A simple FEM solver and its data parallelism

Gundolf Haase

Institute for Mathematics and Scientific Computing
University of Graz, Austria

Graz, Nov 2021

Partial differential equation

Considered Problem Classes

$$\boxed{\begin{array}{lrll} \text{Find } u \text{ such that} & Lu(x) = & f(x) & \forall x \in \Omega \\ & lu(x) = & g(x) & \forall x \in \partial\Omega \end{array}}$$

variational $\Downarrow$ formulation

$$\boxed{\text{Find} \quad u \in \mathbb{V}: \qquad a(u,v) = \langle F, v \rangle \qquad \forall v \in \mathbb{V}}$$

FEM, FDM $\Downarrow$ FVM, FIT

$$\boxed{\text{Solve} \qquad K_h \cdot \underline{u}_h = \underline{f}_h \qquad \underline{u}_h \in \mathbb{R}^{N_h}}$$

- (linear) 2nd order problem.
  - Poisson equation (temperature)
  - Lamé equation (deformation)
  - Maxwell's equations (magnetic field)
- Matrix K_h is sparse, positive definite
  (symmetric, large dimension)
- non-linear and time-dependent problems.

Second order PDE

Find $u \in X := C^2(\Omega) \cap C^1(\Omega \cup \Gamma_2 \cup \Gamma_3) \cap C(\Omega \cup \Gamma_1)$ such that the partial differential equation

$$-\sum_{i,j=1}^{m} \frac{\partial}{\partial x_i}\left(a_{ij}(x)\,\frac{\partial u}{\partial x_j}\right) + \sum_{i=1}^{m} a_i(x)\,\frac{\partial u}{\partial x_i} + a(x)u(x) = f(x) \qquad (1)$$

holds for all $x \in \Omega$ and that the Boundary Conditions (BC)

- $u(x) = g_1(x),\ \forall x \in \Gamma_1$ (Dirichlet (1st-kind) BC),
- $\frac{\partial u}{\partial N} := \sum_{i,j=1}^{m} a_{ij}(x)\frac{\partial u(x)}{\partial x_j}\, n_i(x) = g_2(x),\ \forall x \in \Gamma_2$ (Neumann (2nd-kind) BC),
- $\frac{\partial u}{\partial N} + \alpha(x)u(x) = g_3(x),\ \forall x \in \Gamma_3$ (Robin (3rd-kind) BC).

are satisfied.

with $u(x)$ as classical continuous solution of the PDE.

Variational formulation

Choose the space of test functions $V_0 = \{v \in V = H^1(\Omega) : v = 0 \text{ on } \Gamma_1\}$, where $V = H^1(\Omega)$ is the basic space

$$\boxed{\begin{aligned}
&\text{Find } u \in V_g \text{ such that } a(u,v) = \langle F, v\rangle \quad \forall\, v \in V_0, \text{ where}\\
&a(u,v) := \int_\Omega \left(\sum_{i,j=1}^m a_{ij} \frac{\partial u}{\partial x_j} \frac{\partial v}{\partial x_i} + \sum_{i=1}^m a_i \frac{\partial u}{\partial x_i} v + auv \right) dx + \int_{\Gamma_3} \alpha uv \, ds,\\
&\langle F, v\rangle := \int_\Omega fv \, dx + \int_{\Gamma_2} g_2 v \, ds + \int_{\Gamma_3} g_3 v \, ds,\\
&V_g := \{v \in V = H^1(\Omega) : v = g_1 \text{ on } \Gamma_1\},\\
&V_0 := \{v \in V : v = 0 \text{ on } \Gamma_1\}.
\end{aligned}} \tag{2}$$

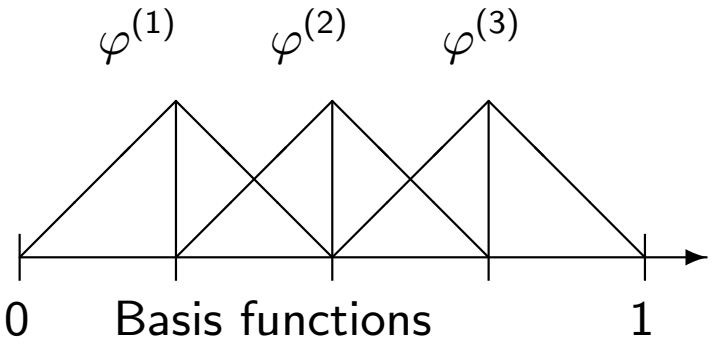
with $u(x)$ as weak continuous solution of the PDE.

Finite Elements

Continuous solution $u(x) \longrightarrow$ discrete solution $\underline{u}_h$ from the finite dimensional space

$$V_h = \text{span}\left\{\varphi^{(i)} : i \in \overline{\omega}_h\right\} = \left\{v_h = \sum_{i \in \overline{\omega}_h} v^{(i)}\varphi^{(i)}\right\} = \text{span}\ \overline{\Phi} \subset V \tag{3}$$

spanned by the (linear independent) basis functions $\overline{\Phi} = [\varphi^{(i)} : i \in \overline{\omega}_h] = [\varphi_1, \ldots, \varphi_{\overline{N}_h}]$ with $\overline{\omega}_h$ as indices of basis functions. 1D linear basis functions with finite support on the neighboring elements are presented in the following picture:



Our example: Laplace equation

$$\boxed{\begin{array}{lrll} \text{Find } u \text{ such that} & -\Delta u(x) = & f(x) & \forall x \in \Omega = [0,1]^2 \\ & u(x) = & 0 & \forall x \in \partial\Omega \end{array}}$$

variational $\Downarrow$ formulation

$$\boxed{\begin{array}{lrl} \text{Find} \quad u \in \mathbb{V}: & a(u,v) := & \displaystyle\int_\Omega \nabla^T v(x) \cdot \nabla u(x) dx \\ & \langle F, v \rangle := & \displaystyle\int_\Omega f(x) v(x) dx \end{array}}$$

FEM, FDM $\Downarrow$ FVM, FIT

$$\boxed{\text{Solve} \qquad K_h \cdot \underline{u}_h = \underline{f}_h \qquad \underline{u}_h \in \mathbb{R}^{N_h}}$$

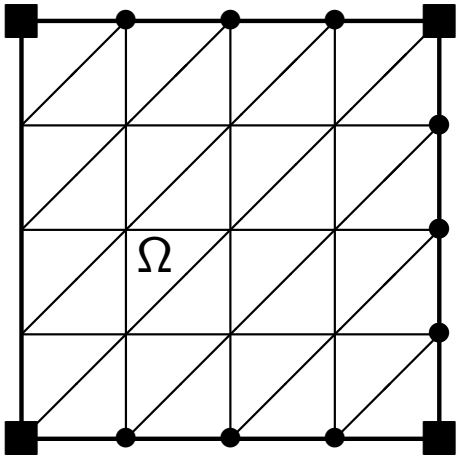
with $K^{ij} := \displaystyle\int_\Omega \nabla^T \varphi_j(x) \cdot \nabla \varphi_i(x) dx = \sum_{\tau_e \in \operatorname{supp} \varphi_i \cap \operatorname{supp} \varphi_j} \int_{\tau_e} \nabla^T \varphi_j(x) \cdot \nabla \varphi_i(x) dx$

How to solve Laplace equation?

1. Generate a finite element mesh.
2. Determine matrix pattern (sparse matrix!) and allocate storage.
3. Calculate Matrix K_h and r.h.s. $\underline{f}_h$ for each element.
   $$\int_{\tau_e} \nabla^T \varphi_j(x) \cdot \nabla \varphi_i(x) dx$$
4. Accumulate the element entries.
   $$\sum_{\tau_e \in \mathrm{supp}\, \varphi_i \cap \mathrm{supp}\, \varphi_j}$$
5. Solve the system of equations $K_h \cdot \underline{u}_h = \underline{f}_h$.

Discretizing the domain $[xl, xr] \times [yb, yt]$

nx=ny=4 intervals



- trangular elements
- linear shape functions
- `GetMesh(nx, ny, xl, xr, yb, yt, nnode, xc, nelem, ia);`
- OUTPUT:
  - `nnode` : number of nodes
  - `xc[2*nnode]` : node coordinates
  - `nelem` : number of finite elements
  - `ia[3*nelem]` : element connectivity (3 node numbers per element)

Storing the sparse matrix

CRS: compressed row storage The matrix

$$K_{n\times m} = \begin{pmatrix} 10 & 0 & 0 & -2 \\ 3 & 9 & 0 & 0 \\ 0 & 7 & 8 & 7 \\ 3 & 0 & 8 & 7 \end{pmatrix}$$

can be stored using just two `integer` vectors and one `real`/`double` vector.

Values : $sk =$

10	−2	3	9	7	8	7	3	8	7

Column index : $ik =$

1	4	1	2	2	3	4	1	3	4

Starting index of row : $id =$

1	3	5	8	11

Dimensions for n rows and nnz non-zero elements in matrix:
`sk[nn], ik[nn], id[n+1]` Note that (in C/C++) `id[n] = nnz`.

also: Compressed Column Storage (CCS), Compressed Diagonal Storage (CDS), Jagged Diagonal Storage (JDS), ELLPACK, . . .

Matrix generation in code

- Determine matrix pattern and allocate memory for CRS

```
Get_Matrix_Pattern(nelem, 3, ia, nnz, id, ik, sk);
```

  - `nnz` : number of non-zereo elements in matrix
  - `id[nnode+1]`, `ik[nnz]` allocated and initialized
  - `sk[nnz]` allocated

- Calculate Matrix entries and accumulate them

```
GetMatrix (nelem, 3, ia, nnode, xc, nnz, id, ik, sk, f);
```

  - `sk[nnz]` matrix values initialized
  - `f[nnode]` r.h.s. initialized

- Apply Dirichlet boundary conditions

```
ApplyDirichletBC(nx, ny, neigh, u, id, ik, sk, f);
```

  - `sk[nnz]` matrix values adapted to B.C.
  - `f[nnode]` r.h.s. adapted to B.C.
  - `nx, ny` represent the geometry a input
  - `neigh` represents neighboring domains in parallel context

Solve the system of equations via Jacobi iteration

We solve $K\underline{u} = f$ by the Jacobi iteration ($\omega = 1$)

$$\underline{u}^{k+1} := \underline{u}^{k+1} + \omega D^{-1}\left(\underline{f} - K \cdot \underline{u}^{k}\right)$$

```
JacobiSolve(nnode, id, ik, sk, f, u );
```

until the relative error in the $KD^{-1}K$-norm is smaller than $\varepsilon = 10^{-5}$.

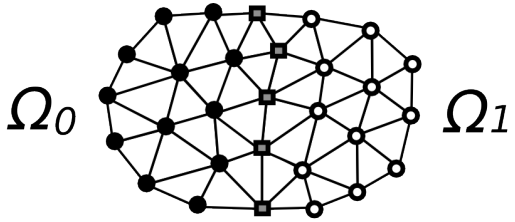
$$
\begin{array}{lll}
D & := & \mathrm{diag}(K) \\
\underline{u} & := & 0 \\
\underline{r} & := & \underline{f} - K \cdot \underline{u}^0 \\
\underline{w} & := & D^{-1} \cdot \underline{r} \\
\sigma & := & \sigma_0 \;:=\; (\underline{w}, \underline{r}) \\
k & := & 0 \\
\text{while} & \sigma > \varepsilon^2 \cdot \sigma_0 & \text{do} \\
 & k & := & k+1 \\
 & \underline{u}^k & := & \underline{u}^{k-1} + \omega \cdot \underline{w} & \text{// vector arithmetics} \\
 & \underline{r} & := & \underline{f} - K \cdot \underline{u}^k & \text{// sparse matrix-times-vector + vector arithmetics} \\
 & \underline{w} & := & D^{-1} \cdot \underline{r} & \text{// vector arithmetics} \\
 & \sigma & := & (\underline{w}, \underline{r}) & \text{// inner product} \\
\text{end} & & & &
\end{array}
$$

Data Parallelism for distributed memory

Decomposing the mesh

The f.e. mesh is partitioned into P non-overlapping subdomains. (METIS,PARMETIS; SCOTCH, PT-SCOTCH)

Unique mapping of an element to exacly one subdomain.

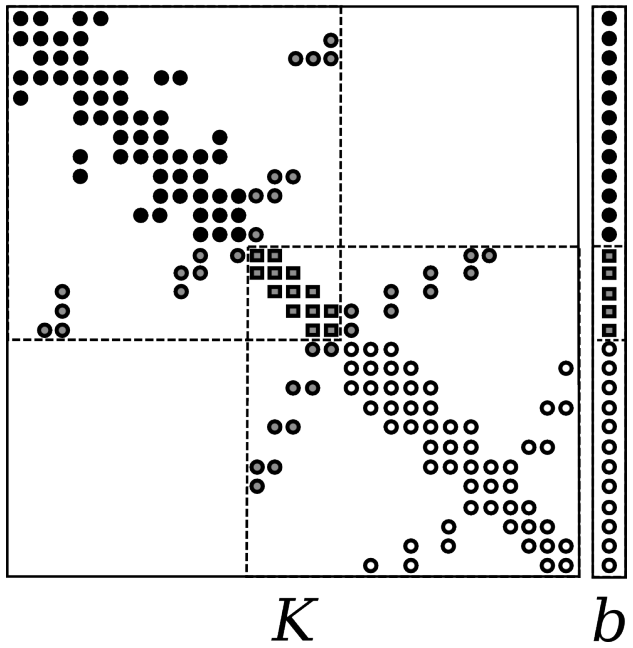


Decompose linear system

$$K^{ij} = \sum_{\tau_h} \int_{\tau_h} \nabla\varphi_i \cdot \nabla\varphi_j$$

into two subsystems K_0 and K_1:

1. Non-overlapping decomposition of finite elements.
2. Overlapping nodes on boundary between subdomains.



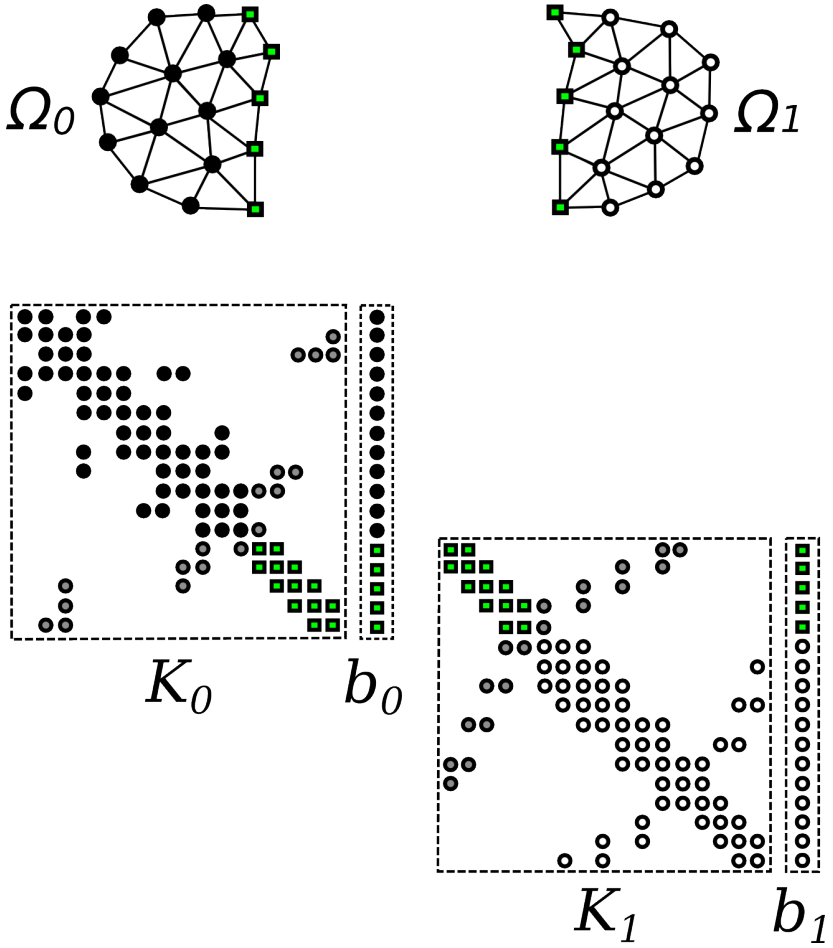
Decomposition of matrix I

Local system

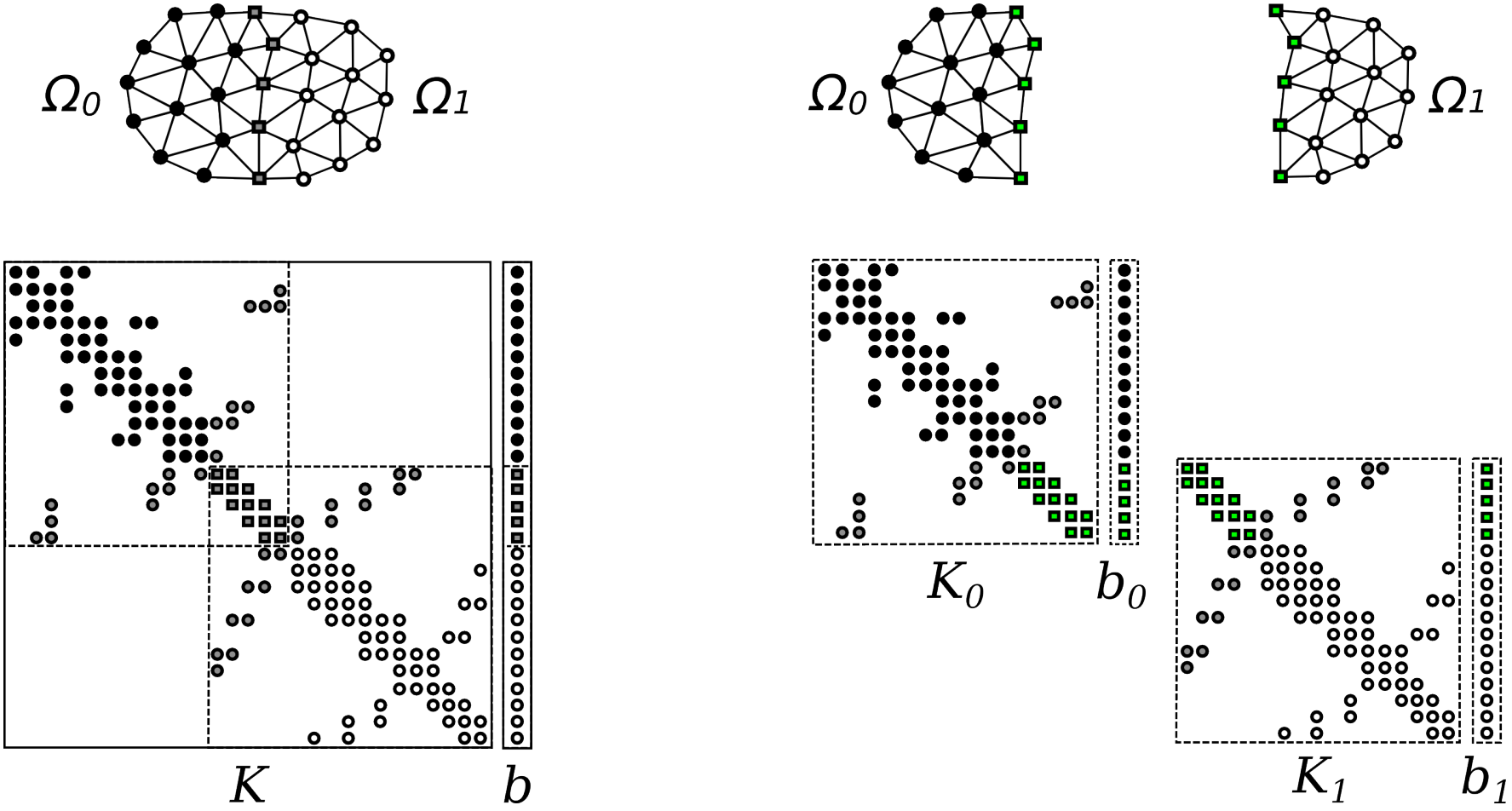
$$K_s^{ij} = \sum_{\tau_h \cap \Omega_s} \int_{\tau_h} \nabla \varphi_i \cdot \nabla \varphi_j$$

assembled locally:

- Distribute geometry
- Compute local stiffness matrix
- Assemble local *distributed* equation system.

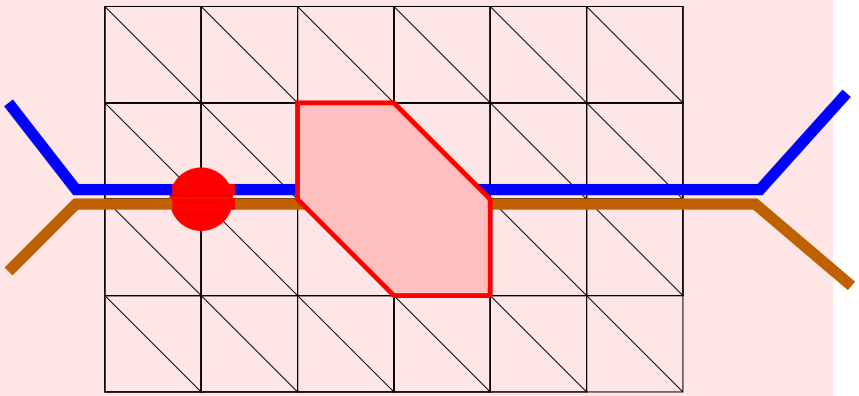


Decomposition of matrix II



Data representations

accumulated

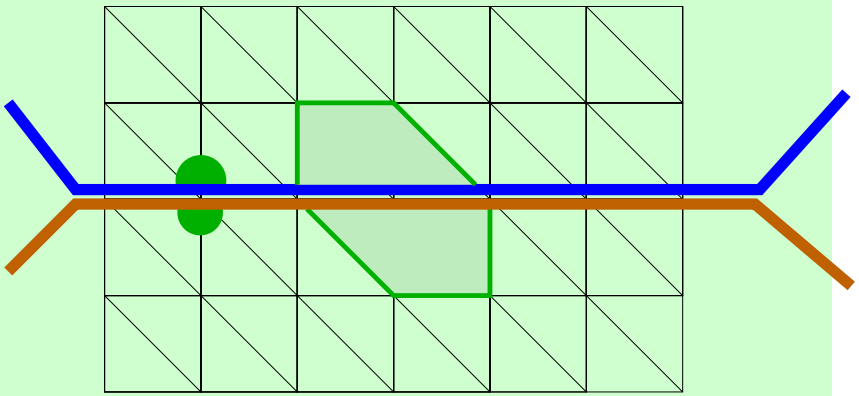


$$\underline{\mathfrak{u}}_s = A_s \underline{\mathfrak{u}}$$

$$\mathfrak{K}_s = A_s \mathfrak{K} A_s^T$$

$$\mathfrak{K}^{ij} = \sum_{\tau_h} \int_{\tau_h} \nabla \varphi_i \cdot \nabla \varphi_j$$

distributed



$$\underline{r} = \sum_{s=1}^{P} A_s^T \underline{r}_s$$

$$K = \sum_{s=1}^{P} A_s^T K_s A_s$$

$$K_s^{ij} = \sum_{\tau_h \cap \Omega_s} \int_{\tau_h} \nabla \varphi_i \cdot \nabla \varphi_j$$

Parallel Linear Algebra

Global-to-local map

$$A_i = \begin{bmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ & & & \end{bmatrix}$$

Scalar product

$$\langle \mathfrak{w}, \mathsf{r} \rangle = \mathfrak{w}^T \cdot \mathsf{r} = \mathfrak{w}^T \cdot \sum_{i=1}^{P} A_i^T \mathsf{r}_i = \sum_{i=1}^{P} (A_i \mathfrak{w})^T \mathsf{r}_i = \sum_{i=1}^{P} \langle \mathfrak{w}_i, \mathsf{r}_i \rangle$$

Matrix-vector product

$$\mathsf{f} := \sum_{i=1}^{P} A_i^T \mathsf{f}_i = \sum_{i=1}^{P} A_i^T \mathsf{K}_i \mathfrak{u}_i = \sum_{i=1}^{P} A_i^T \mathsf{K}_i A_i \mathfrak{u} = \mathsf{K} \cdot \mathfrak{u}$$

Jacobi iteration

$$\mathfrak{u} := \mathfrak{u} + \omega \mathfrak{D}^{-1} \sum_{k=1}^{P} A_k^T \left(\mathsf{f}_k - \mathsf{K}_k \mathfrak{u}_k \right)$$

Parallel Linear Algebra

no communication	global communication	next neighbor comm.
$\underline{v} \leftarrow K \cdot \underline{\mathfrak{s}}$	$\langle \mathfrak{w}, r \rangle = \sum_{s=1}^{P} \langle \mathfrak{w}_s, r_s \rangle$	$\underline{\mathfrak{r}}_s \leftarrow A_s \sum_{k=1}^{P} A_k^T \underline{r}_k$
$\underline{r} \leftarrow \underline{f} + \alpha \cdot \underline{v}$		
$\underline{\mathfrak{w}} \leftarrow \underline{\mathfrak{u}} + \alpha \cdot \underline{\mathfrak{s}}$		$\mathfrak{K}_s \leftarrow A_s (\sum_{k=1}^{P} A_k^T K_k A_k) A_s^T$
$\underline{r} \leftarrow R^{-1} \cdot \underline{\mathfrak{w}}$		

$$R = diag\{R_{ii}\}_{i=1}^{N} = \sum_{s=1}^{P} A_s \cdot A_s^T$$

and $R^{-1} \equiv I = \sum_{s=1}^{P} A_s I_s A_s^T$ (partition of unity)

Our example: Domain Decomposition

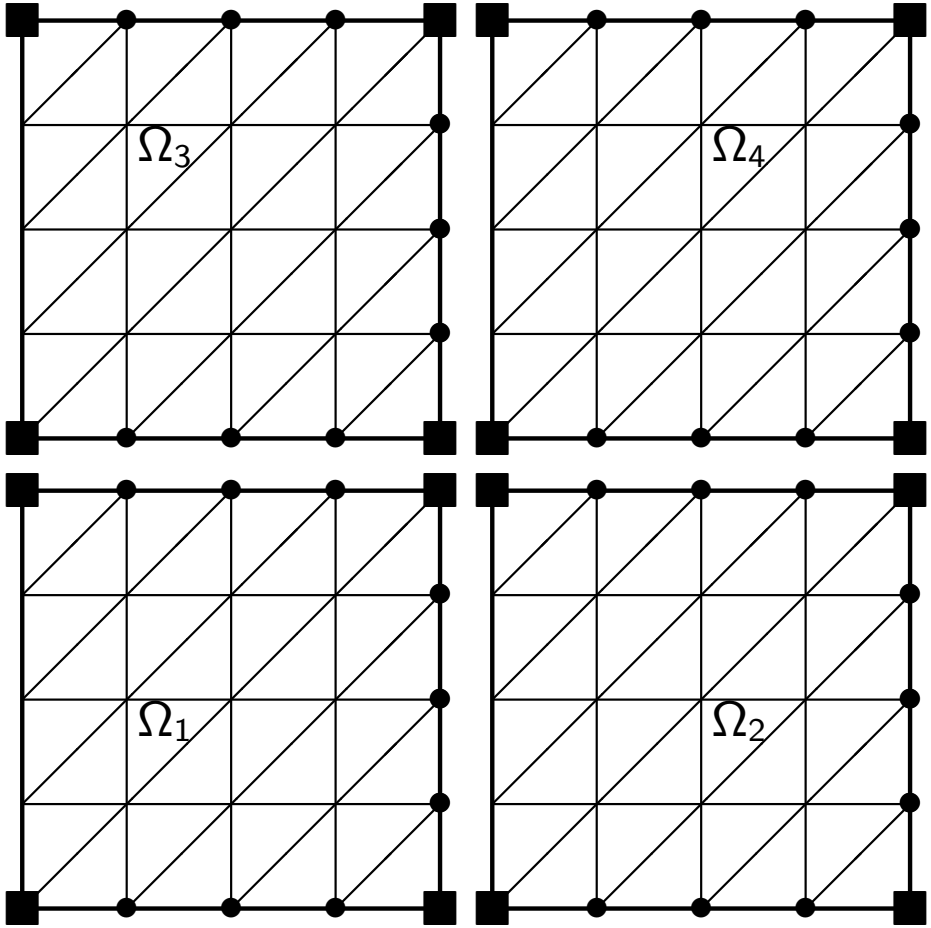


Figure: Non-overlapping elements.

Parallel matrix generation

- Each process s posesses the elements of Ω_s.

```
GetMesh(nx, ny, xl, xr, yb, yt, nnode, xc, nelem, ia);
```

  with individual `xl, xr, yb, yt` in our example
- The local (distributed) matrix

$$K_s^{ij} := \sum_{\tau_h \cap \Omega_s} \int_{\tau_h} \nabla \varphi_i \cdot \nabla \varphi_j$$

  is calculated by using directly the sequential routines

```
Get_Matrix_Pattern(nelem, 3, ia, nnz, id, ik, sk);
GetMatrix (nelem, 3, ia, nnode, xc, nnz, id, ik, sk, f);
ApplyDirichletBC(nx, ny, neigh, u, id, ik, sk, f);
```

Parallel Jacobi iteration for decomposed domain

We solve $K\underline{u} = f$ by the Jacobi iteration ($\omega = 1$)

$$\underline{u}^{k+1} := \underline{u}^{k+1} + \omega D^{-1}\left(\underline{f} - K \cdot \underline{u}^k\right)$$

on P processes with distributed data.

`JacobiSolve(nnode, id, ik, sk, f, u );`

$$\begin{array}{lll}
\mathfrak{D} & := & \sum_{s=1}^{P} A_s^T \operatorname{diag}(\mathrm{K}_s) A_s \qquad \text{// next neighbor comm. of a vector} \\
\underline{\mathfrak{u}} & := & 0 \\
\underline{\mathrm{r}} & := & \underline{\mathrm{f}} - \mathrm{K} \cdot \underline{\mathfrak{u}}^0 \\
\underline{\mathfrak{w}} & := & \mathfrak{D}^{-1} \cdot \sum_{s=1}^{P} A_s^T \underline{\mathrm{r}}_s \qquad \text{// next neighbor comm.} \\
\sigma & := & \sigma_0 \;:=\; (\underline{\mathfrak{w}}, \underline{\mathrm{r}}) \qquad \text{// parallel reduction} \\
k & := & 0 \\
\text{while} & \sigma > \varepsilon^2 \cdot \sigma_0 & \text{do} \\
 & k & := \; k+1 \\
 & \underline{\mathfrak{u}}^k & := \; \underline{\mathfrak{u}}^{k-1} + \omega \cdot \underline{\mathfrak{w}} \qquad \text{// no comm.} \\
 & \underline{\mathrm{r}} & := \; \underline{\mathrm{f}} - \mathrm{K} \cdot \underline{\mathfrak{u}}^k \qquad \text{// no comm.} \\
 & \underline{\mathfrak{w}} & := \; \mathfrak{D}^{-1} \cdot \sum_{s=1}^{P} A_s^T \underline{\mathrm{r}}_s \qquad \text{// next neighbor comm.} \\
 & \sigma & := \; (\underline{\mathfrak{w}}, \underline{\mathrm{r}}) \qquad \text{// parallel reduction} \\
\text{end} & &
\end{array}$$