

A simple FEM solver and its data parallelism

Gundolf Haase

Institute for Mathematics and Scientific Computing
University of Graz, Austria

Graz, Oct 2025



Partial differential equation

Considered Problem Classes

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

variational \Downarrow formulation

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

FEM, FDM \Downarrow FVM, FIT

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

▶ (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

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

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

Find $u \in V_g$ such that $a(u, v) = \langle F, v \rangle \quad \forall v \in V_0$, where

$$a(u, v) := \int_{\Omega} \left(\sum_{i,j=1}^m a_{ij} \frac{\partial u}{\partial x_j} \frac{\partial v}{\partial x_i} + \sum_{i=1}^m a_i \frac{\partial u}{\partial x_i} v + auv \right) dx + \int_{\Gamma_3} \alpha uv ds,$$
$$\langle F, v \rangle := \int_{\Omega} fv dx + \int_{\Gamma_2} g_2 v ds + \int_{\Gamma_3} g_3 v ds, \quad (2)$$

$$V_g := \{v \in V = H^1(\Omega) : v = g_1 \text{ on } \Gamma_1\},$$

$$V_0 := \{v \in V : v = 0 \text{ on } \Gamma_1\}.$$

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

Finite Elements

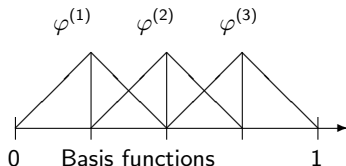
Continuous solution $u(x) \rightarrow$ discrete solution \underline{u}_h from the finite dimensional space

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

spanned by the (linear independent) basis functions

$\bar{\Phi} = [\varphi^{(i)} : i \in \bar{\omega}_h] = [\varphi_1, \dots, \varphi_{N_h}]$ with $\bar{\omega}_h$ 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

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

variational \Downarrow formulation

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

FEM, FDM \Downarrow FVM, FIT

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

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

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.

$$\int_{\tau_e} \nabla^T \varphi_j(x) \cdot \nabla \varphi_i(x) dx$$

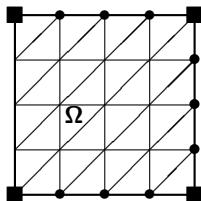
4. Accumulate the element entries.

$$\sum_{\tau_e \in \text{supp } \varphi_i \cap \text{supp } \varphi_j}$$

5. Solve the system of equations $K_h \cdot \underline{u}_h = \underline{f}_h$.

Discretizing the domain $[x_l, x_r] \times [y_b, y_t]$

$n_x = n_y = 4$ intervals



- ▶ triangular 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-zero elements in matrix
- ▶ `id[nnz+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 $K\underline{u} = \underline{f}$ by the Jacobi iteration ($\omega = 1$)

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

JacobiSolve(mnode, id, ik, sk, f, u);

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

$D := \text{diag}(K)$

$\underline{u} := \underline{0}$

$\underline{r} := \underline{f} - K \cdot \underline{u}^0$

$\underline{w} := D^{-1} \cdot \underline{r}$

$\sigma := \sigma_0 := (\underline{w}, \underline{r})$

$k := 0$

while $\sigma > \varepsilon^2 \cdot \sigma_0$ do

$k := k + 1$

$\underline{u}^k := \underline{u}^{k-1} + \omega \cdot \underline{w}$

// vector arithmetics

$\underline{r} := \underline{f} - K \cdot \underline{u}^k$

// sparse matrix-times-vector + vector arithmetics

$\underline{w} := D^{-1} \cdot \underline{r}$

// vector arithmetics

$\sigma := (\underline{w}, \underline{r})$

// inner product

end

Data Parallelism for distributed memory

Decomposing the mesh

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

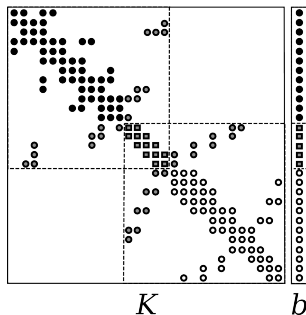
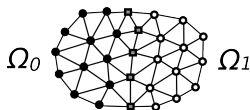
Unique mapping of an element to exactly one subdomain.

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.



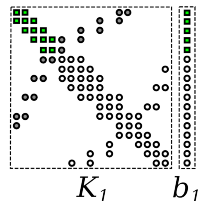
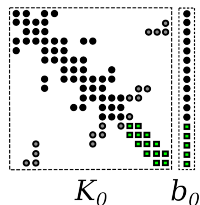
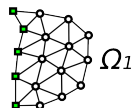
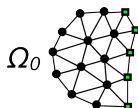
Decomposition of matrix I

Local system

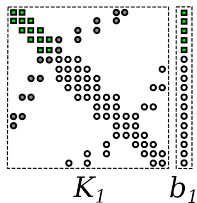
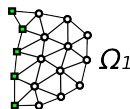
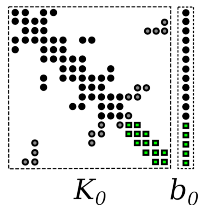
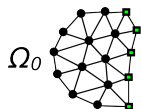
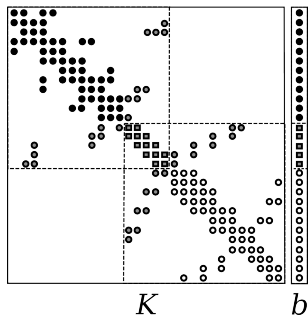
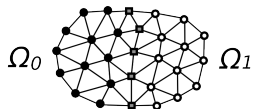
$$K_s^{ij} = \sum_{\tau_h \cap \Omega_s} \int_{\tau_h} \nabla \varphi_i \cdot \nabla \varphi_j$$

assembled locally:

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

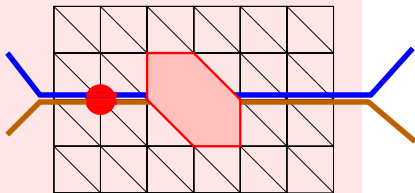


Decomposition of matrix II



Data representations

accumulated

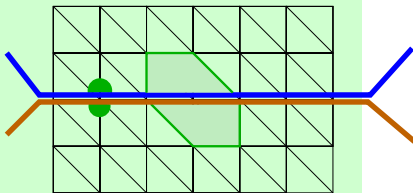


$$\underline{u}_s = A_s \underline{u}$$

$$\mathcal{R}_s = A_s \mathcal{R} A_s^T$$

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

distributed



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

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

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

Parallel Linear Algebra

Global-to-local map

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

Scalar product

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

Matrix-vector product

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

Jacobi iteration

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

Parallel Linear Algebra

no communication

$$\begin{aligned}\underline{v} &\leftarrow \mathbf{K} \cdot \underline{s} \\ \underline{r} &\leftarrow \underline{f} + \alpha \cdot \underline{v} \\ \underline{w} &\leftarrow \underline{u} + \alpha \cdot \underline{s} \\ \underline{r} &\leftarrow \mathbf{R}^{-1} \cdot \underline{w}\end{aligned}$$

global communication

$$\langle \underline{w}, \underline{r} \rangle = \sum_{s=1}^P \langle \underline{w}_s, \underline{r}_s \rangle$$

next neighbor comm.

$$\begin{aligned}\underline{r}_s &\leftarrow \mathbf{A}_s \sum_{k=1}^P \mathbf{A}_k^T \underline{r}_k \\ \underline{R}_s &\leftarrow \mathbf{A}_s \left(\sum_{k=1}^P \mathbf{A}_k^T \mathbf{K}_k \mathbf{A}_k \right) \mathbf{A}_s^T\end{aligned}$$

$$\mathbf{R} = \text{diag}\{\mathbf{R}_{ii}\}_{i=1}^N = \sum_{s=1}^P \mathbf{A}_s \cdot \mathbf{A}_s^T$$

$$\text{and } \mathbf{R} \equiv \mathbf{I} = \sum_{s=1}^P \mathbf{A}_s \mathbf{I}_s \mathbf{A}_s^T \quad (\mathbf{R}^{-1} \text{ partition of unity})$$

Our example: Domain Decomposition

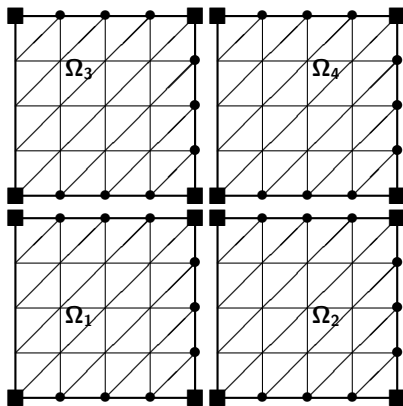


Figure: Non-overlapping elements.

Parallel matrix generation

- ▶ Each process s possesses the elements of Ω_s .

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

with individual x_l , x_r , y_b , y_t 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 $K\underline{u} = \underline{f}$ by the Jacobi iteration ($\omega = 1$)

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

on P processes with distributed data.

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

```
 $\mathcal{D} := \sum_{s=1}^P A_s^T \text{diag}(K_s) A_s$  // next neighbor comm. of a vector
```

```
 $\underline{u} := 0$ 
```

```
 $\underline{r} := \underline{f} - K \cdot \underline{u}^0$ 
```

```
 $\underline{w} := \mathcal{D}^{-1} \cdot \sum_{s=1}^P A_s^T \underline{r}_s$  // next neighbor comm.
```

```
 $\sigma := \sigma_0 := (\underline{w}, \underline{r})$  // parallel reduction
```

```
 $k := 0$ 
```

```
while  $\sigma > \varepsilon^2 \cdot \sigma_0$  do
```

```
     $k := k + 1$ 
```

```
     $\underline{u}^k := \underline{u}^{k-1} + \omega \cdot \underline{w}$  // no comm.
```

```
     $\underline{r} := \underline{f} - K \cdot \underline{u}^k$  // no comm.
```

```
     $\underline{w} := \mathcal{D}^{-1} \cdot \sum_{s=1}^P A_s^T \underline{r}_s$  // next neighbor comm.
```

```
     $\sigma := (\underline{w}, \underline{r})$  // parallel reduction
```

```
end
```