Parallel Scientific Computing

Course AMS301 — Fall 2023 — Lecture 7

Algebraic systems resulting from finite element discretizations

Axel Modave

Algebraic systems resulting from finite element discretizations . . .

Problems with more complicated algorithmic structures
Writing/implementing these algorithms requires graph manipulation
Parallel implementation more difficult

Problem considered for this session

Let | an open bounded *domain* $\Omega \subset \mathbb{R}^d$ with a sufficiently regular boundary $\partial\Omega$
| *data* $f \in L^2(\Omega)$ and $g \in L^2(\partial\Omega)$

Find $u \in H^1(\Omega)$ such that $\begin{cases} -\Delta u + u = f & \text{in } \Omega \\ \partial_n u_{|\partial\Omega} = g & \text{on } \partial\Omega \end{cases}$

After discretization with finite elements . . .

Find $\mathbf{x} \in \mathbb{R}^N$ such that $\boxed{\mathbf{A}\mathbf{x} = \mathbf{f}}$

with $\mathbf{A} \in \mathbb{R}^{N \times N}$ and $\mathbf{f} \in \mathbb{R}^N$.

How can we improve the parallel implementation of this problem by using the (sparse) structure of $\mathbf{A}$?

Algebraic systems resulting from finite element discretizations

Recap on finite elements

Sequential implementation

Parallel implementation

Recap on finite elements — Formulation

Exact differential formulation (DF)

Find $u \in H^1(\Omega)$ such that $\begin{cases} -\Delta u + u = f & \text{in } \Omega \\ \partial_n u_{|\partial\Omega} = g & \text{on } \partial\Omega \end{cases}$

$\Updownarrow$ `Equivalent formulations`

Exact variational formulation (VF)

Find $u \in H^1(\Omega)$ such that

$$\int_\Omega \nabla u \cdot \nabla v \, d\Omega + \int_\Omega uv \, d\Omega = \int_\Omega fv \, d\Omega + \int_{\partial\Omega} gv \, d\Omega, \quad \forall v \in H^1(\Omega)$$

$\Downarrow$ `Galerkin approximation:` $V \longrightarrow V_h \subset V$

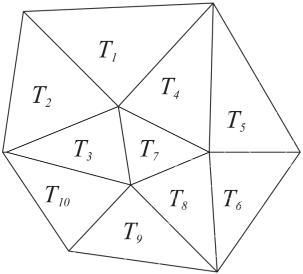
Approximate variational formulation (VF)

Find $u_h \in V_h$ such that

$$\int_\Omega \nabla u_h \cdot \nabla v_h \, d\Omega + \int_\Omega u_h v_h \, d\Omega = \int_\Omega f v_h \, d\Omega + \int_{\partial\Omega} g v_h \, d\Omega, \quad \forall v_h \in V_h$$

Recap on finite elements — Approximation space

For simplicity, we consider 2D cases with polygonal domains.



Mesh $\mathcal{T}_h$

- Set of cells/triangles $\mathcal{T}_h = (T_\ell)_{\ell=1\ldots L}$
- Set of vertices/nodes $\mathcal{M}_h = (M_i)_{i=1\ldots N}$
- Set of edges $\mathcal{E}_h = (E_a)_{a=1\ldots A}$
- h_ℓ = circumcircle diameter of T_ℓ
- $h = \max_\ell h_\ell$ = mesh step of $\mathcal{T}_h$

Properties

- $\overline{\Omega} = \bigcup_{\ell=1\ldots L} T_\ell$
- $T_\ell \cap T_m = \{\varnothing;$ 1 vertex; 1 full edge$\}$ $\longrightarrow$ conformal mesh
- $\mathring{T}_\ell \neq \varnothing$ $\longrightarrow$ no flat triangle

Finite element P_k

For a given mesh $\mathcal{T}_h$, we define:

$$V_h := \left\{ v_h \in C^0(\overline{\Omega}) : v_h|_{T_\ell} \in P_k(T_\ell), \ell = 1 \ldots L \right\} \subset V \quad \textit{(by construction)}$$

where $P_k(T)$ is the space of polynomials of degree $\leq k$.

Recap on finite elements — Theoretical aspects *(for information)*

Find $u \in V$	such that $a(u,v) = b(v)$,	$\forall v \in V$
Find $u_h \in V_h$	such that $a(u_h, v_h) = b(v_h)$,	$\forall v_h \in V_h$

Exact problem

- Equivalence of formulation: u solution of (DF) $\Leftrightarrow$ u solution of (VF)
- Well-posedness: by Lax-Milgram Theorem

Approximate problem

- Well-posedness: by Lax-Milgram Theorem
- Convergence of the numerical solution:

Let $(\mathcal{T}_h)_h$, a regular family of meshes composed of elements P_k.
If $u \in H^{k+1}(\Omega)$ with $k \geq 1$, then there exist constants C_1 and C_2 such that, $\forall h$,

$$\|u - u_h\|_{L^2(\Omega)} \leq C_1\, h^{k+1} |u|_{k+1,\Omega}$$
$$\|u - u_h\|_{H^1(\Omega)} \leq C_2\, h^{k} |u|_{k+1,\Omega}$$

Recap on finite elements — Algebraic system [1/3]

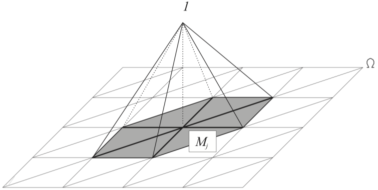
For the sake of simplicity, we consider P_1 finite elements.

Basis functions for V_h

- Lagrange functions $(\phi_i)_{i=1\ldots N}$:

$$\boxed{\phi_i \in V_h \,:\, \phi_i(M_j) = \delta_{ij},\ \forall i, j}$$

Property: $\mathrm{supp}(\phi_i) = \bigcup_{\ell \text{ s.t. } M_i \subset T_\ell} T_\ell$



- The dimension of V_h is N. $\longrightarrow$ Number of vertices/nodes in the mesh
- The functions $(\phi_i)_{i=1\ldots N}$ form a basis of V_h.
- Every solution $v_h \in V_h$ is characterized by the values $(v_h(M_i))_{i=1\ldots N}$:

$$\boxed{v_h(\mathbf{x}) = \sum_{i=1}^{N} v_h(M_i)\, \phi_i(\mathbf{x}), \quad \forall \mathbf{x} \in \Omega}$$

Recap on finite elements — Algebraic system [2/3]

Find $u_h \in V_h$ such that

$$\int_\Omega \nabla u_h \cdot \nabla v_h \, d\Omega + \int_\Omega u_h v_h \, d\Omega = \int_\Omega f v_h \, d\Omega + \int_{\partial\Omega} g v_h \, d\Omega, \quad \forall v_h \in V_h$$

Since $V_h = \mathrm{span}(\phi_1, \ldots, \phi_N)$, it is sufficient to use the basis functions as test functions:

Find $u_h \in V_h$ such that

$$\int_\Omega \nabla u_h \cdot \nabla \phi_i \, d\Omega + \int_\Omega u_h \phi_i \, d\Omega = \int_\Omega f \phi_i \, d\Omega + \int_{\partial\Omega} g \phi_i \, d\Omega, \quad i = 1 \ldots N$$

The approximate solution can be written as

$$u_h(\mathbf{x}) = \sum_{j=1}^{N} \underbrace{u_h(M_j)}_{X_j} \, \phi_j(\mathbf{x})$$

Find $(X_j)_{j=1\ldots N} \in \mathbb{R}^N$ such that

$$\sum_{j=1}^{N} \left[\int_\Omega \nabla \phi_j \cdot \nabla \phi_i \, d\Omega + \int_\Omega \phi_j \phi_i \, d\Omega \right] X_j = \int_\Omega f \phi_i \, d\Omega + \int_{\partial\Omega} g \phi_i \, d\Omega, \quad i = 1 \ldots N$$

Find $\mathbf{x} \in \mathbb{R}^N$ such that $\boxed{\mathbf{A}\mathbf{x} = \mathbf{f}}$

with

$$
\begin{aligned}
A_{ij} &= \int_\Omega \nabla\phi_j \cdot \nabla\phi_i \, d\Omega + \int_\Omega \phi_j \phi_i \, d\Omega && (i,j = 1 \dots N) \\
f_i &= \int_\Omega f\phi_i \, d\Omega + \int_{\partial\Omega} g\phi_i \, d\Omega && (i = 1 \dots N) \\
x_j &= X_j && (j = 1 \dots N)
\end{aligned}
$$

Properties

- $\mathbf{A}$ is symmetric positive definite.

For $\mathbf{y} \in \mathbb{R}^N \backslash \{0\}$:

$$
\begin{aligned}
(\mathbf{A}\mathbf{y}|\mathbf{y}) &= \sum_{i=1}^N \sum_{j=1}^N y_i \, a(\phi_i, \phi_j) \, y_j \\
&= a\left(\sum_{i=1}^N \phi_i y_i, \, \sum_{j=1}^N \phi_j y_j\right) && \textit{(bilinearity of } a\textit{)} \\
&= a(y_h, y_h) \geq \alpha_a \|y_h\|^2 && \textit{(coercivity of } a\textit{)}
\end{aligned}
$$

- $\mathbf{A}$ is (very) sparse.

$$
\left.
\begin{aligned}
&A_{ij} \neq 0 \text{ if } \mathrm{supp}(\phi_i) \cap \mathrm{supp}(\phi_j) \neq \emptyset \\
&\mathrm{supp}(\phi_i) = \bigcup_{\ell \text{ s.t. } M_i \subset T_\ell} T_\ell
\end{aligned}
\right\} \Rightarrow \quad A_{ij} \neq 0 \text{ if } \exists \ell \text{ such that } M_i, M_j \subset T_\ell
$$

Algebraic systems resulting from finite element discretizations

Recap on finite elements

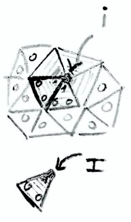
Sequential implementation

Parallel implementation

$$\boxed{\mathbf{Ax} = \mathbf{f}} \quad \text{with} \quad \begin{array}{|l} A_{ij} = \int_\Omega \nabla\phi_j \cdot \nabla\phi_i \, d\Omega + \int_\Omega \phi_j \phi_i \, d\Omega \quad (i,j = 1,\dots,N) \\ f_i = \int_\Omega f \phi_i \, d\Omega + \int_{\partial\Omega} g \phi_i \, d\Omega \quad (i = 1,\dots,N) \end{array}$$

Computation of matrix $\mathbf{A}$

The elements of $\mathbf{A}$ can be rewritten as:



$$\begin{aligned} A_{ij} &= \int_\Omega (\nabla\phi_j \cdot \nabla\phi_i + \phi_j\phi_i) \, d\Omega \\ &= \sum_{\ell=1}^{L} \int_{\mathring{T}_\ell} (\nabla\phi_j \cdot \nabla\phi_i + \phi_j\phi_i) \, dT \\ &= \sum_{\substack{\ell \text{ such that} \\ M_i, M_j \subset T_\ell}} \int_{\mathring{T}_\ell} (\nabla\phi_j \cdot \nabla\phi_i + \phi_j\phi_i) \; dT \end{aligned}$$

For each element T_ℓ, we define three local basis functions $(\tau_I^\ell)_{I=1}^3$ such that:

$$\boxed{\phi_i|_{T_\ell} = \tau_I^\ell} \quad (I = 1,2,3)$$

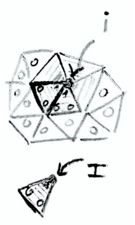
with $M_i \subset T_\ell$ and the corresponding indices $\texttt{LocalToGlobal}(\ell, I) = i$.

Sequential implementation — Building the system [2/3]

$$\boxed{\mathbf{Ax} = \mathbf{f}} \quad \text{with} \quad \left| \begin{array}{ll} A_{ij} = \int_\Omega \nabla\phi_j \cdot \nabla\phi_i \, d\Omega + \int_\Omega \phi_j \phi_i \, d\Omega & (i, j = 1, \dots, N) \\ f_i = \int_\Omega f \phi_i \, d\Omega + \int_{\partial\Omega} g \phi_i \, d\Omega & (i = 1, \dots, N) \end{array} \right.$$

Computation of matrix $\mathbf{A}$

The elements of $\mathbf{A}$ can rewritten as:



$$\begin{aligned} A_{ij} &= \int_\Omega (\nabla\phi_j \cdot \nabla\phi_i + \phi_j\phi_i) \, d\Omega \\ &= \sum_{\ell=1}^{L} \int_{\mathring{T}_\ell} (\nabla\phi_j \cdot \nabla\phi_i + \phi_j\phi_i) \, dT \\ &= \sum_{\substack{\ell \text{ such that} \\ M_i, M_j \subset T_\ell}} \int_{\mathring{T}_\ell} (\nabla\phi_j \cdot \nabla\phi_i + \phi_j\phi_i) \; dT \\ &= \sum_{\substack{\ell \text{ such that} \\ M_i, M_j \subset T_\ell}} \int_{\mathring{T}_\ell} \left(\nabla\tau_J^\ell \cdot \nabla\tau_I^\ell + \tau_J^\ell \tau_I^\ell\right) \; dT \quad \text{with} \left| \begin{array}{l} \texttt{LocalToGlobal}(\ell, I) = i \\ \texttt{LocalToGlobal}(\ell, J) = j \end{array} \right. \\ &= \sum_{\substack{\ell \text{ such that} \\ M_i, M_j \subset T_\ell}} A_{IJ}^\ell \quad \text{with } A_{IJ}^\ell = \int_{\mathring{T}_\ell} \left(\nabla\tau_J^\ell \cdot \nabla\tau_I^\ell + \tau_J^\ell \tau_I^\ell\right) dT \end{aligned}$$

The matrix $\mathbf{A}^\ell \in \mathbb{R}^{3\times3}$ is a local element-wise matrix corresponding to element T_ℓ.

Sequential implementation — Building the system [3/3]

Assembling of $\mathbf{A}$

```
Initialization: A = 0;
for ℓ = 1, ..., L do
    Computation of local matrix A^ℓ;
    for I = 1, 2, 3 do
        for J = 1, 2, 3 do
            i ← LocalToGlobal(ℓ, I);
            j ← LocalToGlobal(ℓ, J);
            A_ij ← A_ij + A^ℓ_IJ;
        end
    end
end
```

Assembling of $\mathbf{f}$ *(volume term)*

```
Initialization: f = 0;
for ℓ = 1, ..., L do
    Computation of local vector f^ℓ;
    for I = 1, 2, 3 do
        i ← LocalToGlobal(ℓ, I);
        f_i ← f_i + f^ℓ_I;
    end
end
```

Parallelization strategy?

Computation of a matrix-vector product $\mathbf{y} = \mathbf{A}\mathbf{z}$

We would like to compute

$$y_i = \sum_{j=1}^{N} A_{ij} z_j \qquad (i = 1, \dots, N)$$

where
- y_i is a resulting quantity associated to node M_i
- z_j is a quantity associated to node M_j *(e.g. solution, residual, ...)*
- $A_{ij} \neq 0$ only if there is at least one triangle containing M_i and M_j i.e. if (M_i, M_j) is an edge $\in \mathcal{E}_h$.

Matrix-vector product $\mathbf{y} = \mathbf{A}\mathbf{z}$

```
for M_i ∈ 𝓜_h do
    for M_j ∈ 𝓜_h such that (M_i, M_j) ∈ 𝓔_h do
        y_i ← y_i + A_ij z_j;
    end
end
```

Parallelization strategy?

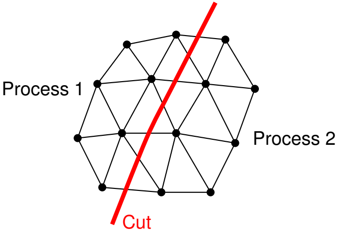
Algebraic systems resulting from finite element discretizations

Recap on finite elements

Sequential implementation

Parallel implementation

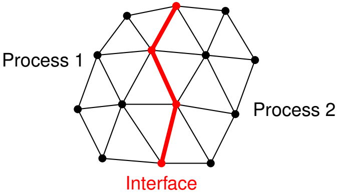
Partition by groups of vertices



Parallel assembling
not natural

Parallel matrix-vector product
rather natural

Partition by groups of elements



Parallel assembling
rather natural

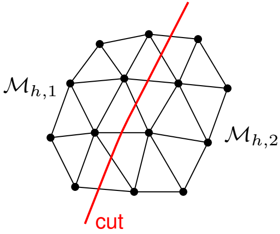
Parallel matrix-vector product
not natural

Parallel implementation — Strategy by groups of vertices [1/3]

The vertices/nodes are distributed between the differents processes:

$$\mathcal{M}_h = \bigcup_{p=1}^{P} \mathcal{M}_{h,p} \qquad \text{with } \mathcal{M}_{h,p} \cap \mathcal{M}_{h,q} = \emptyset \text{ if } p \neq q$$

where $\mathcal{M}_{h,p}$ is the group of vertices/nodes corresponding to process p.



Strategy for matrix-vector product $\mathbf{y} = \mathbf{A}\mathbf{z}$

- Each process p computes the part of $\mathbf{y}$ corresponding to nodes $\mathcal{M}_{h,p}$:

$$y_i = \sum_j A_{ij} z_j \text{ with } i \in \mathcal{M}_{h,p}$$

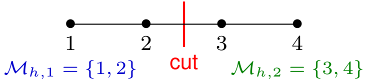
- Each process p stores the elements of $\mathbf{z}$ and $\mathbf{y}$, and the lines of $\mathbf{A}$ with indices $\in \mathcal{M}_{h,p}$.

A priori, no duplication of data, but computing $\mathbf{y}$ requires communications.
The edges between nodes of $\mathcal{M}_{h,1}$ and $\mathcal{M}_{h,2}$ indicates the dependencies.

Parallel implementation — Strategy by groups of vertices [2/3]

Illustration in 1D

Configuration with 3 P1 elements and 4 nodes:



Matrix-vector product:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} \cdot & \cdot & 0 & 0 \\ \cdot & \cdot & \cdot & 0 \\ 0 & \cdot & \cdot & \cdot \\ 0 & 0 & \cdot & \cdot \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} \cdot & \cdot & 0 & 0 \\ \cdot & \cdot & \cdot & 0 \\ 0 & \cdot & \cdot & \cdot \\ 0 & 0 & \cdot & \cdot \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} + \begin{bmatrix} \cdot & \cdot & 0 & 0 \\ \cdot & \cdot & \cdot & 0 \\ 0 & \cdot & \cdot & \cdot \\ 0 & 0 & \cdot & \cdot \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix}$$

Computed by process 1
$\mathbf{y}_1 = \mathbf{A}_1 \mathbf{z}$

Computed by process 2
$\mathbf{y}_2 = \mathbf{A}_2 \mathbf{z}$

Computing y_2 (on proc 1) requires z_1 and z_2 (on proc 1) and z_3 (on proc p_2)

Parallel algorithms

Parallel assembly of $\mathbf{A}$
On each process $p = 1, \ldots, P$**:** Assemble matrix $\mathbf{A}_p$ corresponding to lines of $\mathbf{A}$ with indices $i \in \mathcal{M}_{h,p}$;

Parallel matrix-vector product $\mathbf{y} = \mathbf{Az}$

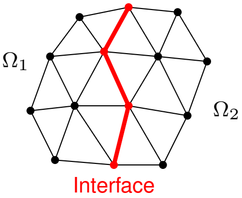
On each process $p = 1, \ldots, P$**:**
for q *such that* $\mathcal{M}_{h,p} \cap \mathcal{M}_{h,q} \neq \emptyset$ **do**
    Send values $\{z_i\}_{i \in \mathcal{M}_{h,p} \text{ s.t. } \exists (i,j) \in \mathcal{E}_h \text{ with } j \in \mathcal{M}_{h,q}}$ to process q;
    Recv values $\{z_j\}_{j \in \mathcal{M}_{h,q} \text{ s.t. } \exists (i,j) \in \mathcal{E}_h \text{ with } i \in \mathcal{M}_{h,p}}$ from process q;
end
for $i \in \mathcal{M}_{h,p}$ **do**
    for j *such that* (i,j) *is an edge* **do**
        $y_i \leftarrow y_i + A_{p,ij} z_j$;
    end
end

Temporary storage, to store nodal values corresponding to neighboring process

The elements are distributed between the differents processes:

$$\overline{\Omega} = \bigcup_{p=1}^{P} \overline{\Omega}_p$$

where $\overline{\Omega}_p$ is the group of elements corresponding to process p.
If $\mathcal{M}_{h,p}$ is the set of nodes/vertices of Ω_p, then $\mathcal{M}_{h,p} \cap \mathcal{M}_{h,q} \neq \emptyset$ if $\overline{\Omega}_p \cap \overline{\Omega}_q \neq \emptyset$.



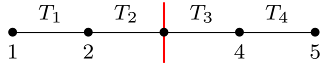
Strategy for matrix-vector product $\mathbf{y} = \mathbf{A}\mathbf{z}$

- Each process p performs the operations corresponding to the elements of $\overline{\Omega}_p$.
- Each process p stores elements of $\mathbf{z}$ and $\mathbf{y}$ and the lines of $\mathbf{A}$ corresponding to vertices/nodes $\mathcal{M}_{h,p}$ (i.e. both interior and interface nodes).

Duplication of data corresponding to interface nodes

Illustration in 1D

Configuration with 4 P1 elements and 5 nodes:



$\mathcal{T}_{h,1} = \{T_1, T_2\}$ $\quad$ $\mathcal{T}_{h,2} = \{T_3, T_4\}$

$\mathcal{M}_{h,1} = \{1, 2, 3\}$ $\quad$ $\mathcal{M}_{h,2} = \{3, 4, 5\}$

Matrix-vector product:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} \cdot & \cdot & 0 & 0 & 0 \\ \cdot & \cdot & \cdot & 0 & 0 \\ 0 & \cdot & \cdot & \cdot & 0 \\ 0 & 0 & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & \cdot & \cdot \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \end{bmatrix}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} \cdot & \cdot & 0 & 0 & 0 \\ \cdot & \cdot & \cdot & 0 & 0 \\ 0 & \cdot & \cdot & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \cdot & \cdot & 0 \\ 0 & 0 & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & \cdot & \cdot \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \end{bmatrix}$$

Computed by proc. 1 $\quad$ Computed by proc. 2

$\mathbf{y}_1 = \mathbf{A}_1 \mathbf{z}$ $\quad$ $\mathbf{y}_2 = \mathbf{A}_2 \mathbf{z}$

Vector $\mathbf{y}_p$ contains total sums $\sum_j A_{ij} z_j$ for the internal nodes and partial sums for the interface nodes.

Parallel algorithms

Parallel assembly of $\mathbf{A}$
On each process $p = 1, \ldots, P$**:** Assemble matrix $\mathbf{A}_p$ corresponding to elements of $\mathcal{T}_p$;

Parallel matrix-vector product $\mathbf{y} = \mathbf{A}\mathbf{z}$

On each process $p = 1, \ldots, P$**:**
for $i \in \mathcal{M}_{h,p}$ **do**
    for $j \in \mathcal{M}_{h,p}$ *such that* $(i,j) \in \mathcal{E}_h$ **do**
        $y_i \leftarrow y_i + A_{p,ij} z_j$;
    end
end
for q *such that* $\mathcal{M}_{h,p} \cap \mathcal{M}_{h,q} \neq \emptyset$ **do**
    Send/Recv values $\{y_i\}$ for the interface nodes $\mathcal{M}_{h,p} \cap \mathcal{M}_{h,q}$;
    Accumulate these values to compute the total sums;
end

Summary

- **Finite element scheme**
  - Exact/Approximate variational formulation of an elliptic problem
  - P_1 finite elements — Convergence rate: h^2 in L^2-norm and h^1 in H^1-norm
  - Linear system $\mathbf{Ax} = \mathbf{f}$:
    - $\mathbf{A} \in \mathbb{R}^{N \times N}$ is symmetric, positive definite, sparse
    - $\mathbf{x} \in \mathbb{R}^{N}$ contains the nodal values of the solution
    - N is the number of nodes/vertices
- **Implementation**
  - Main loops:
    - Loop over the elements for the matrix assembly
    - Loop over the unknowns/nodes/vertices for solving the linear system
  - Strategy for parallel implementation:
    - Partitionning by groups of nodes
    - Partitionning by groups of elements