Parallelism in Applied Math Calculations

Victor Eijkhout

374/394 spring 2013

ODEs and PDEs

Time-evolving phenomena: IVP (Initial Value Problem), usually Ordinary Differential Equations

Space-constraint phenomena: BVP (Boundary Value Problem), usually Partial Differential Equations

Approximating Differential Equations

Finite difference approximation

We turn the continuous problem into a discrete one, by looking at finite time/space steps.

Assume all functions are sufficiently smooth, and use Taylor series:

$$u(t+\Delta t) = u(t) + u'(t)\Delta t + u''(t)\frac{\Delta t^2}{2!} + u'''(t)\frac{\Delta t^3}{3!} + \cdots$$

This gives for u':

$$u'(t) = \frac{u(t+\Delta t) - u(t)}{\Delta t} + O(\Delta t^2)$$

So we approximate

$$u'(t) \approx \frac{u(t+\Delta t) - u(t)}{\Delta t}$$

and the "truncation error" is $O(\Delta t^2)$.

Finite differences 2

How does this help? In $u' = f(t, u)$ substitute

$$u'(t) \to \frac{u(t + \Delta t) - u(t)}{\Delta t}$$

giving

$$\frac{u(t + \Delta t) - u(t)}{\Delta t} = f(t, u(t))$$

or

$$u(t + \Delta t) = u(t) + \Delta t\, f(t, u(t))$$

Let $t_0 = 0$, $t_{k+1} = t_k + \Delta t = \cdots = (k + 1)\Delta t$, $u(t_k) = u_k$:

$$u_{k+1} = u_k + \Delta t\, f(t_k, u_k)$$

Discretization
'Explicit Euler' or 'Euler forward'.

Does this compute something close to the true solution?
'Discretization error'

Boundary value problems

Consider $u''(x) = f(x, u, u')$ for $x \in [a, b]$ where $u(a) = u_a$, $u(b) = u_b$ in 1D and

$$-u_{xx}(\bar{x}) - u_{yy}(\bar{x}) = f(\bar{x}) \text{ for } x \in \Omega = [0, 1]^2 \text{ with } u(\bar{x}) = u_0 \text{ on } \delta\Omega. \tag{1}$$

in 2D.

Approximation of 2nd order derivatives

Taylor series (write h for δx):

$$u(x+h) = u(x) + u'(x)h + u''(x)\frac{h^2}{2!} + u'''(x)\frac{h^3}{3!} + u^{(4)}(x)\frac{h^4}{4!} + u^{(5)}(x)\frac{h^5}{5!} + \cdots$$

and

$$u(x-h) = u(x) - u'(x)h + u''(x)\frac{h^2}{2!} - u'''(x)\frac{h^3}{3!} + u^{(4)}(x)\frac{h^4}{4!} - u^{(5)}(x)\frac{h^5}{5!} + \cdots$$

Subtract:

$$u(x+h) + u(x-h) = 2u(x) + u''(x)h^2 + u^{(4)}(x)\frac{h^4}{12} + \cdots$$

so

$$u''(x) = \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} - u^{(4)}(x)\frac{h^4}{12} + \cdots$$

Numerical scheme:

$$-\frac{u(x+h) - 2u(x) + u(x-h)}{h^2} = f(x, u(x), u'(x))$$

(2nd order PDEs are very common!)

This leads to linear algebra

$$-\frac{u(x+h)-2u(x)+u(x-h)}{h^2} = f(x, u(x), u'(x))$$

Equally spaced points on $[0,1]$: $x_k = kh$ where $h = 1/n$, then

$$-u_{k+1} + 2u_k - u_{k-1} = h^2 f(x_k, u_k, u'_k) \quad \text{for } k = 1, \ldots, n-1$$

Written as matrix equation:

$$\begin{pmatrix} 2 & -1 & & \\ -1 & 2 & -1 & \\ & \ddots & \ddots & \ddots \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \end{pmatrix} = \begin{pmatrix} h^2 f_1 + u_0 \\ h^2 f_2 \\ \vdots \end{pmatrix}$$

Matrix properties

- Very sparse, banded
- Symmetric (only because 2nd order problem)
- Sign pattern: positive diagonal, nonpositive off-diagonal (true for many second order methods)
- Positive definite (just like the continuous problem)

Initial Boundary value problem

Heat conduction in a rod $T(x,t)$ for $x \in [a,b]$, $t > 0$:

$$\frac{\partial}{\partial t} T(x,t) - \alpha \frac{\partial^2}{\partial x^2} T(x,t) = q(x,t)$$

- Initial condition: $T(x,0) = T_0(x)$
- Boundary conditions: $T(a,t) = T_a(t)$, $T(b,t) = T_b(t)$
- Material property: $\alpha > 0$ is thermal diffusivity
- Forcing function: $q(x,t)$ is externally applied heating.

The equation $u''(x) = f$ above is the steady state.

Discretization

Space discretization: $x_0 = a$, $x_n = b$, $x_{j+1} = x_j + \Delta x$
Time discretiation: $t_0 = 0$, $t_{k+1} = t_k + \Delta t$
Let T_j^k approximate $T(x_j, t_k)$

Space:

$$\frac{\partial}{\partial t} T(x_j, t) - \alpha \frac{T(x_{j-1}, t) - 2T(x_j, t) + T(x_{j+1}, t)}{\Delta x^2} = q(x_j, t)$$

Explicit time stepping:

$$\frac{T_j^{k+1} - T_j^k}{\Delta t} - \alpha \frac{T_{j-1}^k - 2T_j^k + T_{j+1}^k}{\Delta x^2} = q_j^k$$

Implicit time stepping:

$$\frac{T_j^{k+1} - T_j^k}{\Delta t} - \alpha \frac{T_{j-1}^{k+1} - 2T_j^{k+1} + T_{j+1}^{k+1}}{\Delta x^2} = q_j^{k+1}$$

Computational form: explicit

$$T_j^{k+1} = T_j^k + \frac{\alpha \Delta t}{\Delta x^2}(T_{j-1}^k - 2T_j^k + T_{j+1}^k) + \Delta t q_j^k$$

This has an explicit form:

$$\underset{\sim}{T}^{k+1} = \left(I + \frac{\alpha \Delta t}{\Delta x^2} K\right) \underset{\sim}{T}^k + \Delta t \underset{\sim}{q}^k$$

where K is the Laplace matrix

Computational form: implicit

$$T_j^{k+1} - \frac{\alpha \Delta t}{\Delta x^2}(T_{j-1}^k - 2T_j^k + T_{j+1}^k) = T_j^k + \Delta t q_j^k$$

This has an implicit form:

$$\left(I - \frac{\alpha \Delta t}{\Delta x^2} K\right) \underset{\sim}{T}^{k+1} = \underset{\sim}{T}^k + \Delta t \underset{\sim}{q}^k$$

Needs to solve a linear system in every time step

Stability of explicit scheme

Needs

$$\Delta t < \frac{\Delta x^2}{2\alpha}$$

big restriction on size of time steps

Stability of implicit scheme

Stability condition always satisfied: method always stable.

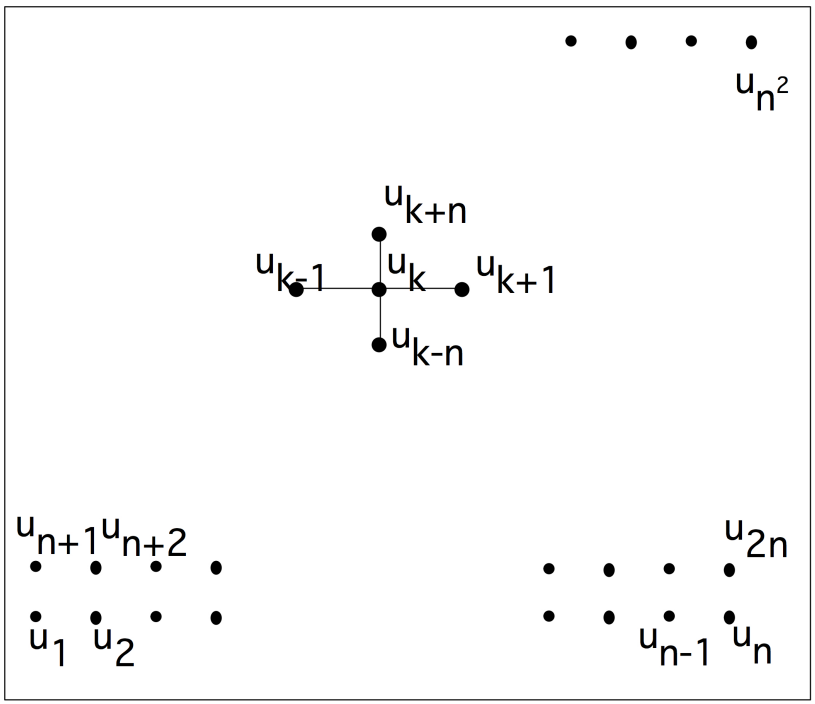
Sparse matrix in 2D case

Sparse matrices so far were tridiagonal: only in 1D case.

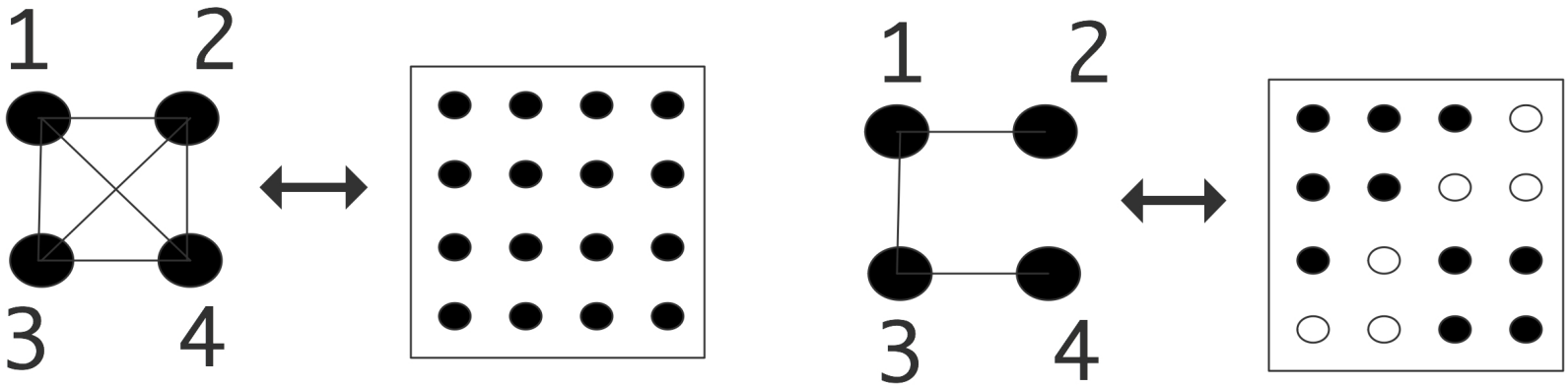
Two-dimensional: $-u_{xx} - u_{yy} = f$ on unit square $[0,1]^2$

Difference equation:

$$4u(x,y) - u(x+h,y) - u(x-h,y) - u(x,y+h) - u(x,y-h) = h^2 f(x,y)$$



Graph theory of sparse matrices



The graph view of things

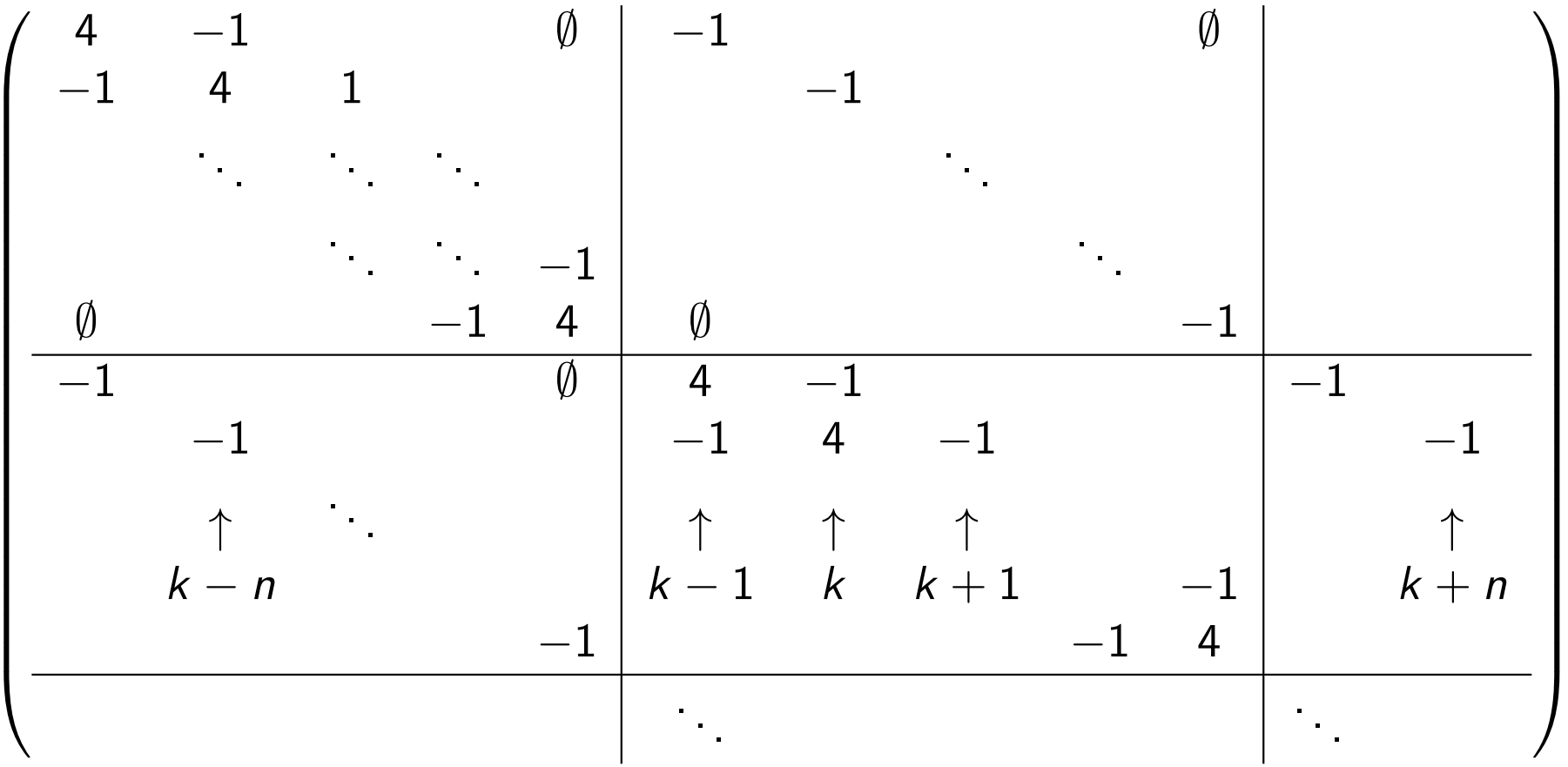
Poisson eq:

$$4u_k - u_{k-1} - u_{k+1} - u_{k-n} - u_{k+n} = f_k$$

Consider a graph where $\{u_k\}_k$ are the edges
and (u_i, u_j) is an edge iff $a_{ij} \neq 0$.

This is the (adjacency) graph of a sparse matrix.

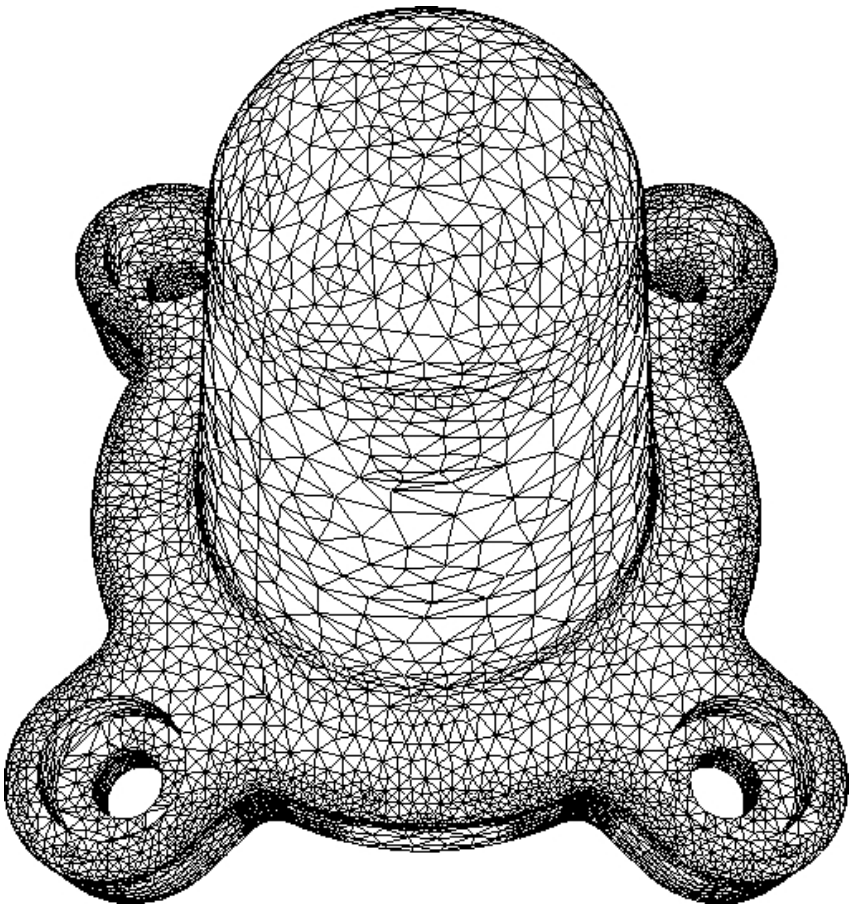
Sparse matrix from 2D equation



Matrix properties

- Very sparse, banded
- Symmetric (only because 2nd order problem)
- Sign pattern: positive diagonal, nonpositive off-diagonal (true for many second order methods)
- Positive definite (just like the continuous problem)
- Constant diagonals: only because of the constant coefficient differential equation

Realistic meshes



Iterative solution methods

Two different approaches

Solve $Ax = b$

Direct methods:

- Deterministic
- Exact up to machine precision
- Expensive (in time and space)

Iterative methods:

- Only approximate
- Cheaper in space and (possibly) time
- Convergence not guaranteed

Iterative methods

Choose any x_0 and repeat

$$x^{k+1} = Bx^k + c$$

until $\|x^{k+1} - x^k\|_2 < \epsilon$ or until $\frac{\|x^{k+1} - x^k\|_2}{\|x^k\|} < \epsilon$

Example of iterative solution

Example system

$$\begin{pmatrix} 10 & 0 & 1 \\ 1/2 & 7 & 1 \\ 1 & 0 & 6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 21 \\ 9 \\ 8 \end{pmatrix}$$

with solution $(2, 1, 1)$.

Suppose you know (physics) that solution components are roughly the same size, and observe the dominant size of the diagonal, then

$$\begin{pmatrix} 10 & & \\ & 7 & \\ & & 6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 21 \\ 9 \\ 8 \end{pmatrix}$$

might be a good approximation: solution $(2.1, 9/7, 8/6)$.

Iterative example′

Example system

$$\begin{pmatrix} 10 & 0 & 1 \\ 1/2 & 7 & 1 \\ 1 & 0 & 6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 21 \\ 9 \\ 8 \end{pmatrix}$$

with solution $(2, 1, 1)$.

Also easy to solve:

$$\begin{pmatrix} 10 & & \\ 1/2 & 7 & \\ 1 & 0 & 6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 21 \\ 9 \\ 8 \end{pmatrix}$$

with solution $(2.1, 7.95/7, 5.9/6)$.

Abstract presentation

- To solve $Ax = b$; too expensive; suppose $K \approx A$ and solving $Kx = b$ is possible
- Define $Kx_0 = b$, then error correction $x_0 = x + e_0$, and $A(x_0 - e_0) = b$
- so $Ae_0 = Ax_0 - b = r_0$; this is again unsolvable, so
- $K\tilde{e}_0$ and $x_1 = x_0 - \tilde{e}_0$.
- now iterate: $e_1 = x_1 - x$, $Ae_1 = Ax_1 - b = r_1$ et cetera

Error analysis

- One step

$$
\begin{aligned}
r_1 &= Ax_1 - b = A(x_0 - \tilde{e}_0) - b && (2)\\
&= r_0 - AK^{-1}r_0 && (3)\\
&= (I - AK^{-1})r_0 && (4)
\end{aligned}
$$

- Inductively: $r_n = (I - AK^{-1})^n r_0$ so $r_n \downarrow 0$ if $|\lambda(I - AK^{-1})| < 1$
  Geometric reduction (or amplification!)
- This is 'stationary iteration': every iteration step the same.
  Simple analysis, limited applicability.

General form of iterative methods

Iterative scheme:

$$x_{i+1} = x_0 + K^{-1}\pi^{(i)}(AK^{-1})r_0$$

Multiply by A and subtract b:

$$r_{i+1} = r_0 + \tilde{\pi}^{(i)}(AK^{-1})r_0$$

So:

$$r_i = \hat{\pi}^{(i)}(AK^{-1})r_0$$

where $\hat{\pi}^{(i)}$ is a polynomial of degree i with $\hat{\pi}^{(i)}(0) = 1$.

$\Rightarrow$ convergence theory

General form of iterative methods 2.

$$r_{i+1}\gamma_{i+1,i} = AK^{-1}r_i + \sum_{j\leq i} r_j\gamma_{ji}$$

and $\gamma_{i+1,i} = \sum_{j\leq i}\gamma_{ji}$.

Write this as $AK^{-1}R = RH$ where

$$H = \begin{pmatrix} -\gamma_{11} & -\gamma_{12} & \cdots & & \\ \gamma_{21} & -\gamma_{22} & -\gamma_{23} & \cdots & \\ 0 & \gamma_{32} & -\gamma_{33} & -\gamma_{34} & \\ \emptyset & \ddots & \ddots & \ddots & \ddots \end{pmatrix}$$

H is a Hessenberg matrix, and note zero column sums.

Divide A out:

$$x_{i+1}\gamma_{i+1,i} = K^{-1}r_i + \sum_{j\leq i} x_j\gamma_{ji}$$

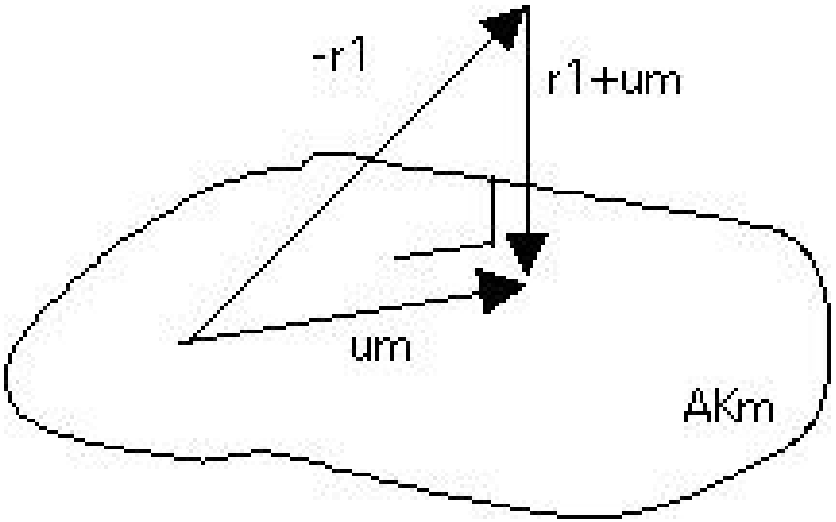
Orthogonality

Idea one:

> *If you can make all your residuals orthogonal to each other, and the matrix is of dimension n, then after n iterations you have to have converged: it is not possible to have an $n+1$-st residuals that is orthogonal and nonzero.*

Idea two:

> *The sequence of residuals spans a series of subspaces of increasing dimension, and by orthogonalizing the initial residual is projected on these spaces. This means that the errors will have decreasing sizes.*


-r1
r1+um
um
AKm

Full Orthogonalization Method

See also GMRES:

Let r_0 be given
For $i \geq 0$:
  let $s \leftarrow K^{-1} r_i$
  let $t \leftarrow AK^{-1} r_i$
  for $j \leq i$:
    let γ_j be the coefficient so that $t - \gamma_j r_j \perp r_j$
  for $j \leq i$:
    form $s \leftarrow s - \gamma_j x_j$
    and $t \leftarrow t - \gamma_j r_j$
  let $x_{i+1} = (\sum_j \gamma_j)^{-1} s$, $r_{i+1} = (\sum_j \gamma_j)^{-1} t$.

Conjugate Gradients

Basic idea:

$$r_i^t K^{-1} r_j = 0 \quad \text{if } i \neq j.$$

Split recurrences:

$$\begin{cases} x_{i+1} = x_i - \delta_i p_i \\ r_{i+1} = r_i - \delta_i A p_i \\ p_i = K^{-1} r_i + \sum_{j<i} \gamma_{ij} p_j, \end{cases}$$

Symmetric Positive Definite case

Three term recurrence is enough:

$$\begin{cases} x_{i+1} = x_i - \delta_i p_i \\ r_{i+1} = r_i - \delta_i A p_i \\ p_{i+1} = K^{-1} r_{i+1} + \gamma_i p_i \end{cases}$$

Preconditioned Conjugate Gradietns

Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$
for $i = 1, 2, \ldots$
  solve $Mz^{(i-1)} = r^{(i-1)}$
  $\rho_{i-1} = r^{(i-1)^T} z^{(i-1)}$
  if $i = 1$
    $p^{(1)} = z^{(0)}$
  else
    $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
    $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$
  endif
  $q^{(i)} = Ap^{(i)}$
  $\alpha_i = \rho_{i-1}/p^{(i)^T} q^{(i)}$
  $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence; continue if necessary
end

Observations on iterative methods

- Conjugate gradients: constant storage and inner products; works only for symmetric systems
- GMRES (like FOM): growing storage and inner products: restarting and numerical cleverness
- BiCGstab and QMR: relax the orthogonality

Choice of K

- The closer K is to A, the faster convergence.
- Diagonal and lower triangular choice mentioned above: let

$$A = D_A + L_A + U_A$$

  be a splitting into diagonal, lower triangular, upper triangular part, then
- Jacobi method: $K = D_A$ (diagonal part),
- Gauss-Seidel method: $K = D_A + L_A$ (lower triangle, including diagonal)
- SOR method: $K = \omega D_A + L_A$

Choice of K through incomplete LU

- Inspiration from direct methods: let $K = LU \approx A$

Gauss elimination:

```
for k,i,j:
   a[i,j] = a[i,j] - a[i,k] * a[k,j] / a[k,k]
```

Incomplete variant:

```
for k,i,j:
  if a[i,j] not zero:
    a[i,j] = a[i,j] - a[i,k] * a[k,j] / a[k,k]
```

$\Rightarrow$ sparsity of $L + U$ the same as of A

CG derived from minimization

Special case of SPD:

For which vector x with $\|x\| = 1$ is $f(x) = 1/2x^t Ax - b^t x$ minimal? (5)

Taking derivative:

$$f'(x) = Ax - b.$$

Update

$$x_{i+1} = x_i + p_i \delta_i$$

optimal value:

$$\delta_i = \operatorname*{argmin}_{\delta} \|f(x_i + p_i \delta)\| = \frac{r_i^t p_i}{p_1^t A p_i}$$

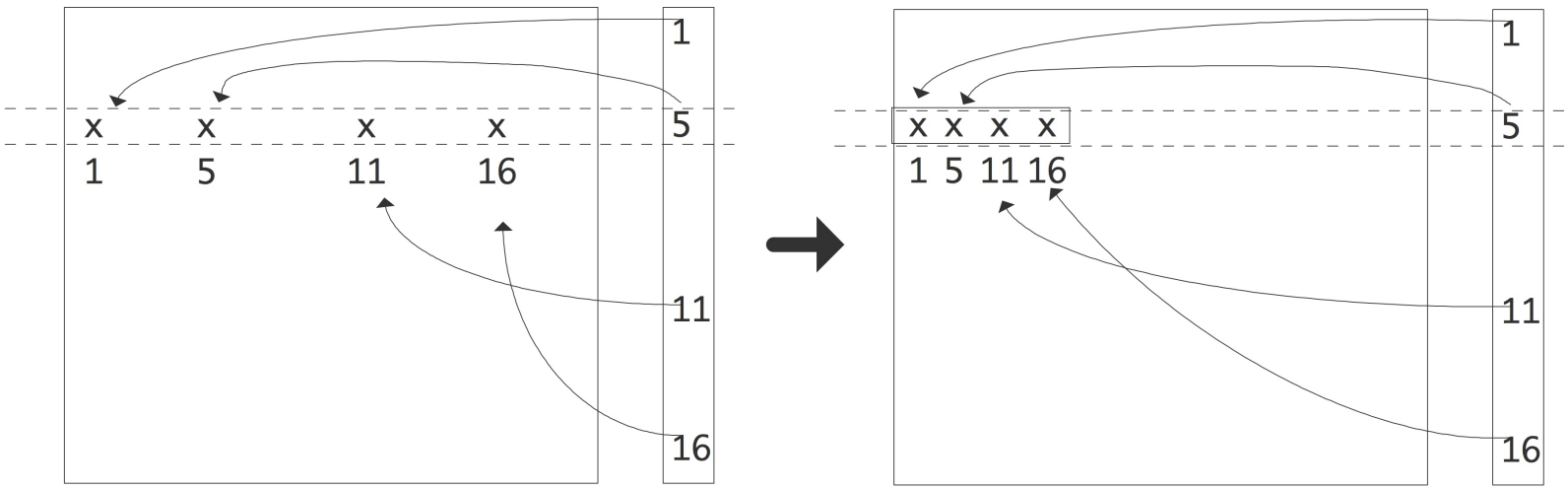
Other constants follow from orthogonality.

Sparse matrix-vector product

Sparse matrix storage

Matrix above has many zeros: n^2 elements but only $O(n)$ nonzeros. Big waste of space to store this as square array.

Matrix is called 'sparse' if there are enough zeros to make specialized storage feasible.



Compressed Row Storage

$$A = \begin{pmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{pmatrix} . \tag{6}$$

Compressed Row Storage (CRS): store all nonzeros by row, their column indices, pointers to where the columns start (1-based indexing):

`val`	10	-2	3	9	3	7	8	7	$3 \cdots 9$	13	4	2	-1
`col_ind`	1	5	1	2	6	2	3	4	$1 \cdots 5$	6	2	5	6

`row_ptr`	1	3	6	9	13	17	20

.

Sparse matrix operations

Most common operation: matrix-vector product

```
for (row=0; row<nrows; row++) {
   s = 0;
   for (icol=ptr[row]; icol<ptr[row+1]; icol++) {
      int col = ind[icol];
      s += a[aptr] * x[col];
      aptr++;
   }
   y[row] = s;
}
```

Operations with changes to the nonzero structure are much harder!

Locality?

Sparse matrix operations

Most common operation: matrix-vector product

```
for (row=0; row<nrows; row++) {
   s = 0;
   for (icol=ptr[row]; icol<ptr[row+1]; icol++) {
      int col = ind[icol];
      s += a[aptr] * x[col];
      aptr++;
   }
   y[row] = s;
}
```

Operations with changes to the nonzero structure are much harder!

Indirect addressing of `x` gives low spatial and temporal locality.

Parallel matrix-vector product

- Assume a division by block rows
- Every processor p has a set of row indices I_p

Mvp on processor p:

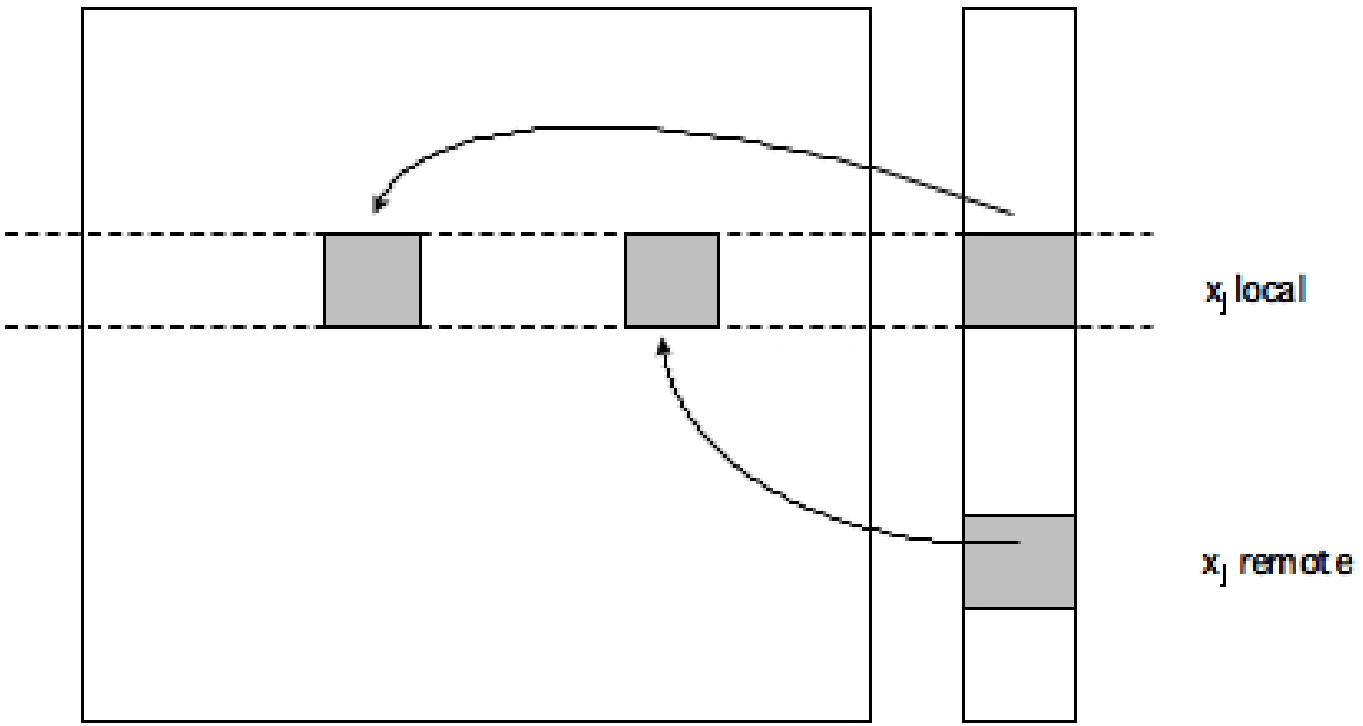
$$\forall_i : y_i = \sum_j a_{ij} x_j$$

$$\forall_i : y_i = \sum_q \sum_{j \in I_q} a_{ij} x_j$$

Local and remote parts:

$$\forall_i : y_i = \sum_{j \in I_p} a_{ij} x_j + \sum_{q \neq p} \sum_{j \in I_q} a_{ij} x_j$$

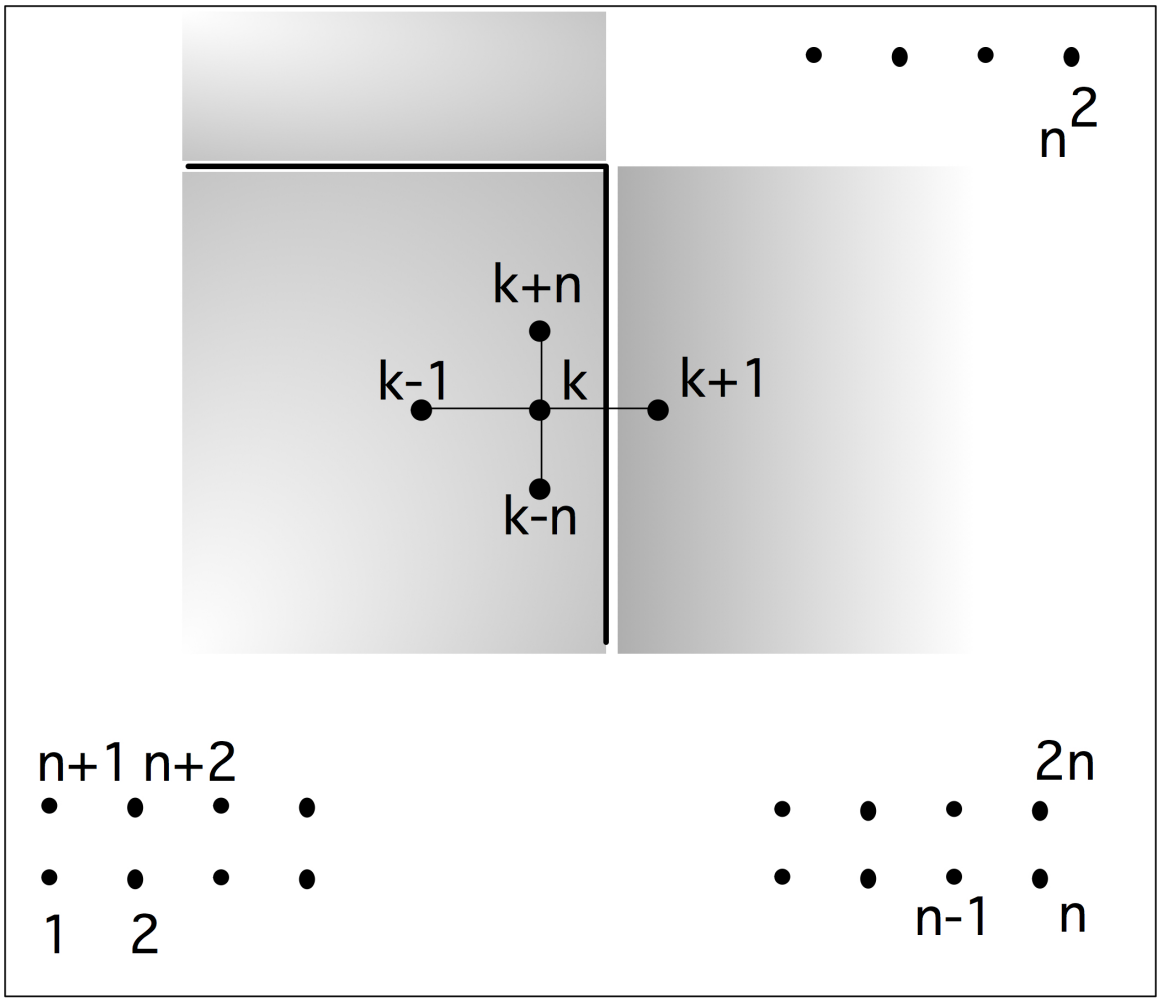
Local part I_p can be executed right away, I_q requires communication.



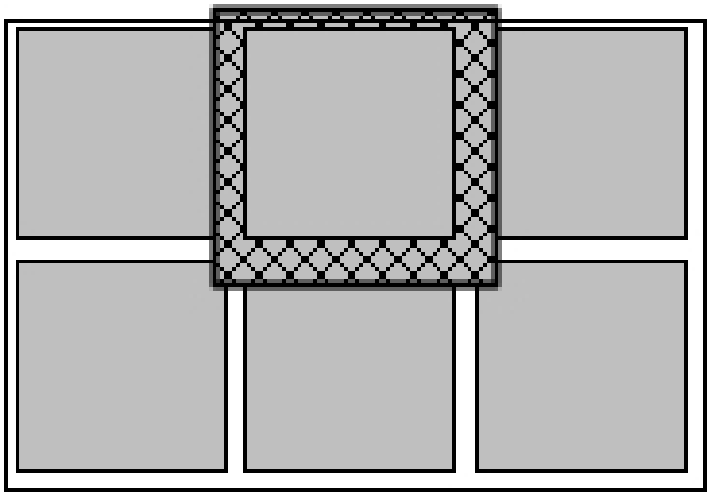
Combine:

Sparse matrix-vector product

Difference stencil



induces ghost region:



Limited number of neighbours, limited buffer space

Scaling

Separately 1D and 2D partitioning of the domain.

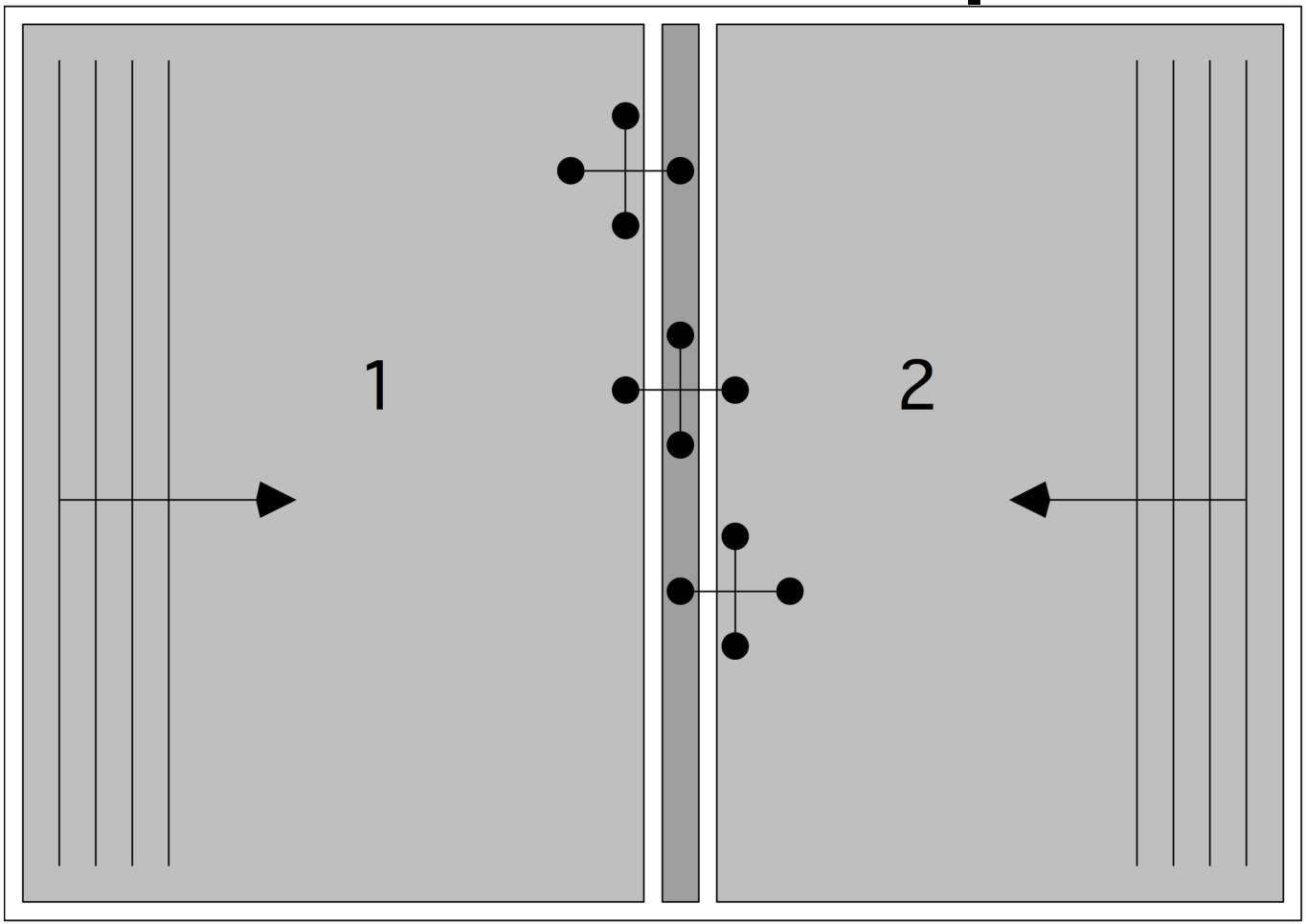
MPI Implementation

Gaussian elimination

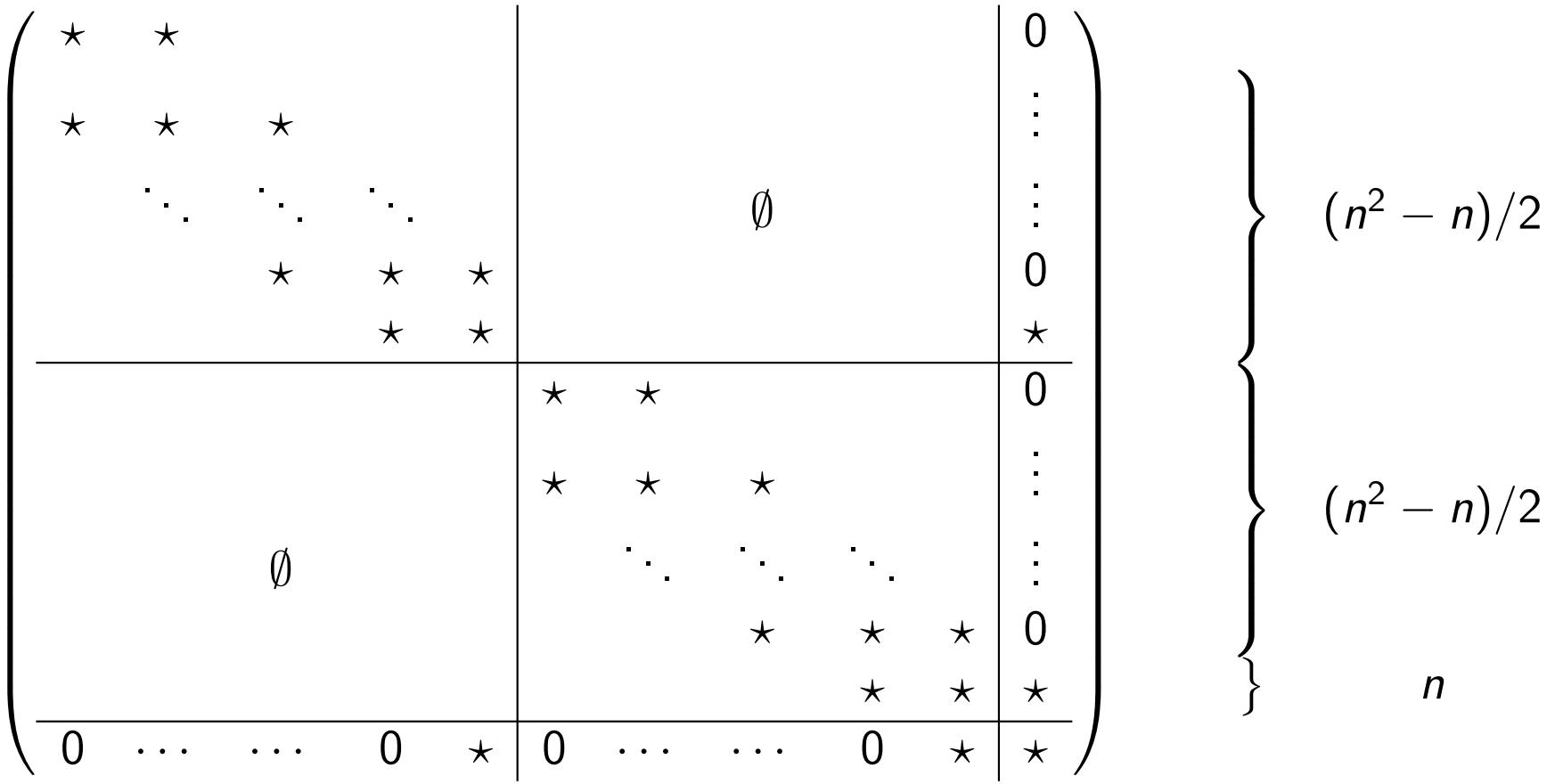
Differences between product and solving

- Treatment of sparsity
- Operation count
- Ease of parallelizing

Domain decomposition



3


$(n^2-n)/2$
$(n^2-n)/2$
n

DD factorization

$$A^{\mathrm{DD}} = \begin{pmatrix} A_{11} & \emptyset & A_{13} \\ \emptyset & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} =$$

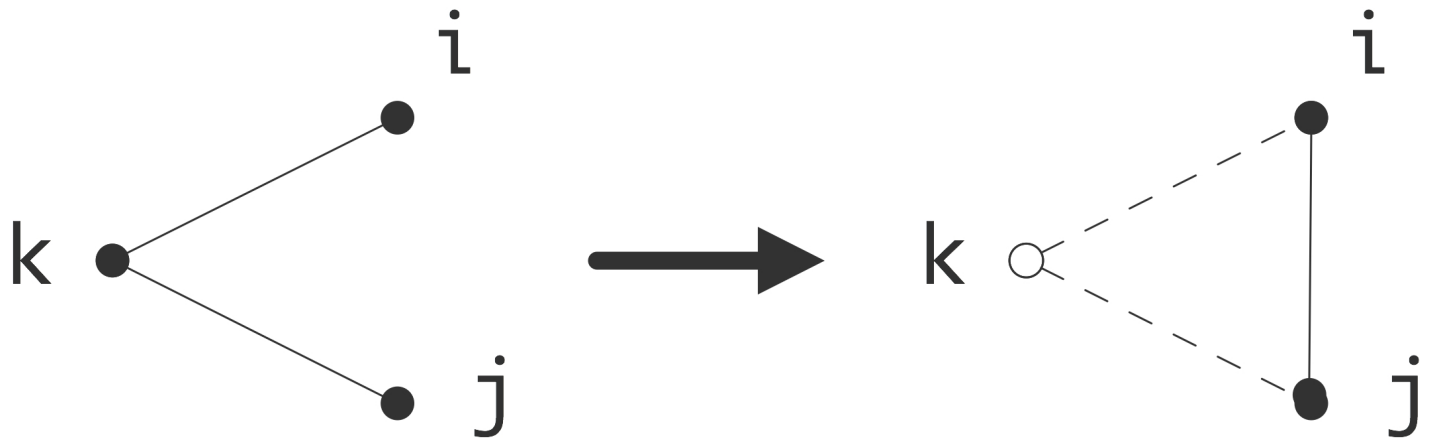
$$\begin{pmatrix} I & & \\ \emptyset & I & \\ A_{31}A_{11}^{-1} & A_{32}A_{22}^{-1} & I \end{pmatrix} \begin{pmatrix} A_{11} & \emptyset & A_{13} \\ & A_{22} & A_{23} \\ & & S \end{pmatrix}$$

$$S = A_{33} - A_{31}A_{11}^{-1}A_{13} - A_{32}A_{22}^{-1}A_{23}$$

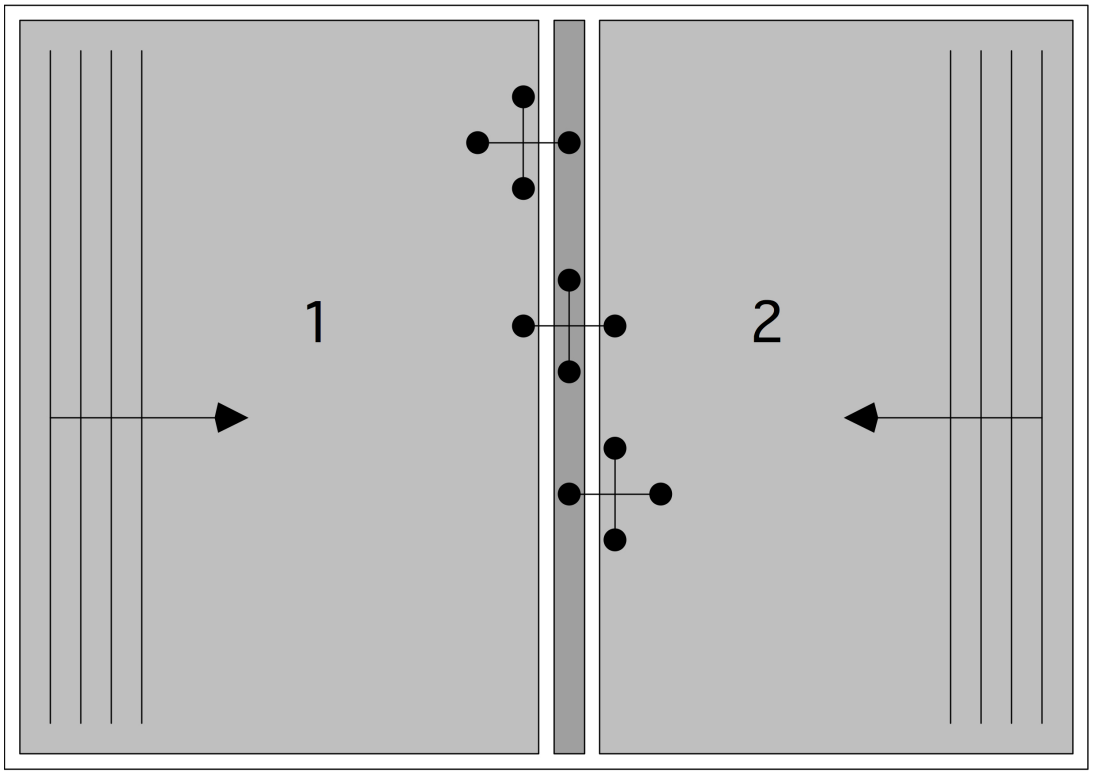
Parallelism. . .

Graph theory of sparse elimination

$$a_{ij} \leftarrow a_{ij} - a_{ik} a_{kk}^{-1} a_{kj}$$

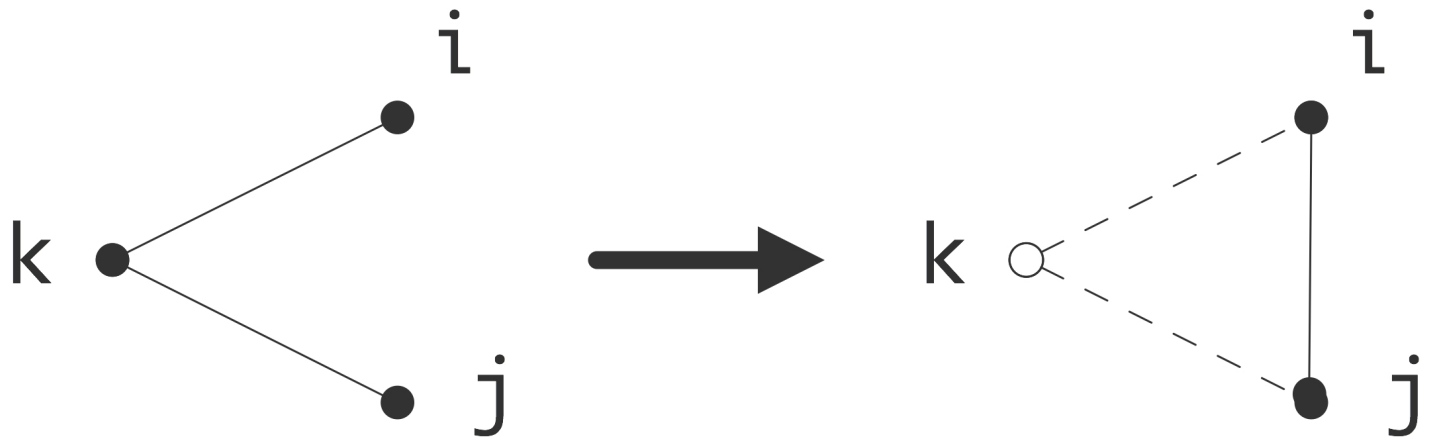


Graph theory of sparse elimination



Graph theory of sparse elimination

$$a_{ij} \leftarrow a_{ij} - a_{ik} a_{kk}^{-1} a_{kj}$$



So inductively S is dense

Recursive bisection

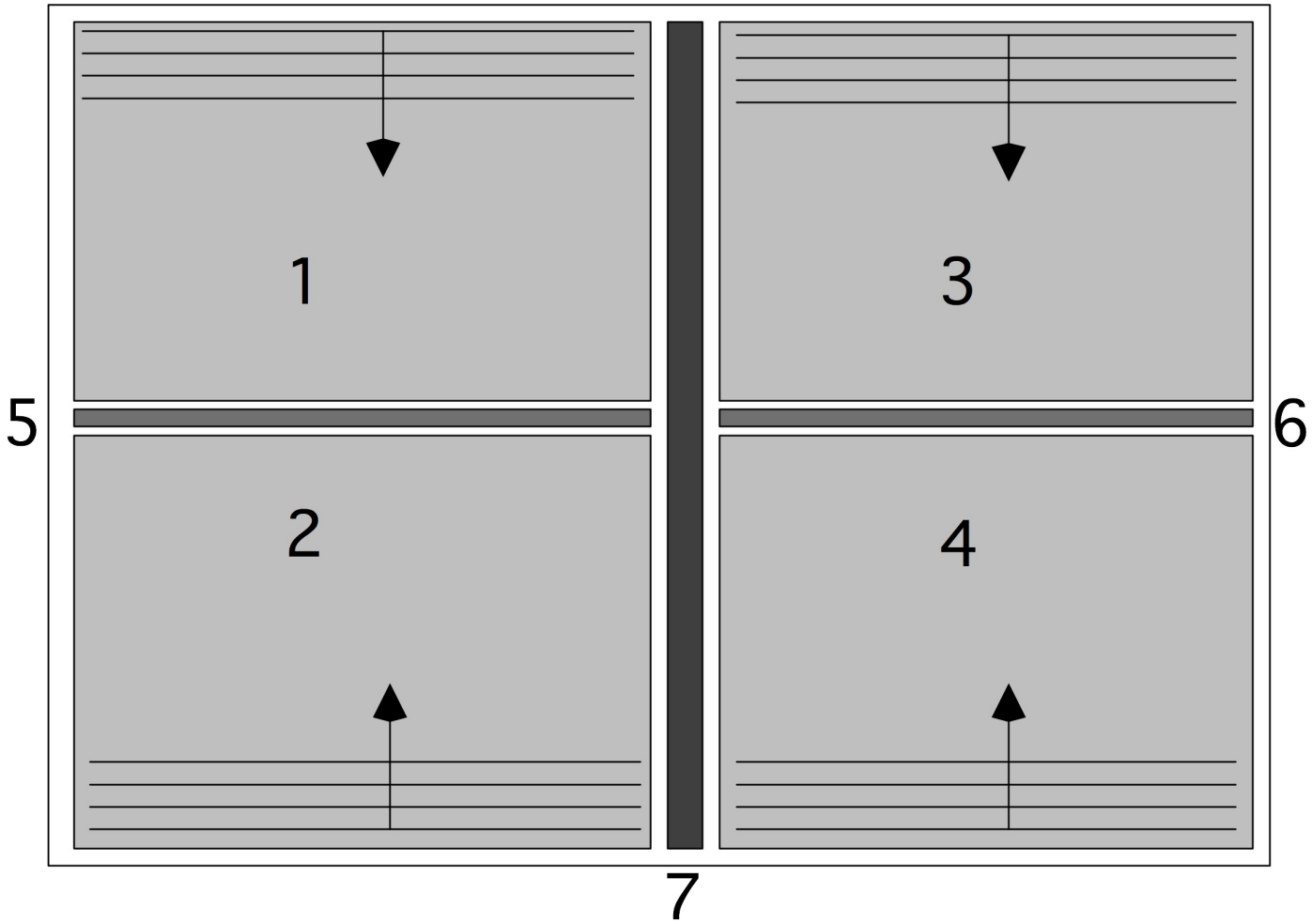
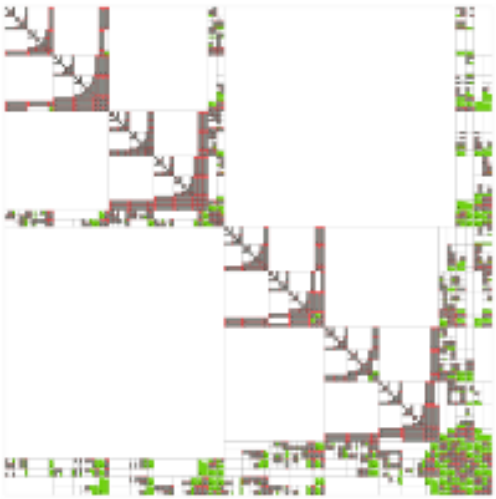


Figure : A four-way domain decomposition

$$A^{\mathrm{DD}} = \begin{pmatrix} A_{11} & & & & A_{15} & & A_{17} \\ & A_{22} & & & A_{25} & & A_{27} \\ & & A_{33} & & & A_{36} & A_{37} \\ & & & A_{44} & & A_{46} & A_{47} \\ A_{51} & A_{52} & & & A_{55} & & A_{57} \\ & & A_{63} & A_{64} & & A_{66} & A_{67} \\ A_{71} & A_{72} & A_{73} & A_{74} & A_{75} & A_{76} & A_{77} \end{pmatrix}$$

The domain/operator/graph view is more insightful, don't you think?

How does this look in reality?



Fill-in is a function of ordering

$$\begin{pmatrix} * & * & \cdots & * \\ * & * & & \emptyset \\ \vdots & & \ddots & \\ * & \emptyset & & * \end{pmatrix}$$

After factorization the matrix is dense.
Can this be permuted?

Fill-in during LU

Fill-in: index (i, j) where $a_{ij} = 0$ but $\ell_{ij} \neq 0$ or $u_{ij} \neq 0$.

2D BVP: Ω is $n \times n$, gives matrix of size $N = n^2$, with bandwidth n.

Matrix storage $O(N)$

LU storage $O(N^{3/2})$

LU factorization work $O(N^2)$

Without proof: nested dissection $O(N \log N)$ space, $O(N^{3/2})$ work

Parallel preconditioners

Sparse operations in parallel: mvp

Mvp $y = Ax$

```
for i=1..n
  y[i] = sum over j=1..n a[i,j]*x[j]
```

In parallel:

```
for i=myfirstrow..mylastrow
  y[i] = sum over j=1..n a[i,j]*x[j]
```

How about ILU solve?

Consider $Lx = y$

```
for i=1..n
  x[i] = (y[i] - sum over j=1..i-1 ell[i,j]*x[j])
         / a[i,i]
```

Parallel code:

```
for i=myfirstrow..mylastrow
  x[i] = (y[i] - sum over j=1..i-1 ell[i,j]*x[j])
         / a[i,i]
```

Problems?

Block method

```
for i=myfirstrow..mylastrow
  x[i] = (y[i] - sum over j=myfirstrow..i-1 ell[i,j]*x[j]
         / a[i,i]
```

Block Jacobi with local GS solve

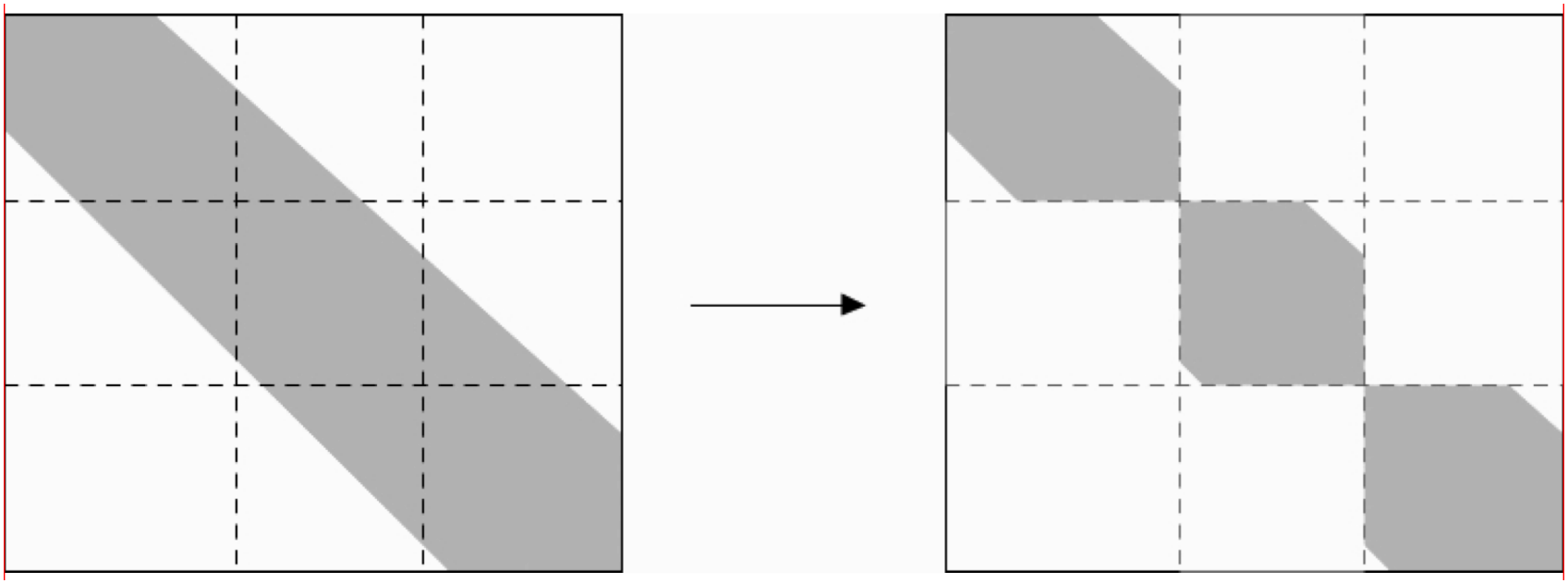


Figure : Sparsity pattern corresponding to a block Jacobi preconditioner

Multicolour ILU