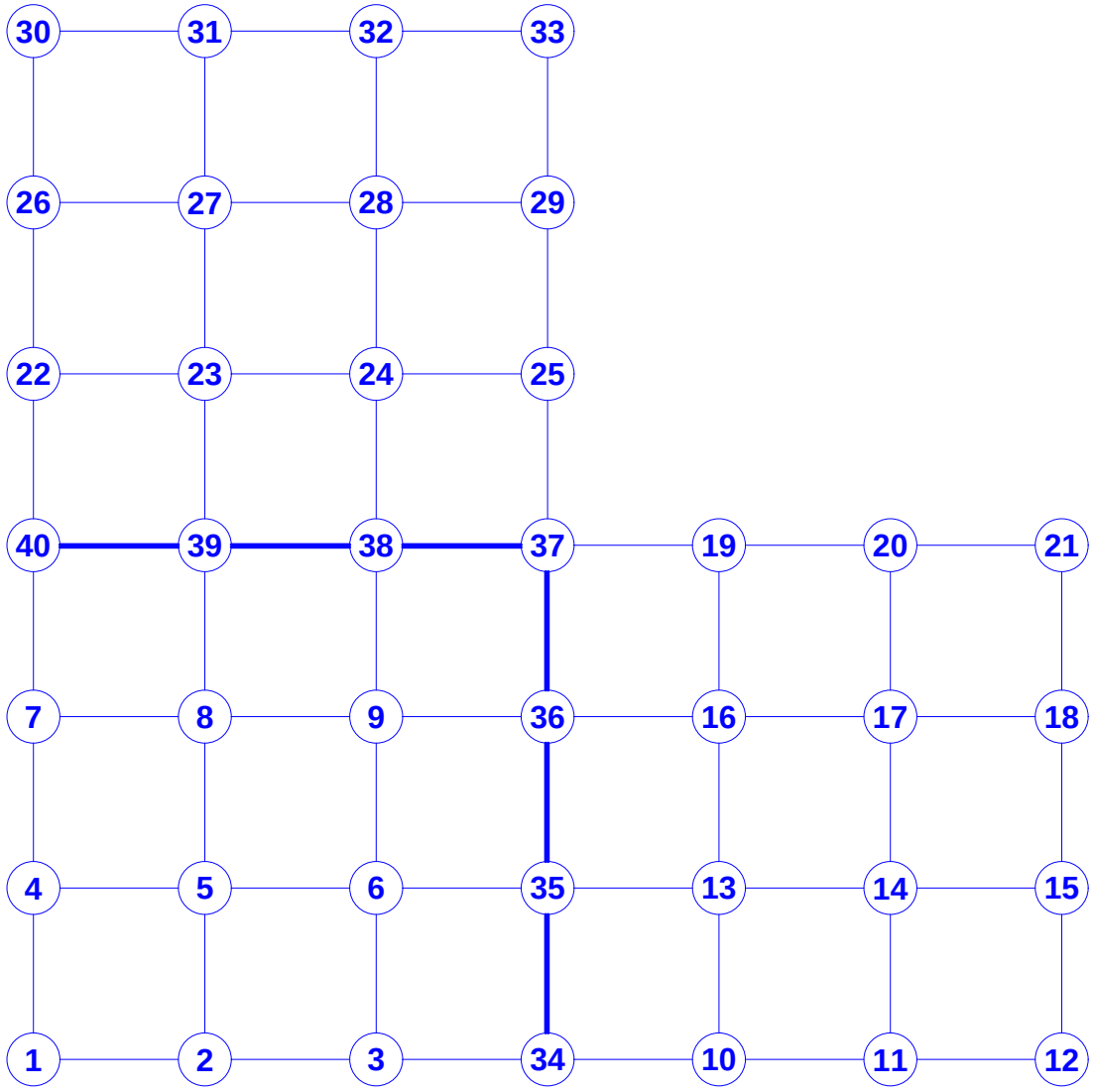
$$\begin{cases} \Delta u = & f \text{ in } \Omega \\ u = u_\Gamma & \text{on } \Gamma = \partial\Omega. \end{cases}$$

Domain:

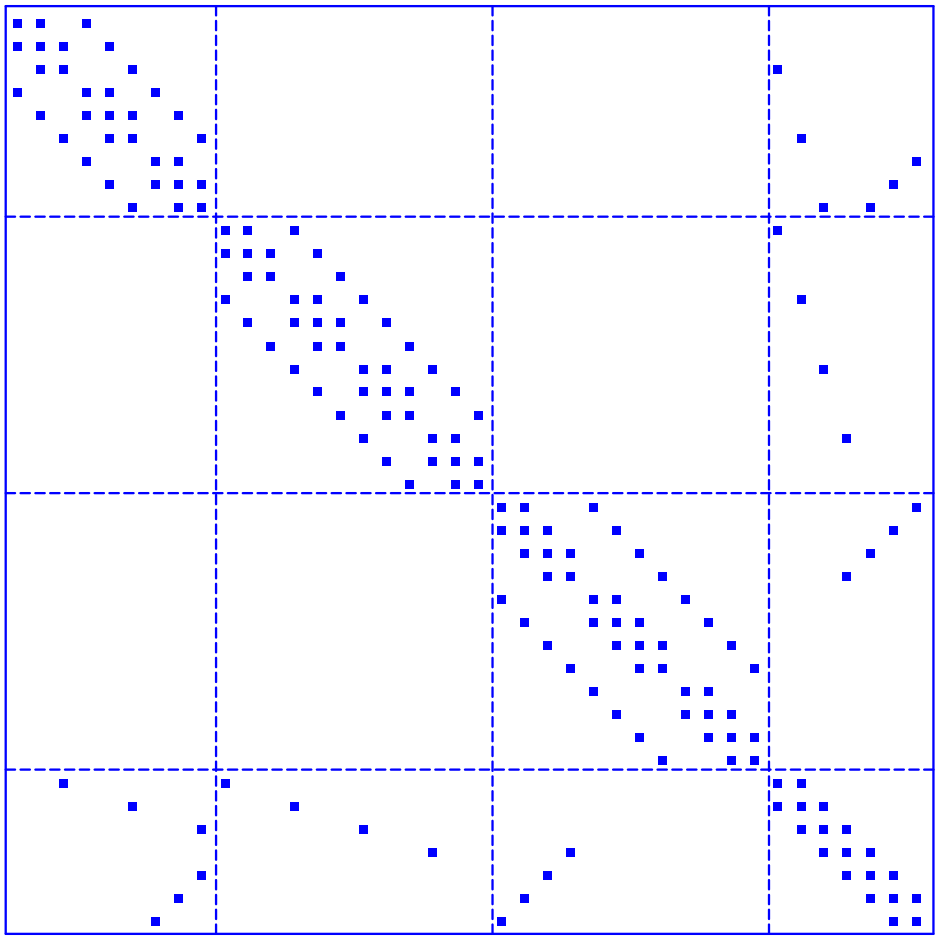
$$\Omega = \bigcup_{i=1}^{s} \Omega_i,$$



➨ **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.**

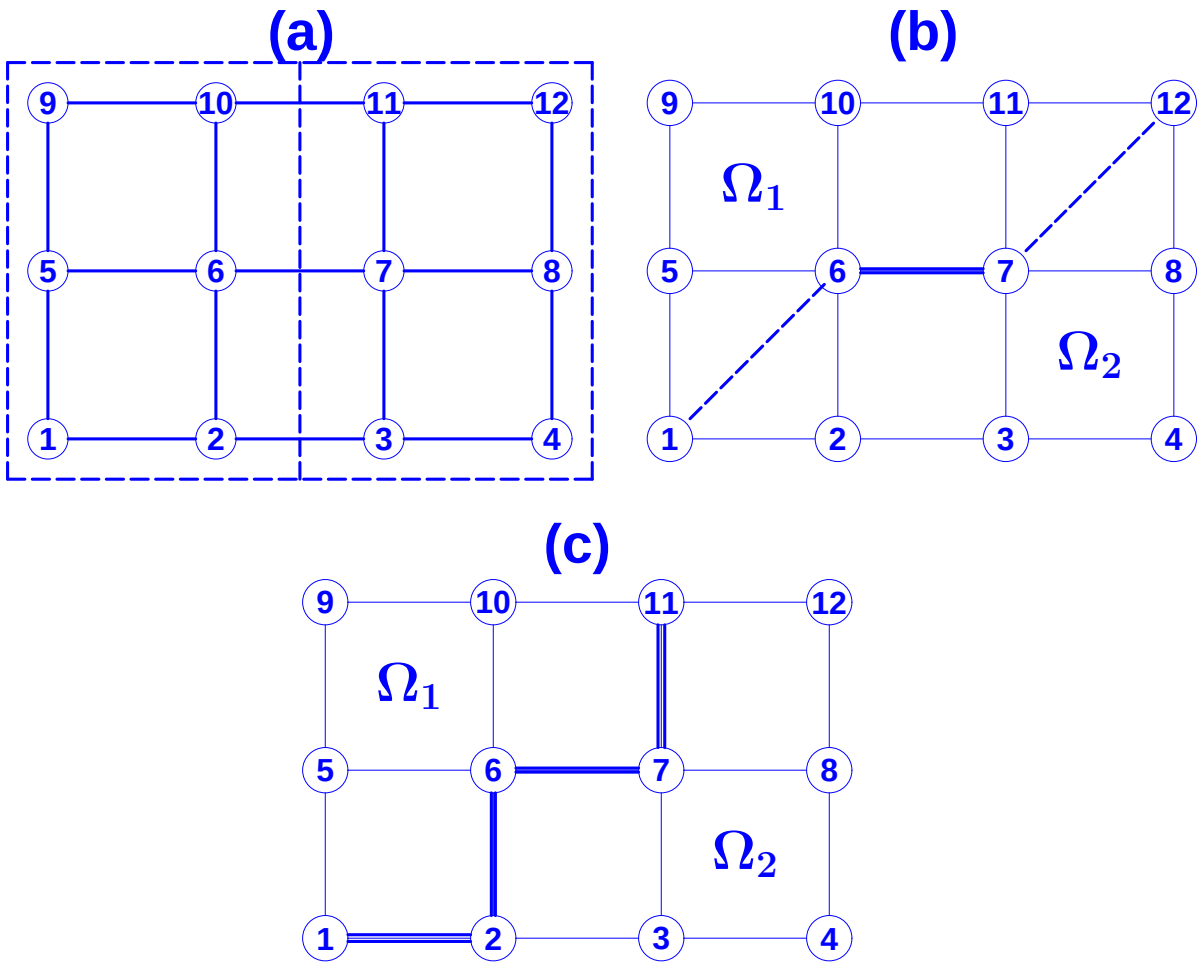


Discretization of domain



Coef£cient Matrix

Types of mappings



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

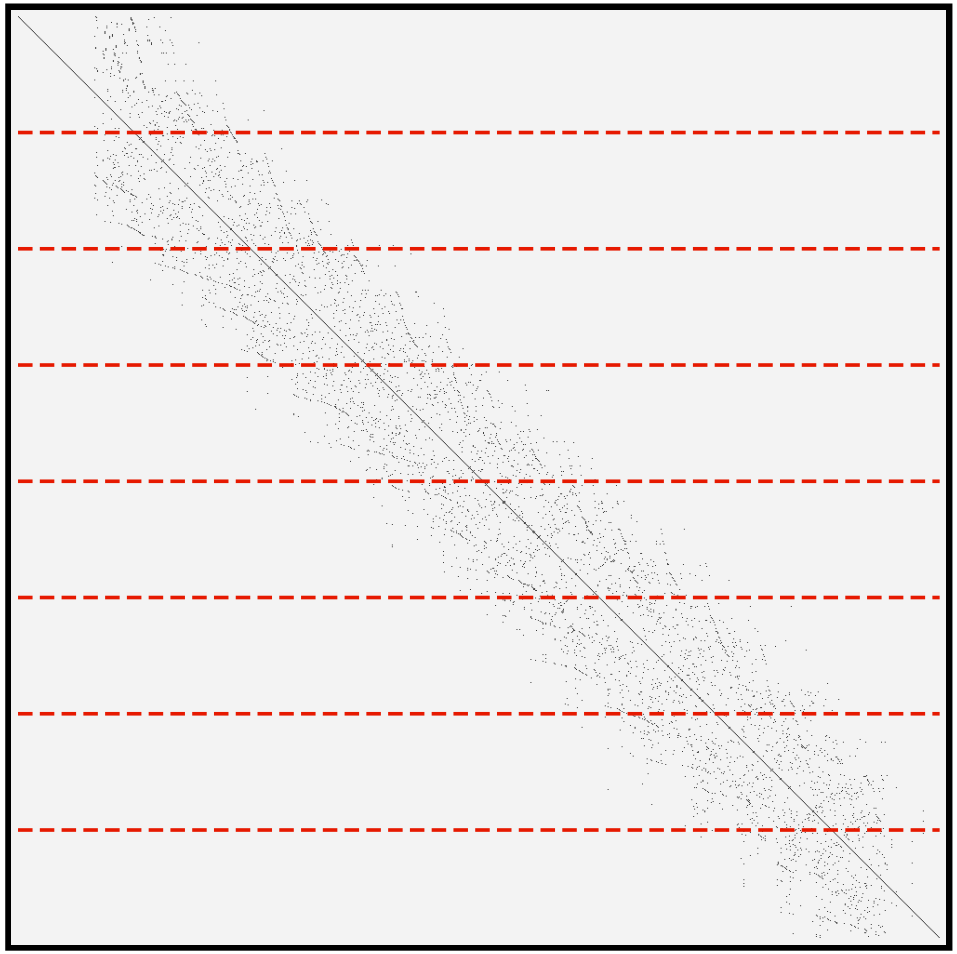
DISTRIBUTED SPARSE MATRICES

Generalization: Distributed Sparse Systems

- **Simple illustration:**

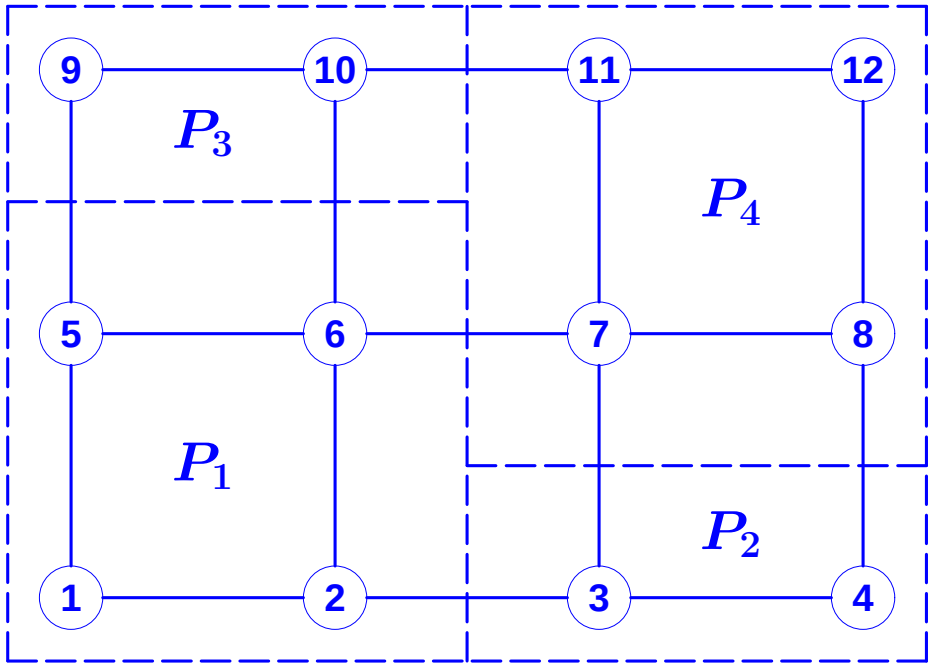
Block assignment

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.

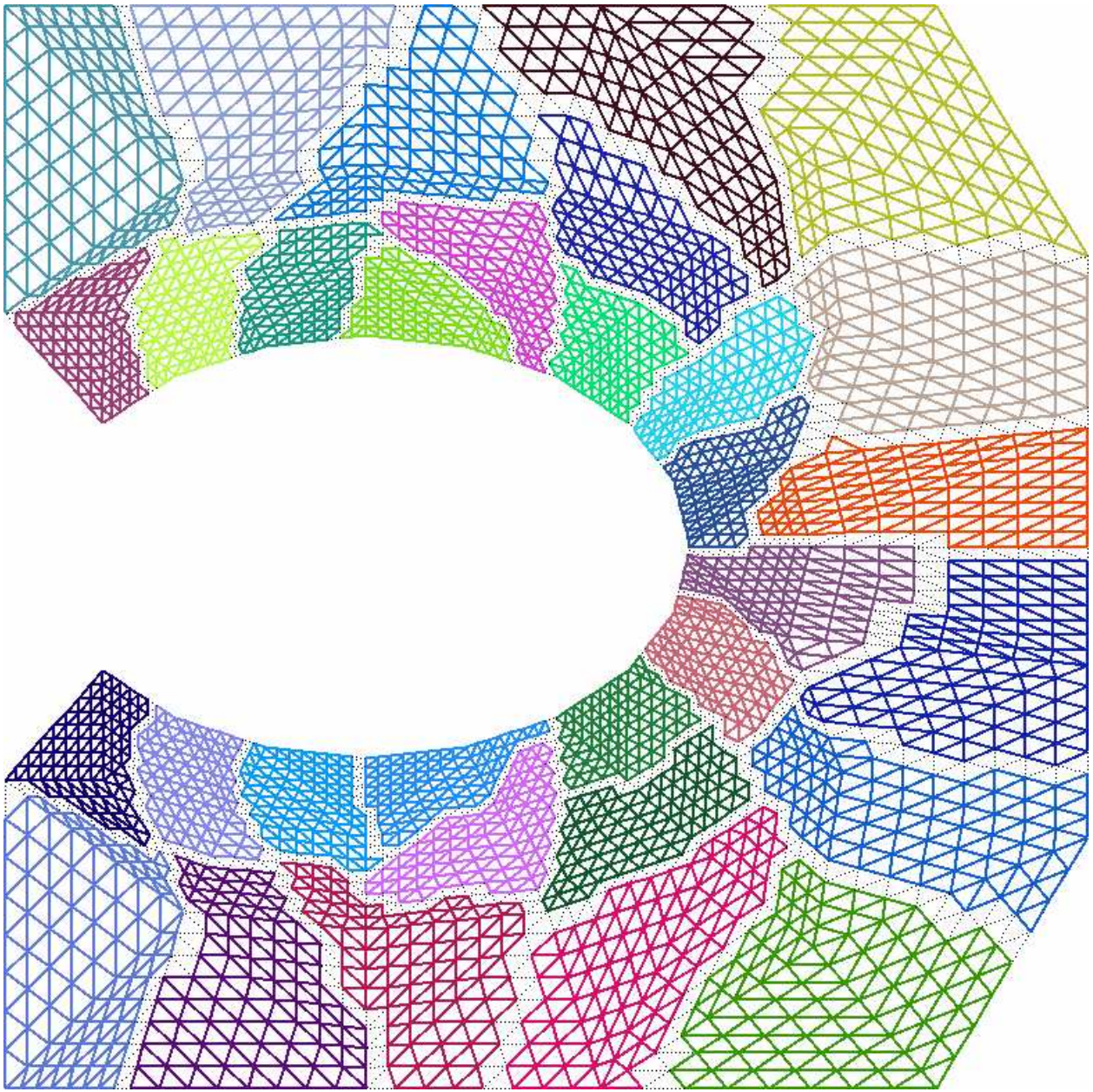
- Partition can be very general.

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

- 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, ...

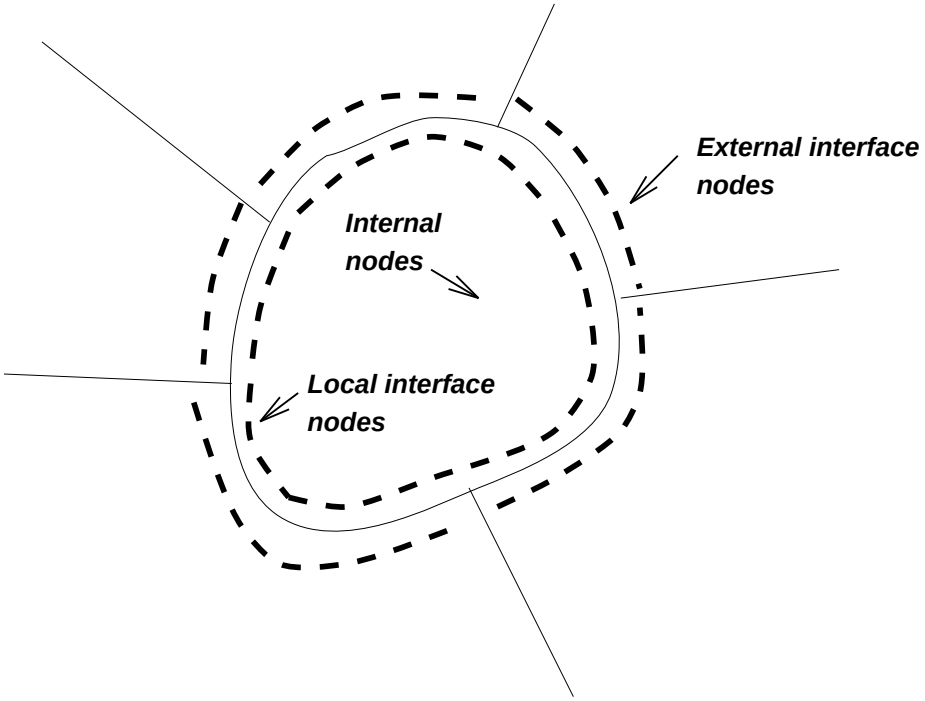


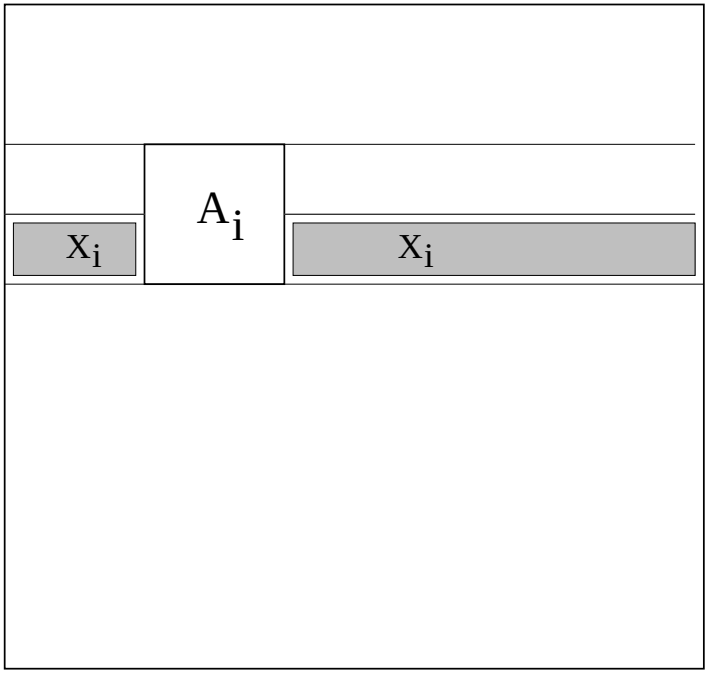
Distributed Sparse matrices (continued)

- **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?**

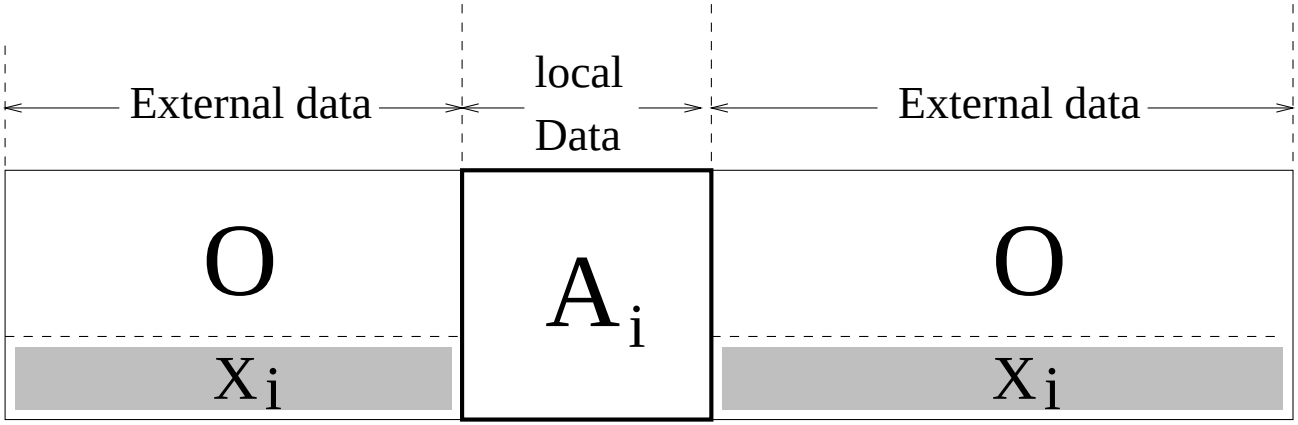
Two views of a distributed sparse matrix



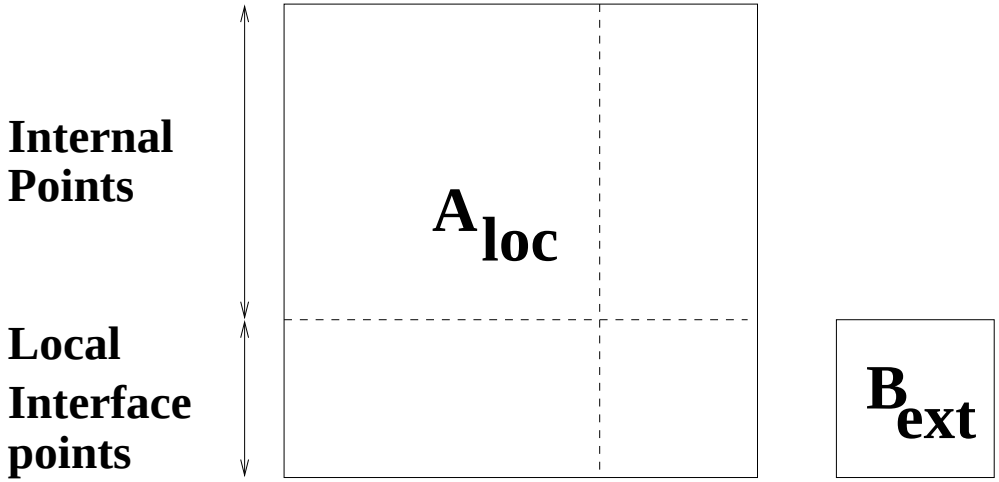


- **Local interface variables always ordered last.**

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
      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,ie
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).

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);
```

PARALLEL PRECONDITIONERS

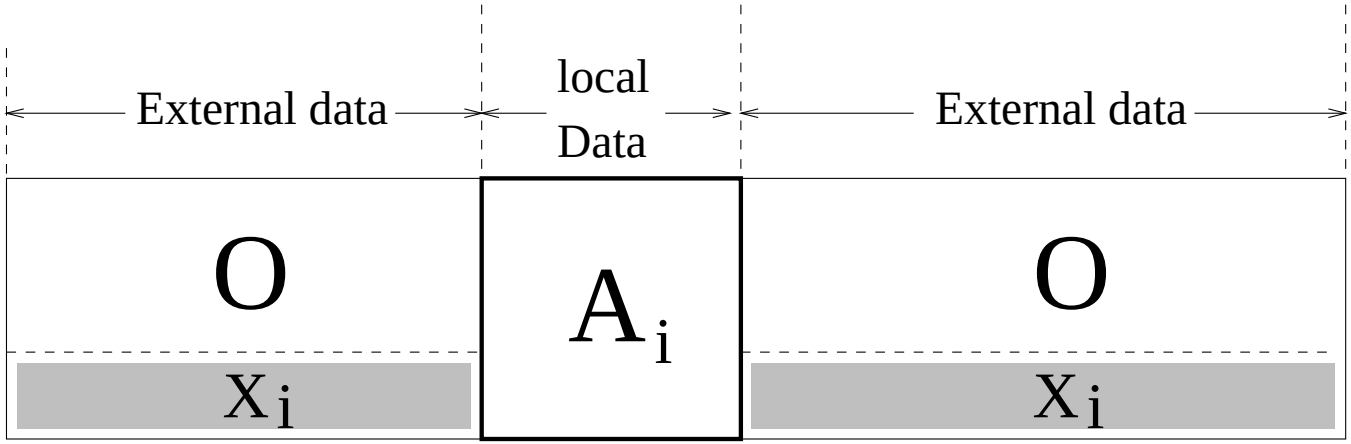
Three approaches:

- **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

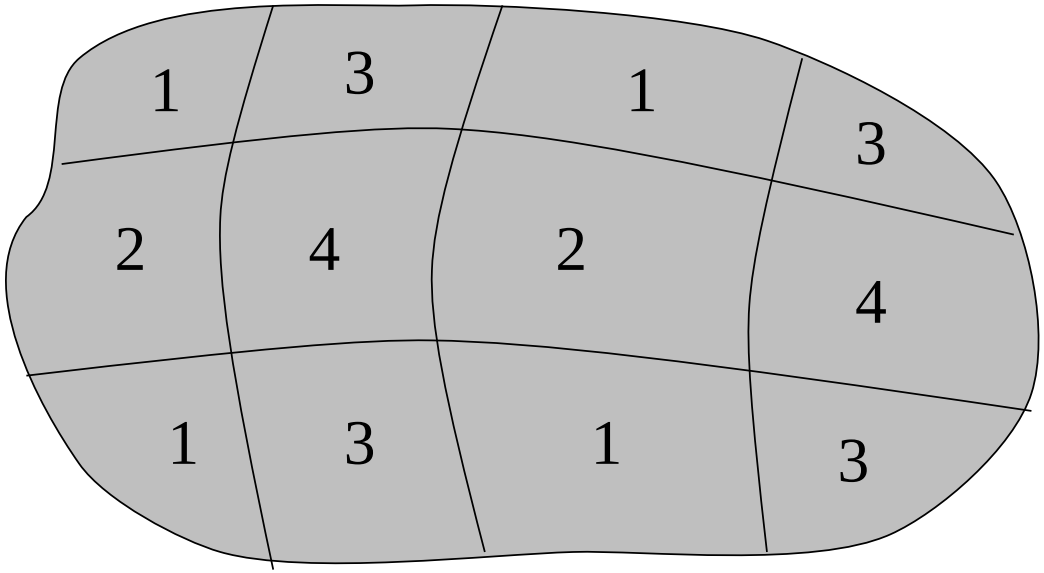
Local view of distributed matrix:



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.**



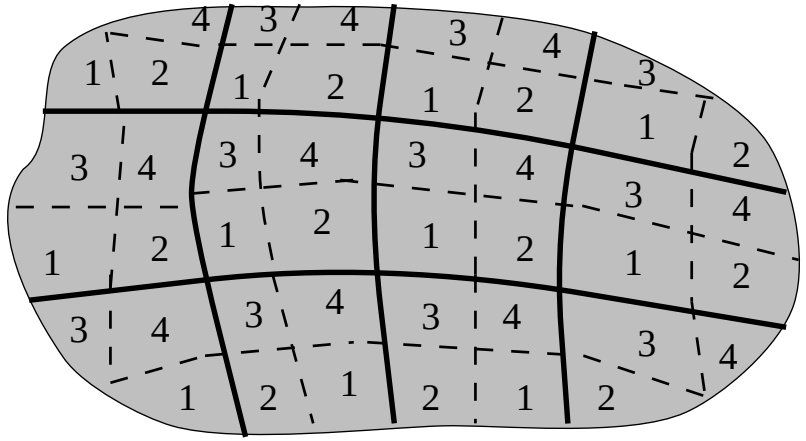
Multicolor Block SOR Iteration (Multiplicative Schwarz):

1. Do $col = 1, \ldots, numcols$
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

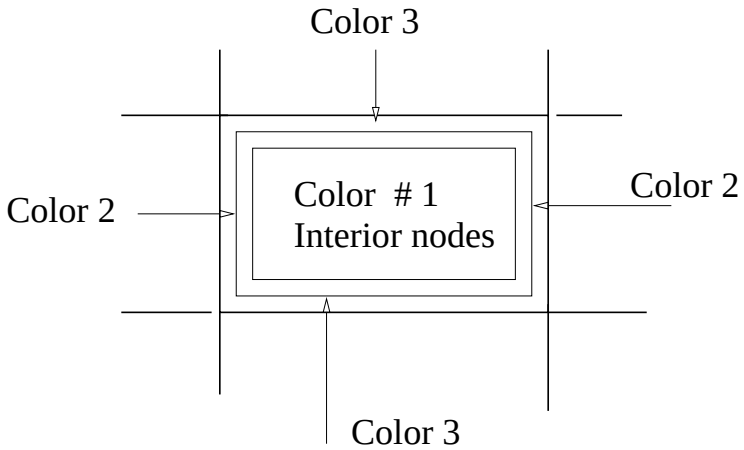
Breaking the sequential color loop

- **"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.

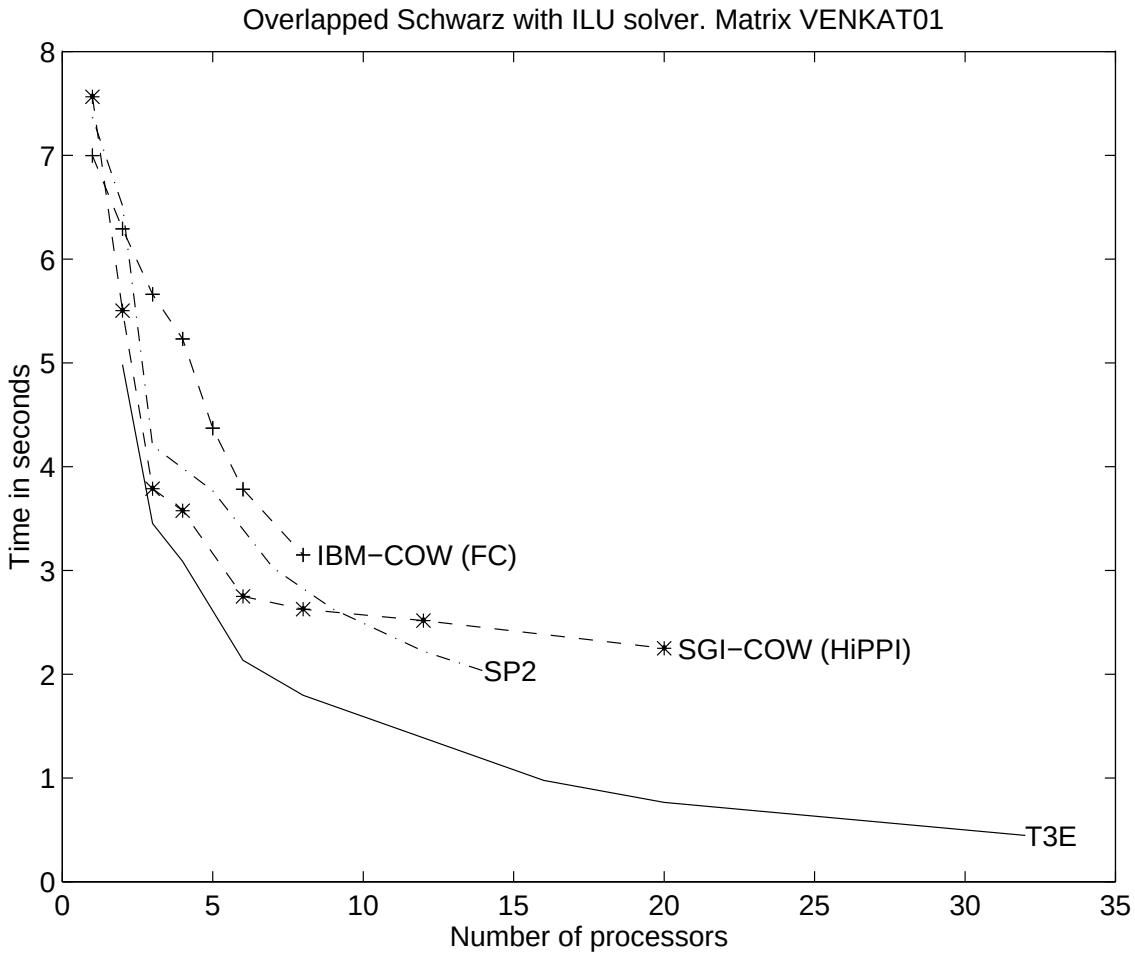
Local Solves

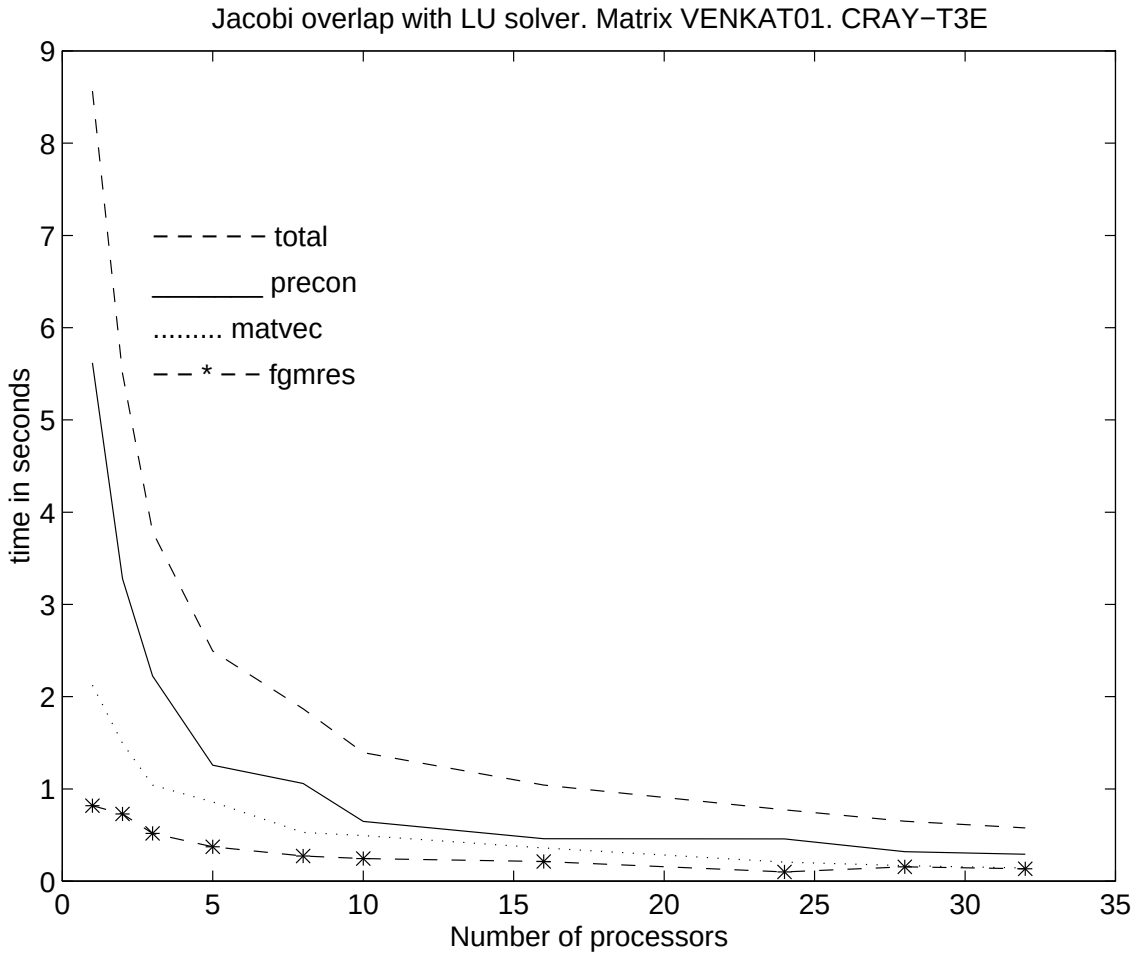
- Each local system $A_i \delta_i = r_i$ can be solved in three ways:

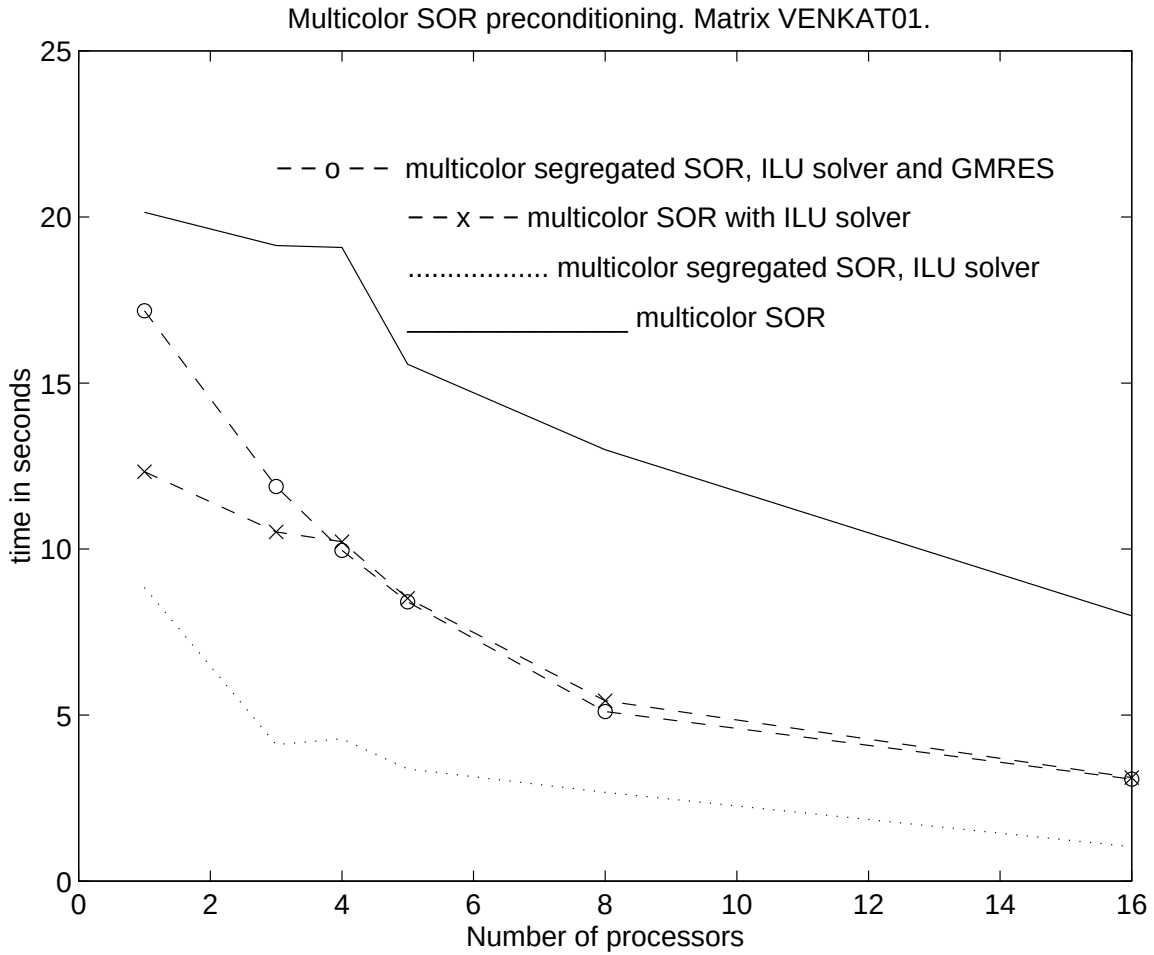
1. By a (sparse) direct solver
2. Using a standard preconditioned Krylov solver
3. Doing a backward-forward solution associated with an accurate ILU (e.g. ILUT) precondioner

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

Performance comparison for different machines




Jacobi overlap with LU solver. Matrix VENKAT01. CRAY−T3E
time in seconds
Number of processors
total
precon
matvec
fgmres
0
1
2
3
4
5
6
7
8
9
0
5
10
15
20
25
30
35

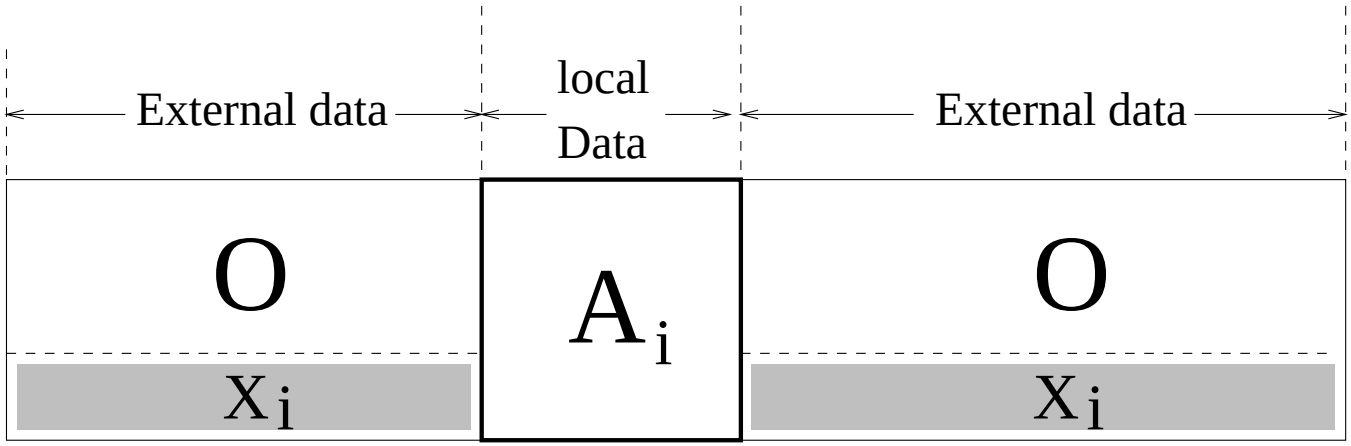

Multicolor SOR preconditioning. Matrix VENKAT01.
multicolor segregated SOR, ILU solver and GMRES
multicolor SOR with ILU solver
multicolor segregated SOR, ILU solver
multicolor SOR
time in seconds
Number of processors
0
5
10
15
20
25
0
2
4
6
8
10
12
14
16

SCHUR COMPLEMENT-BASED PRECONDITIONERS

Schur complement system

Local system can be written as

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



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

- **Local equations**

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

- **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

$$S_i = C_i - E_i B_i^{-1} F_i. \tag{3}$$

Structure of Schur complement system

- **Schur complement system**

$$Sy = g'$$

with

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

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**

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

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 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:

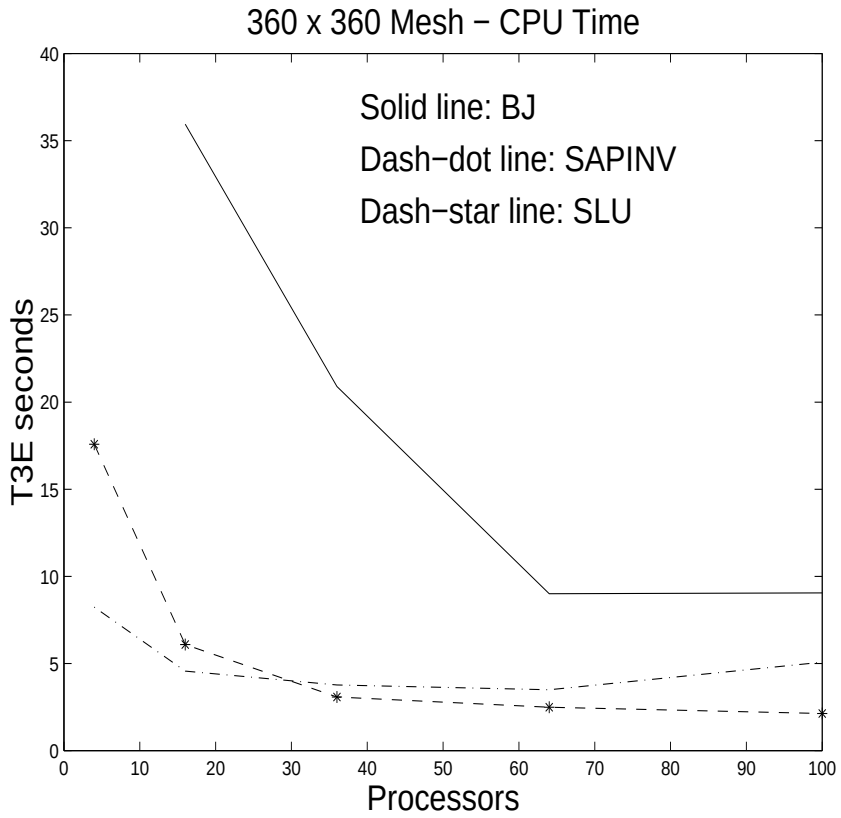
If $$A_i = \begin{pmatrix} L_{B_i} & 0 \\ E_i U_{B_i}^{-1} & L_{S_i} \end{pmatrix} \begin{pmatrix} U_{B_i} & L_{B_i}^{-1} F_i \\ 0 & U_{S_i} \end{pmatrix}$$ **Then** $L_{S_i} U_{S_i} = S_i$

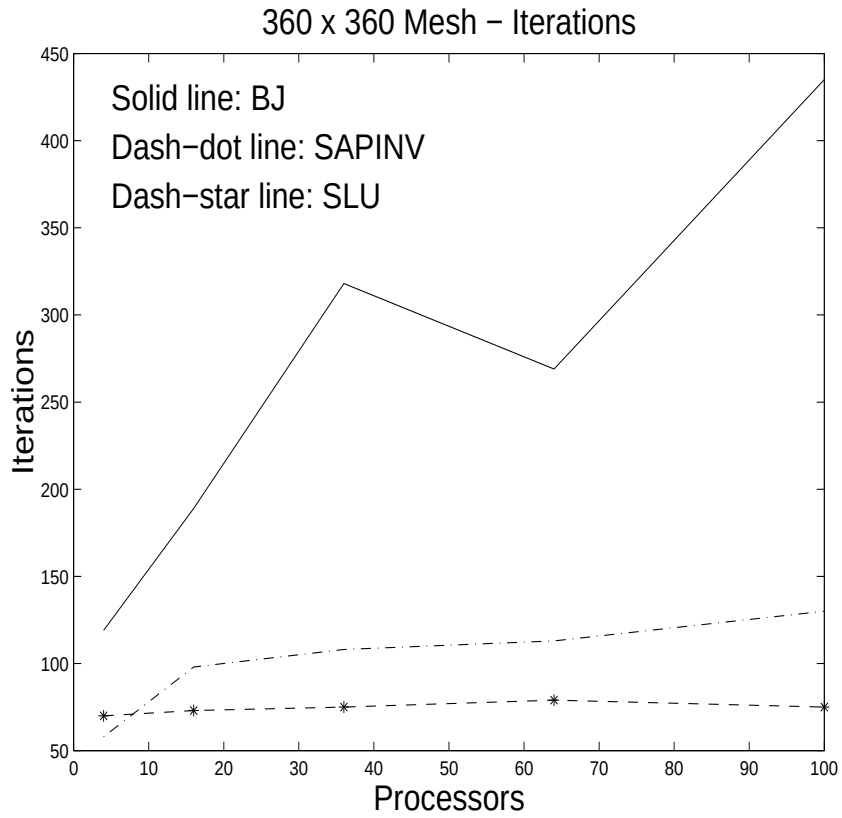
Name	Precon	lfil	4	8	16	24	36	40
raefsky1	SAPINV	10	14	13	10	11	8	8
		20	12	11	9	9	8	8
	SAPINVS	10	16	13	10	11	8	8
		20	13	11	9	9	8	8
	SLU	10	215	197	198	194	166	171
		20	48	50	40	42	41	41
	BJ	10	85	171	173	273	252	263
		20	82	170	173	271	259	259

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

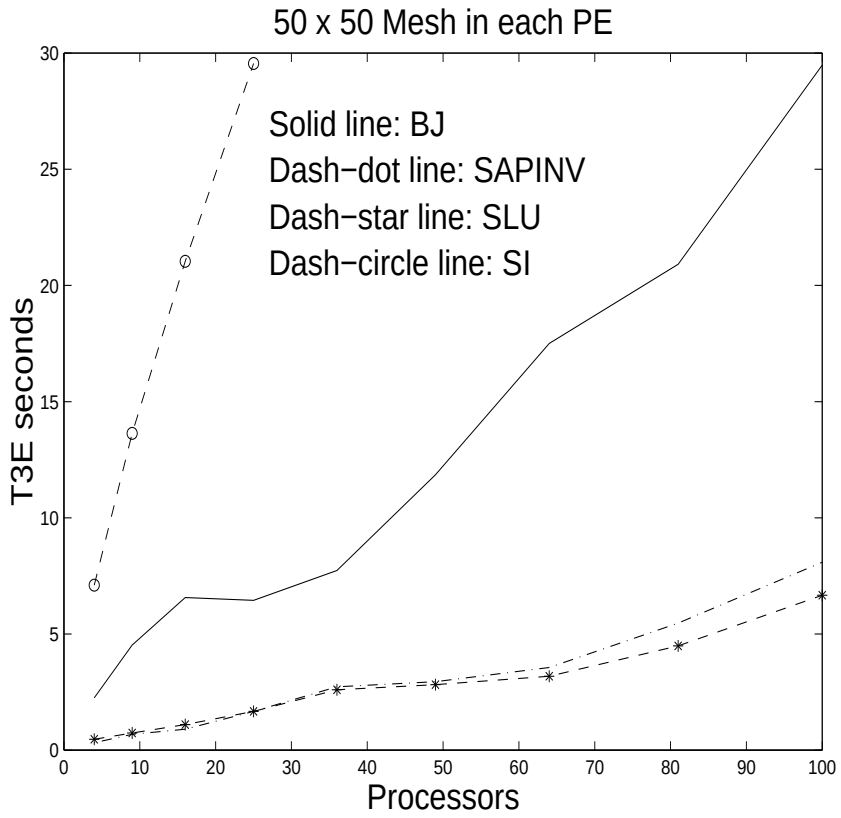
Number of FGMRES(20) iterations for the AF23560 problem.

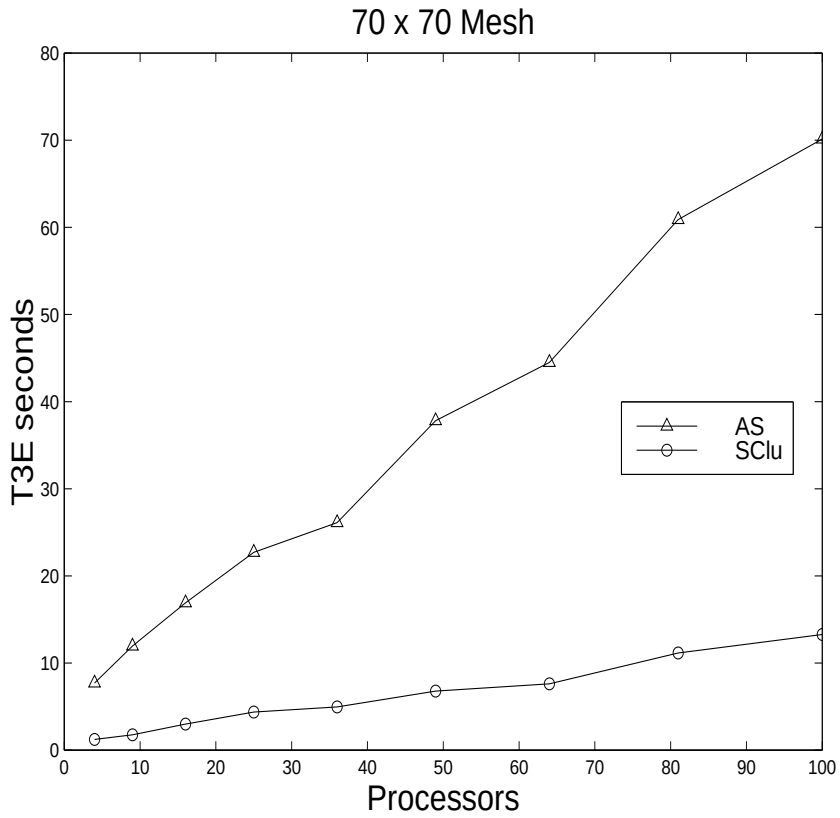




Times and iteration counts for solving a 360×360 discretized Laplacean problem with 3 different preconditioners using ¤exible GMRES(10).

➤ **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).**





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

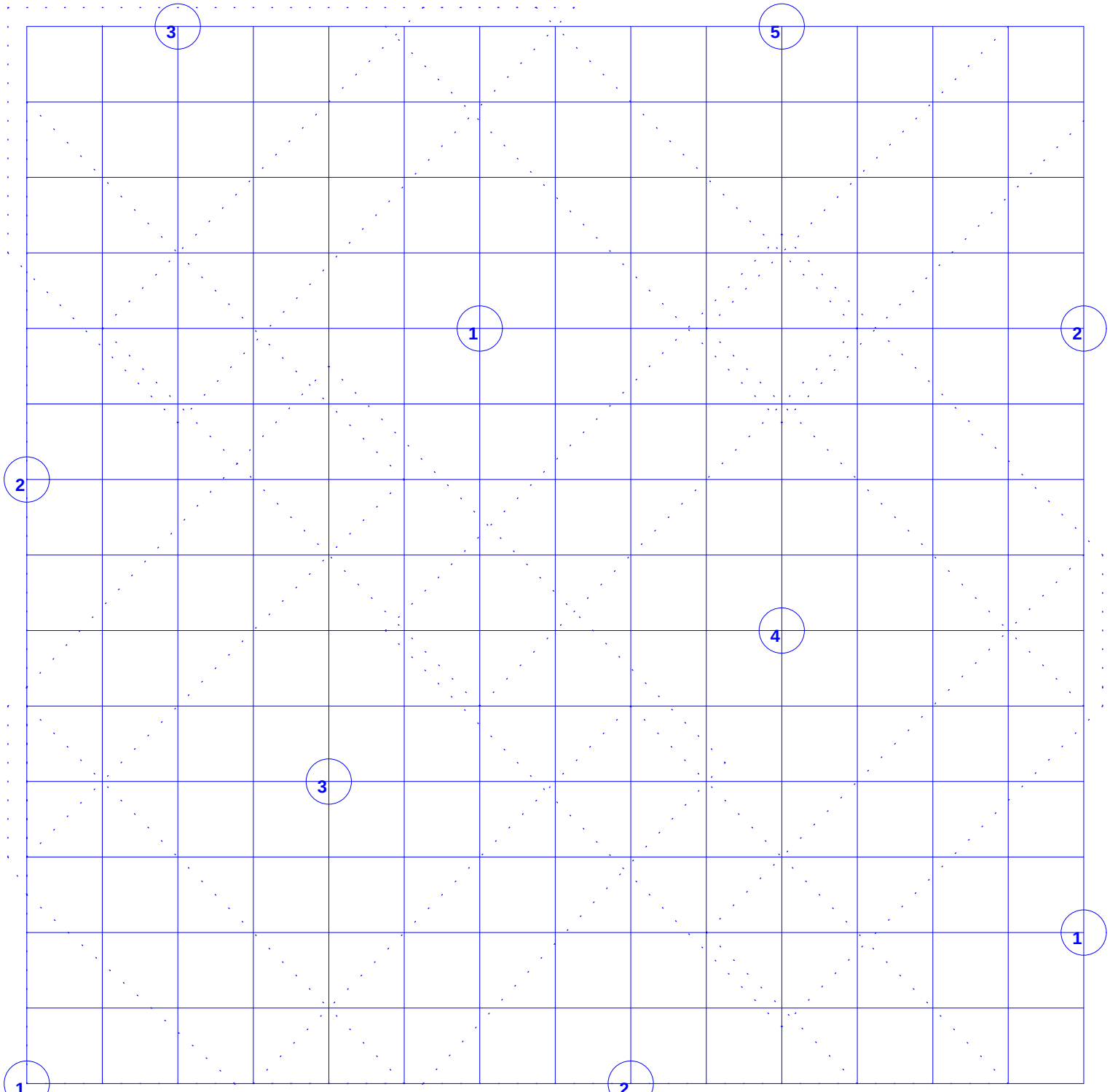
The Level Set Expansion Algorithm

- **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 15×15 £ve – point Finite Difference grid.

- **See [Goehring-Saad '94, See Cai-Saad '95]**


3
5
1
2
2
4
3
1
1
2

Alternative criteria

Criterion commonly used: Partition so that

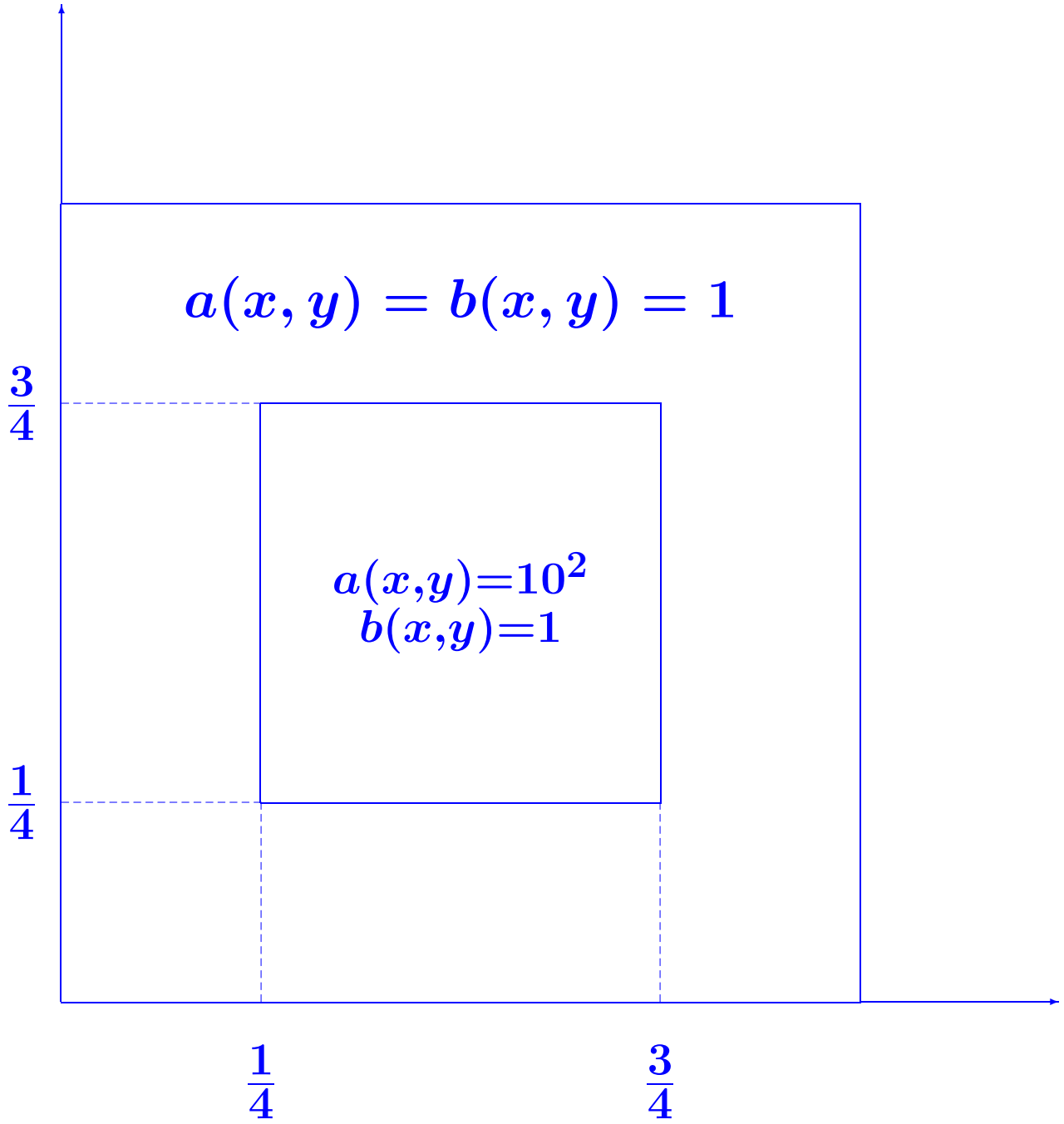
1. each subdomain has about the same number of vertices and

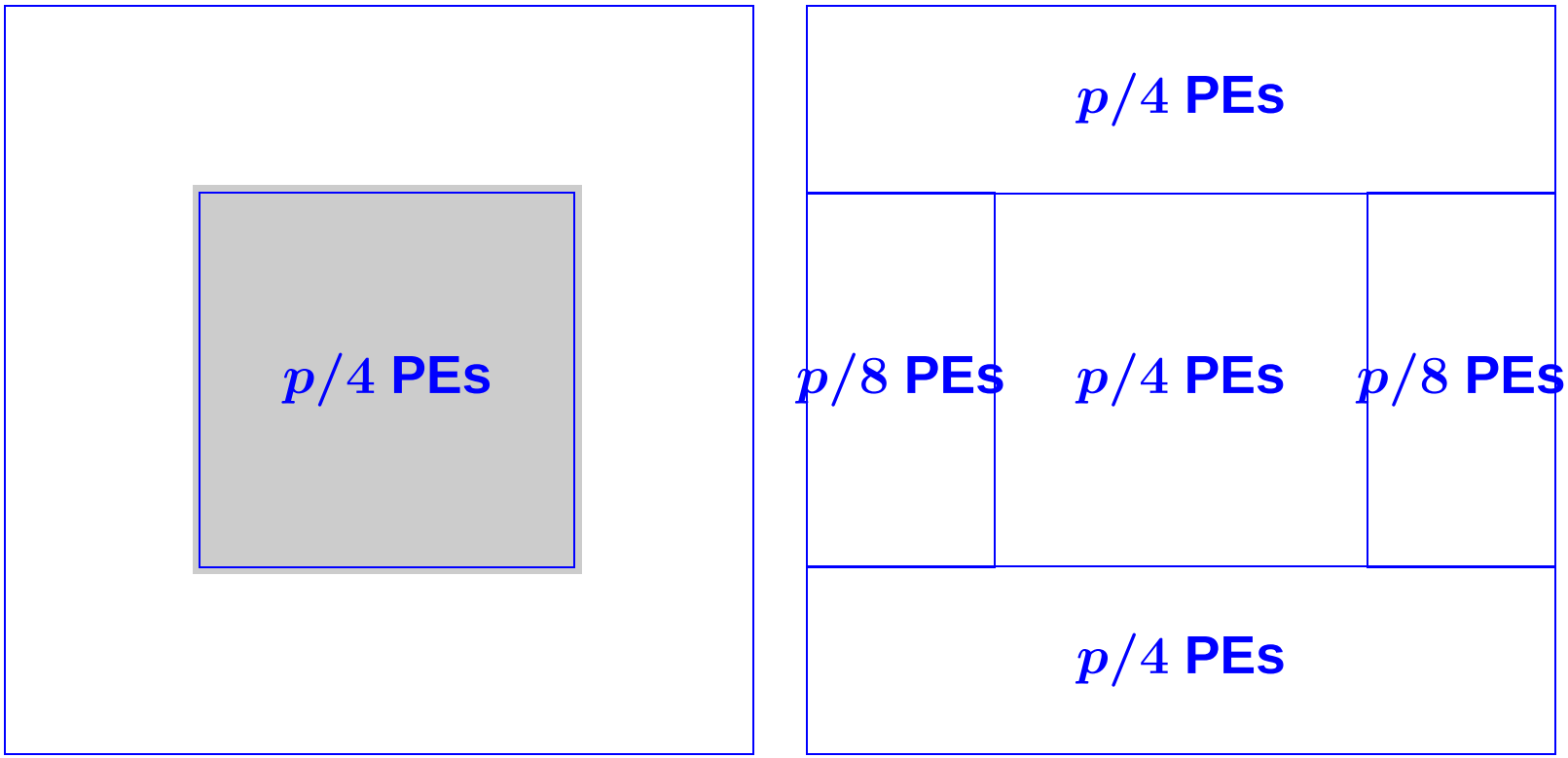
2. The total number of edge cuts is 'minimized'

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 coef£cient.





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: £rst 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.

	Time			Iterations		
PEs	**16**	**24**	**48**	**16**	**24**	**48**
DoubleStripe	**9.66**	**14.34**	**51.29**	**81**	**132**	**341**
DoubleStripe_m	**5.58**	**6.33**	**6.17**	**50**	**61**	**58**
ByHand	**6.39**	**12.73**	**9.98**	**52**	**120**	**87**
ByHand_m	**6.3**	**7.34**	**7.25**	**53**	**62**	**59**

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 £rst 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 £nd 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).

	Time	Iterations
Standard	175.27	161
Shortest-Path	122.95	106

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