Lecture 11: Three-dimensional topology optimization. Domain Decomposition methods and parallel computing.

Florian Feppon

Spring 2022 – Seminar for Applied Mathematics

ETH zürich

Outline

Outline

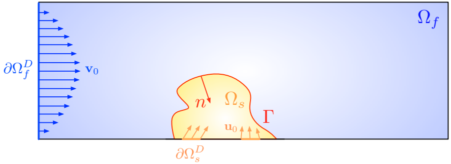
1. Challenges in three-dimensional topology optimization
2. A glimpse on domain decomposition methods and PETSc
3. Implementing a topology optimization test case

Outline

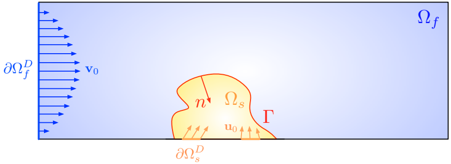
1. Challenges in three-dimensional topology optimization
2. A glimpse on domain decomposition methods and PETSc
3. Implementing a topology optimization test case

Outline

The coupled physics model



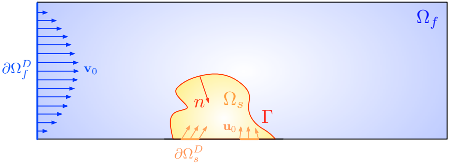
The coupled physics model



- Incompressible Navier-Stokes system for the velocity and pressure $(\boldsymbol{v}, p)$ in Ω_f

$$-\mathrm{div}(\sigma_f(\boldsymbol{v}, p)) + \rho \nabla \boldsymbol{v}\, \boldsymbol{v} = \boldsymbol{f}_f \text{ in } \Omega_f$$

The coupled physics model



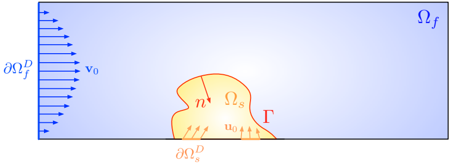
- Incompressible Navier-Stokes system for the velocity and pressure $(\boldsymbol{v}, p)$ in Ω_f

$$-\mathrm{div}(\sigma_f(\boldsymbol{v}, p)) + \rho\nabla\boldsymbol{v}\,\boldsymbol{v} = \boldsymbol{f}_f \text{ in } \Omega_f$$

- Convection-diffusion for the temperature T in Ω_f and Ω_s:

$$\begin{aligned} -\mathrm{div}(k_f\nabla T_f) + \rho\boldsymbol{v}\cdot\nabla T_f &= Q_f \quad \text{in } \Omega_f \\ -\mathrm{div}(k_s\nabla T_s) &= Q_s \quad \text{in } \Omega_s \end{aligned}$$

The coupled physics model



- Incompressible Navier-Stokes system for the velocity and pressure $(\boldsymbol{v}, p)$ in Ω_f

$$-\mathrm{div}(\sigma_f(\boldsymbol{v}, p)) + \rho\nabla\boldsymbol{v}\,\boldsymbol{v} = \boldsymbol{f}_f \text{ in } \Omega_f$$

- Convection-diffusion for the temperature T in Ω_f and Ω_s:

$$\begin{aligned} -\mathrm{div}(k_f\nabla T_f) + \rho\boldsymbol{v}\cdot\nabla T_f &= Q_f \quad \text{in } \Omega_f \\ -\mathrm{div}(k_s\nabla T_s) &= Q_s \quad \text{in } \Omega_s \end{aligned}$$

- Thermo-elasticity with fluid-structure interaction for $\boldsymbol{u}$ iin Ω_s :

$$\begin{aligned} -\mathrm{div}(\sigma_s(\boldsymbol{u}, T_s)) &= \boldsymbol{f}_s && \text{in } \Omega_s \\ \sigma_s(\boldsymbol{u}, T_s)\cdot\boldsymbol{n} &= \sigma_f(\boldsymbol{v}, p)\cdot\boldsymbol{n} && \text{on } \Gamma. \end{aligned}$$

The coupled physics model

Our goal: solve generic topology optimization problems of the type

$$\begin{aligned} \min_{\Gamma} \quad & J(\Gamma, \boldsymbol{v}(\Gamma), p(\Gamma), T(\Gamma), \boldsymbol{u}(\Gamma)) \\ \text{s.c.} \quad & g_i(\Gamma, \boldsymbol{v}(\Gamma), p(\Gamma), T(\Gamma), \boldsymbol{u}(\Gamma)) = 0, \; 1 \leq i \leq p \cdot \\ & h_i(\Gamma, \boldsymbol{v}(\Gamma), p(\Gamma), T(\Gamma), \boldsymbol{u}(\Gamma)) \leq 0, \; 1 \leq i \leq q \end{aligned}$$

where $\boldsymbol{u}(\Gamma)$, $\boldsymbol{v}(\Gamma)$, $p(\Gamma)$, $T(\Gamma)$ are the solutions to PDE models.

Challenges in three-dimensional topology optimization

- In theory, no change from the point of view of the methodology
  1. same PDEs for the state variables

Challenges in three-dimensional topology optimization

- In theory, no change from the point of view of the methodology
  1. same PDEs for the state variables
  2. same PDEs for the adjoint variables

Challenges in three-dimensional topology optimization

- In theory, no change from the point of view of the methodology
  1. same PDEs for the state variables
  2. same PDEs for the adjoint variables
  3. same shape derivatives

Challenges in three-dimensional topology optimization

- In theory, no change from the point of view of the methodology
  1. same PDEs for the state variables
  2. same PDEs for the adjoint variables
  3. same shape derivatives
  4. same null space optimization algorithm

Challenges in three-dimensional topology optimization

- In theory, no change from the point of view of the methodology
  1. same PDEs for the state variables
  2. same PDEs for the adjoint variables
  3. same shape derivatives
  4. same null space optimization algorithm
  5. same shape updates $\Omega \mapsto (I + \boldsymbol{\theta})\Omega$.

Challenges in three-dimensional topology optimization

- In theory, no change from the point of view of the methodology
  1. same PDEs for the state variables
  2. same PDEs for the adjoint variables
  3. same shape derivatives
  4. same null space optimization algorithm
  5. same shape updates $\Omega \mapsto (I + \boldsymbol{\theta})\Omega$.

Challenges in three-dimensional topology optimization

- In theory, no change from the point of view of the methodology
  1. same PDEs for the state variables
  2. same PDEs for the adjoint variables
  3. same shape derivatives
  4. same null space optimization algorithm
  5. same shape updates $\Omega \mapsto (\boldsymbol{I} + \boldsymbol{\theta})\Omega$.
- In practice, the implementation must be **completely** revised.

Challenges in three-dimensional topology optimization

Why is it hard to do three-dimensional Topology Optimization ?

- the size of the meshes become very large:

Challenges in three-dimensional topology optimization

Why is it hard to do three-dimensional Topology Optimization ?

- the size of the meshes become very large:
  1. $O(N^3)$ instead of $O(N^2)$ vertices for resolving a domain with resolution $1/N$

Challenges in three-dimensional topology optimization

Why is it hard to do three-dimensional Topology Optimization ?

- the size of the meshes become very large:
    1. $O(N^3)$ instead of $O(N^2)$ vertices for resolving a domain with resolution $1/N$
    2. vectorial variables have three components instead of two

Challenges in three-dimensional topology optimization

Why is it hard to do three-dimensional Topology Optimization ?

- the size of the meshes become very large:
  1. $O(N^3)$ instead of $O(N^2)$ vertices for resolving a domain with resolution $1/N$
  2. vectorial variables have three components instead of two
  3. the combinatorial complexity of the mesh increases: approximately 6 tetrahedra per vertices rather than 3 triangles per vertices.

Challenges in three-dimensional topology optimization

Why is it hard to do three-dimensional Topology Optimization ?

- the size of the meshes become very large:
  1. $O(N^3)$ instead of $O(N^2)$ vertices for resolving a domain with resolution $1/N$
  2. vectorial variables have three components instead of two
  3. the combinatorial complexity of the mesh increases: approximately 6 tetrahedra per vertices rather than 3 triangles per vertices.

Challenges in three-dimensional topology optimization

Why is it hard to do three-dimensional Topology Optimization ?

- the size of the meshes become very large:
  1. $O(N^3)$ instead of $O(N^2)$ vertices for resolving a domain with resolution $1/N$
  2. vectorial variables have three components instead of two
  3. the combinatorial complexity of the mesh increases: approximately 6 tetrahedra per vertices rather than 3 triangles per vertices.
- Finite element linear systems become very large: **hard to store them in memory, hard to solve them with direct methods (LU decomposition)**. Need for adapted numerical technique.

Challenges in three-dimensional topology optimization

Why is it hard to do three-dimensional Topology Optimization ?

- the size of the meshes become very large:
  1. $O(N^3)$ instead of $O(N^2)$ vertices for resolving a domain with resolution $1/N$
  2. vectorial variables have three components instead of two
  3. the combinatorial complexity of the mesh increases: approximately 6 tetrahedra per vertices rather than 3 triangles per vertices.
- Finite element linear systems become very large: **hard to store them in memory, hard to solve them with direct methods (LU decomposition)**. Need for adapted numerical technique.
- Remeshing becomes also quite expensive in 3D due to the number of combinatorial operations (while it is inexpensive in 2D).

A solution: parallel computing and domain decomposition.

- **parallel computing**: use of distributed processors and memory. Operations are done independently with a few **synchronization** operations.

A solution: parallel computing and domain decomposition.

- **parallel computing**: use of distributed processors and memory. Operations are done independently with a few **synchronization** operations.
- **domain decomposition**: the computational mesh is partitioned into as many submeshes as the number of CPUs. FEM Operations are performed independently on these submeshes (matrix assembly, liner system solve, etc.).

Challenges in three-dimensional topology optimization

A solution: parallel computing and domain decomposition.

- **parallel computing**: use of distributed processors and memory. Operations are done independently with a few **synchronization** operations.
- **domain decomposition**: the computational mesh is partitioned into as many submeshes as the number of CPUs. FEM Operations are performed independently on these submeshes (matrix assembly, liner system solve, etc.).
- **preconditioners**: iterative techniques are used to solve the linear systems. The number of iterations required is decreased by *preconditioning*.

Challenges in three-dimensional topology optimization

A solution: parallel computing and domain decomposition.

- **parallel computing**: use of distributed processors and memory. Operations are done independently with a few **synchronization** operations.
- **domain decomposition**: the computational mesh is partitioned into as many submeshes as the number of CPUs. FEM Operations are performed independently on these submeshes (matrix assembly, liner system solve, etc.).
- **preconditioners**: iterative techniques are used to solve the linear systems. The number of iterations required is decreased by *preconditioning*.

A solution: parallel computing and domain decomposition.

- **parallel computing**: use of distributed processors and memory. Operations are done independently with a few **synchronization** operations.
- **domain decomposition**: the computational mesh is partitioned into as many submeshes as the number of CPUs. FEM Operations are performed independently on these submeshes (matrix assembly, liner system solve, etc.).
- **preconditioners**: iterative techniques are used to solve the linear systems. The number of iterations required is decreased by *preconditioning*.

It is **possible** to mesh in parallel (ParMmg). However the level-set discretization feature is still not yet available.

Challenges in three-dimensional topology optimization

A solution: parallel computing and domain decomposition.

- **parallel computing**: use of distributed processors and memory. Operations are done independently with a few **synchronization** operations.
- **domain decomposition**: the computational mesh is partitioned into as many submeshes as the number of CPUs. FEM Operations are performed independently on these submeshes (matrix assembly, liner system solve, etc.).
- **preconditioners**: iterative techniques are used to solve the linear systems. The number of iterations required is decreased by *preconditioning*.

It is **possible** to mesh in parallel (ParMmg). However the level-set discretization feature is still not yet available.

In the present class, we focus on making FEM operations in parallel.

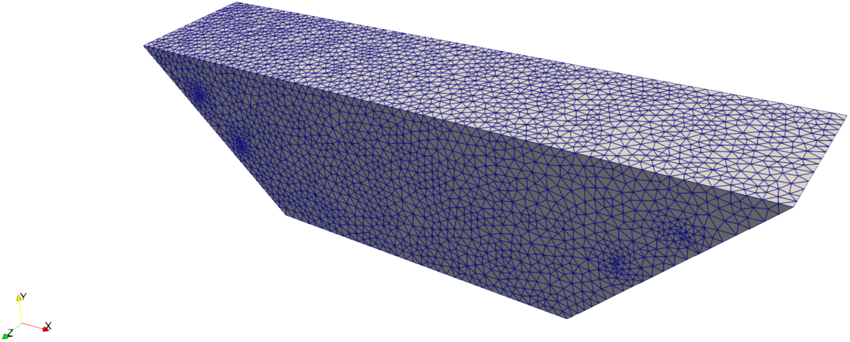
Outline

1. Challenges in three-dimensional topology optimization
2. A glimpse on domain decomposition methods and PETSc
3. Implementing a topology optimization test case

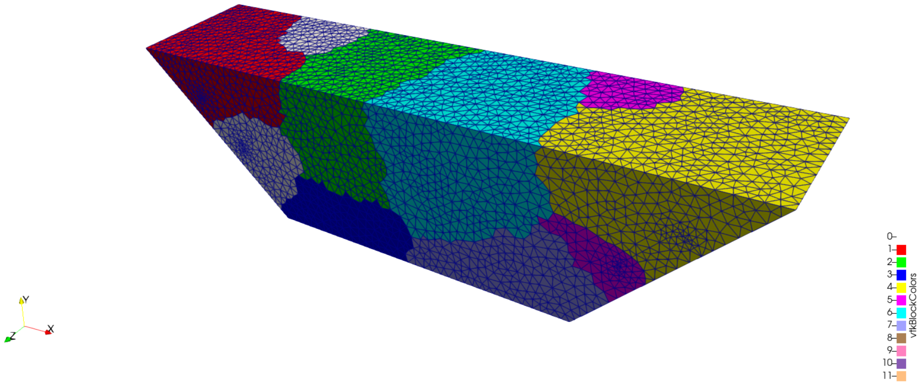
Domain decomposition methods

First step: partitioning the mesh $\mathcal{T}$ into $\mathcal{T}_1, \ldots, \mathcal{T}_{\mathtt{ncpu}}$.

Domain decomposition methods



Domain decomposition methods



Domain decomposition methods

First step: partitioning the mesh $\mathcal{T}$ into $\mathcal{T}_1$, ..., $\mathcal{T}_{\mathtt{ncpu}}$.

Domain decomposition methods

First step: partitioning the mesh $\mathcal{T}$ into $\mathcal{T}_1, \ldots, \mathcal{T}_{\texttt{ncpu}}$.
Several librairies available for this: `metis`, `pardiso`, available in FreeFEM.

Domain decomposition methods

First step: partitioning the mesh $\mathcal{T}$ into $\mathcal{T}_1, \ldots, \mathcal{T}_{\texttt{ncpu}}$.
Several librairies available for this: `metis`, `pardiso`, available in FreeFEM.
Ghost cells are sometimes added to the partitioned submeshes for the **additive schwarz method**.

Domain decomposition methods

First step: partitioning the mesh $\mathcal{T}$ into $\mathcal{T}_1, \ldots, \mathcal{T}_{\texttt{ncpu}}$.
Several librairies available for this: `metis`, `pardiso`, available in FreeFEM.
Ghost cells are sometimes added to the partitioned submeshes for the **additive schwarz method**.
Sometimes the domains $\mathcal{T}_i$ overlap: we use then a partition of unity $(D_i)_{1\leq i\leq \texttt{ncpu}}$ such that

$$1 = \sum_{1\leq i\leq \texttt{ncpu}} D_i.$$

Domain decomposition methods

First step: partitioning the mesh $\mathcal{T}$ into $\mathcal{T}_1, \ldots, \mathcal{T}_{\texttt{ncpu}}$.
Several librairies available for this: `metis`, `pardiso`, available in FreeFEM.
Ghost cells are sometimes added to the partitioned submeshes for the **additive schwarz method**.
Sometimes the domains $\mathcal{T}_i$ overlap: we use then a partition of unity $(D_i)_{1 \le i \le \texttt{ncpu}}$ such that

$$1 = \sum_{1 \le i \le \texttt{ncpu}} D_i.$$

Each of the submeshes have their own address space. The goal is to **minimize communication** between processors.

Domain decomposition methods

First step: partitioning the mesh $\mathcal{T}$ into $\mathcal{T}_1, \ldots, \mathcal{T}_{\texttt{ncpu}}$.
Several librairies available for this: `metis`, `pardiso`, available in FreeFEM.
Ghost cells are sometimes added to the partitioned submeshes for the **additive schwarz method**.
Sometimes the domains $\mathcal{T}_i$ overlap: we use then a partition of unity $(D_i)_{1\leq i\leq \texttt{ncpu}}$ such that

$$1 = \sum_{1\leq i\leq \texttt{ncpu}} D_i.$$

Each of the submeshes have their own address space. The goal is to **minimize communication** between processors.
It can be important to have access to the restriction operators $R_i : \mathcal{T} \to \mathcal{T}_i$ to perform global operations (e.g. reconstruct the global solution from the local one).

Domain decomposition methods

First step: partitioning the mesh $\mathcal{T}$ into $\mathcal{T}_1, \ldots, \mathcal{T}_{\mathrm{ncpu}}$.
Several librairies available for this: `metis`, `pardiso`, available in FreeFEM.
Ghost cells are sometimes added to the partitioned submeshes for the **additive schwarz method**.
Sometimes the domains $\mathcal{T}_i$ overlap: we use then a partition of unity $(D_i)_{1\le i\le \mathrm{ncpu}}$ such that

$$1 = \sum_{1\le i\le \mathrm{ncpu}} D_i.$$

Each of the submeshes have their own address space. The goal is to **minimize communication** between processors.
It can be important to have access to the restriction operators $R_i : \mathcal{T} \to \mathcal{T}_i$ to perform global operations (e.g. reconstruct the global solution from the local one).
In general, we try to avoid as much as possible to use the global solution and prefer to do **everything in parallel** (e.g. matrix assemblies, computing the objective function, etc.).

Domain decomposition methods

First step: partitioning the mesh $\mathcal{T}$ into $\mathcal{T}_1, \ldots, \mathcal{T}_{\mathtt{ncpu}}$.
Several librairies available for this: `metis`, `pardiso`, available in FreeFEM.
Ghost cells are sometimes added to the partitioned submeshes for the **additive schwarz method**.
Sometimes the domains $\mathcal{T}_i$ overlap: we use then a partition of unity $(D_i)_{1\leq i\leq \mathtt{ncpu}}$ such that

$$1 = \sum_{1\leq i\leq \mathtt{ncpu}} D_i.$$

Each of the submeshes have their own address space. The goal is to **minimize communication** between processors.
It can be important to have access to the restriction operators $R_i : \mathcal{T} \to \mathcal{T}_i$ to perform global operations (e.g. reconstruct the global solution from the local one).
In general, we try to avoid as much as possible to use the global solution and prefer to do **everything in parallel** (e.g. matrix assemblies, computing the objective function, etc.).
For instance,

$$\int_{\mathcal{T}} j(u) Dx = \sum_{1\leq i\leq \mathtt{ncpu}} \int_{\mathcal{T}_i} D_i j(R_i u) \mathrm{d}x.$$

Domain decomposition methods

First step: partitioning the mesh $\mathcal{T}$ into $\mathcal{T}_1, \dots, \mathcal{T}_{\texttt{ncpu}}$.
Several librairies available for this: `metis`, `pardiso`, available in FreeFEM.
Ghost cells are sometimes added to the partitioned submeshes for the **additive schwarz method**.
Sometimes the domains $\mathcal{T}_i$ overlap: we use then a partition of unity $(D_i)_{1\leq i\leq \texttt{ncpu}}$ such that

$$1 = \sum_{1\leq i\leq \texttt{ncpu}} D_i.$$

Each of the submeshes have their own address space. The goal is to **minimize communication** between processors.
It can be important to have access to the restriction operators $R_i : \mathcal{T} \to \mathcal{T}_i$ to perform global operations (e.g. reconstruct the global solution from the local one).
In general, we try to avoid as much as possible to use the global solution and prefer to do **everything in parallel** (e.g. matrix assemblies, computing the objective function, etc.).
For instance,

$$\int_{\mathcal{T}} j(u) Dx = \sum_{1\leq i\leq \texttt{ncpu}} \int_{\mathcal{T}_i} D_i j(R_i u)\mathrm{d}x.$$

Everything must be thought in parallel $\to$ completely revised implementation.

- Parallel computing is achieved thanks to a dedicated **Message Passing Interface** (MPI) for communicating between CPUs. Several possibles: OpenMPI, mpich, intel-mpi, etc.

Domain decomposition methods

- Parallel computing is achieved thanks to a dedicated **Message Passing Interface** (MPI) for communicating between CPUs. Several possibles: OpenMPI, mpich, intel-mpi, etc.
- FreeFEM is interfaced with them through the command FreeFem++-mpi.

Domain decomposition methods

- Parallel computing is achieved thanks to a dedicated **Message Passing Interface** (MPI) for communicating between CPUs. Several possibles: OpenMPI, mpich, intel-mpi, etc.
- FreeFEM is interfaced with them through the command FreeFem++-mpi.
- Script is run simultaneously by several processus. Communication of data between processes is achieved by several commands: `mpiComm`, `mpiGroup`, `mpiRequest`, `broadcast`, `mpiAllGather`...

Domain decomposition methods

Linear system solves:

$$Ax = b.$$

- Most efficient method, **if enough memory** : factorization method.

Domain decomposition methods

Linear system solves:

$$Ax = b.$$

- Most efficient method, **if enough memory** : factorization method.

Domain decomposition methods

Linear system solves:

$$Ax = b.$$

- Most efficient method, **if enough memory** : factorization method. $A = LU$ with L and U lower and upper-triangular. Then $A^{-1} = U^{-1}L^{-1}$.

Domain decomposition methods

Linear system solves:

$$Ax = b.$$

- Most efficient method, **if enough memory** : factorization method. $A = LU$ with L and U lower and upper-triangular. Then $A^{-1} = U^{-1}L^{-1}$. Variants to do it in parallel. However requires a lot of memory (complexity of order $O(N^2)$ with N the number of degrees of freedom).

Domain decomposition methods

Linear system solves:

$$Ax = b.$$

- Most efficient method, **if enough memory** : factorization method. $A = LU$ with L and U lower and upper-triangular. Then $A^{-1} = U^{-1}L^{-1}$. Variants to do it in parallel. However requires a lot of memory (complexity of order $O(N^2)$ with N the number of degrees of freedom).
- If not enough memory, (for very large systems) it is more efficient to use **iterative methods** with **preconditioners**.

Domain decomposition methods

Linear system solves:

$$Ax = b.$$

- Most efficient method, **if enough memory** : factorization method. $A = LU$ with L and U lower and upper-triangular. Then $A^{-1} = U^{-1}L^{-1}$. Variants to do it in parallel. However requires a lot of memory (complexity of order $O(N^2)$ with N the number of degrees of freedom).
- If not enough memory, (for very large systems) it is more efficient to use **iterative methods** with **preconditioners**.
- The conjugate gradient method (CG) for positive symmetric A:

  $$x_0 := b; \quad x_{n+1} = x_n - \alpha_n \Pi_n (Ax_n - b)$$

  where Π_n is the orthogonal projection onto $\mathrm{span}(x_0, x_1, \ldots, x_n)^{\perp}$ and α_n an optimal time step.

Domain decomposition methods

Linear system solves:

$$Ax = b.$$

- Most efficient method, **if enough memory** : factorization method. $A = LU$ with L and U lower and upper-triangular. Then $A^{-1} = U^{-1}L^{-1}$. Variants to do it in parallel. However requires a lot of memory (complexity of order $O(N^2)$ with N the number of degrees of freedom).
- If not enough memory, (for very large systems) it is more efficient to use **iterative methods** with **preconditioners**.
- The conjugate gradient method (CG) for positive symmetric A:

  $$x_0 := b; \quad x_{n+1} = x_n - \alpha_n \Pi_n (Ax_n - b)$$

  where Π_n is the orthogonal projection onto $\mathrm{span}(x_0, x_1, \dots, x_n)^{\perp}$ and α_n an optimal time step.

Domain decomposition methods

Linear system solves:

$$Ax = b.$$

- Most efficient method, **if enough memory** : factorization method. $A = LU$ with L and U lower and upper-triangular. Then $A^{-1} = U^{-1}L^{-1}$. Variants to do it in parallel. However requires a lot of memory (complexity of order $O(N^2)$ with N the number of degrees of freedom).
- If not enough memory, (for very large systems) it is more efficient to use **iterative methods** with **preconditioners**.
- The conjugate gradient method (CG) for positive symmetric A:

  $$x_0 := b; \quad x_{n+1} = x_n - \alpha_n \Pi_n (Ax_n - b)$$

  where Π_n is the orthogonal projection onto $\mathrm{span}(x_0, x_1, \ldots, x_n)^{\perp}$ and α_n an optimal time step. CG requires only **matrix-vector** products $\to$ efficient in memory.

Domain decomposition methods

- CG can be extended to arbitrary matrices (Generalized Minimal Residual Method, GMRES).

Domain decomposition methods

- CG can be extended to arbitrary matrices (Generalized Minimal Residual Method, GMRES).
- Iterative methods can be slow to converge.

Domain decomposition methods

- CG can be extended to arbitrary matrices (Generalized Minimal Residual Method, GMRES).
- Iterative methods can be slow to converge.
- They are accelerated by **preconditioning**.

Domain decomposition methods

- CG can be extended to arbitrary matrices (Generalized Minimal Residual Method, GMRES).
- Iterative methods can be slow to converge.
- They are accelerated by **preconditioning**.

Domain decomposition methods

- CG can be extended to arbitrary matrices (Generalized Minimal Residual Method, GMRES).
- Iterative methods can be slow to converge.
- They are accelerated by **preconditioning**.

$$Ax = b \quad \Leftrightarrow M_l A M_r y = M_l b \text{ with } x = M_r y.$$

where M_l and M_r are called left and right-preconditioners.

Domain decomposition methods

- CG can be extended to arbitrary matrices (Generalized Minimal Residual Method, GMRES).
- Iterative methods can be slow to converge.
- They are accelerated by **preconditioning**.

$$Ax = b \quad \Leftrightarrow M_l A M_r y = M_l b \text{ with } x = M_r y.$$

  where M_l and M_r are called left and right-preconditioners.
- The error rate for CG to reach a desired accuracy decreases as

$$\left(\frac{\sqrt{\kappa(A)}-1}{\sqrt{\kappa(A)}+1}\right)^k$$

  where $\kappa(A) = \lambda_{\max}(A)/\lambda_{\min}(A)$ is the condition number of the matrix.

Domain decomposition methods

- CG can be extended to arbitrary matrices (Generalized Minimal Residual Method, GMRES).
- Iterative methods can be slow to converge.
- They are accelerated by **preconditioning**.

$$Ax = b \quad \Leftrightarrow M_l A M_r y = M_l b \text{ with } x = M_r y.$$

  where M_l and M_r are called left and right-preconditioners.
- The error rate for CG to reach a desired accuracy decreases as

$$\left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^k$$

  where $\kappa(A) = \lambda_{\max}(A)/\lambda_{\min}(A)$ is the condition number of the matrix.
- A "good" preconditioner should be chosen such that

$$M_l A M_r \simeq I.$$

Domain decomposition methods

- CG can be extended to arbitrary matrices (Generalized Minimal Residual Method, GMRES).
- Iterative methods can be slow to converge.
- They are accelerated by **preconditioning**.

$$Ax = b \quad \Leftrightarrow M_l A M_r y = M_l b \text{ with } x = M_r y.$$

  where M_l and M_r are called left and right-preconditioners.
- The error rate for CG to reach a desired accuracy decreases as

$$\left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^k$$

  where $\kappa(A) = \lambda_{\max}(A)/\lambda_{\min}(A)$ is the condition number of the matrix.
- A "good" preconditioner should be chosen such that

$$M_l A M_r \simeq I.$$

Domain decomposition methods

- CG can be extended to arbitrary matrices (Generalized Minimal Residual Method, GMRES).
- Iterative methods can be slow to converge.
- They are accelerated by **preconditioning**.

$$Ax = b \quad \Leftrightarrow M_l A M_r y = M_l b \text{ with } x = M_r y.$$

  where M_l and M_r are called left and right-preconditioners.
- The error rate for CG to reach a desired accuracy decreases as

$$\left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^k$$

  where $\kappa(A) = \lambda_{\max}(A)/\lambda_{\min}(A)$ is the condition number of the matrix.
- A "good" preconditioner should be chosen such that

$$M_l A M_r \simeq I.$$

  M_l and M_r need to be approximate left and right-inverses for A.

The additive schwarz method:

- Assemble the matrix A in parallel (construct the matrix A_i on each submesh and update the connecting nodes).

Domain decomposition methods

The additive schwarz method:

- Assemble the matrix A in parallel (construct the matrix A_i on each submesh and update the connecting nodes).
- Define a preconditioner M for the matrix A.

Domain decomposition methods

The additive schwarz method:

- Assemble the matrix A in parallel (construct the matrix A_i on each submesh and update the connecting nodes).
- Define a preconditioner M for the matrix A.
- Solve $MAx = Mb$ with CG or GMRES.

The additive schwarz method:

- Assemble the matrix A in parallel (construct the matrix A_i on each submesh and update the connecting nodes).
- Define a preconditioner M for the matrix A.
- Solve $MAx = Mb$ with CG or GMRES.
- For the addititive Schwarz method, we use the block preconditioner

$$M := \begin{bmatrix} A_1^{-1} & & \\ & \ddots & \\ & & A_m^{-1} \end{bmatrix},$$

where A_i is the restriction of the matrix A to the subdomain i.

Domain decomposition methods

The additive schwarz method:

- Assemble the matrix A in parallel (construct the matrix A_i on each submesh and update the connecting nodes).
- Define a preconditioner M for the matrix A.
- Solve $MAx = Mb$ with CG or GMRES.
- For the addititive Schwarz method, we use the block preconditioner

$$M := \begin{bmatrix} A_1^{-1} & & \\ & \ddots & \\ & & A_m^{-1} \end{bmatrix},$$

  where A_i is the restriction of the matrix A to the subdomain i.
- The inverse of each of the matrices A_i can itself be computed with iterative methods, with **physics dependent** preconditioners.

Physics dependent preconditioners:

- Linear elasticity: Geometric Algebraic Multigrid (GAMG) + CG

Physics dependent preconditioners:

- Linear elasticity: Geometric Algebraic Multigrid (GAMG) + CG

Domain decomposition methods

Physics dependent preconditioners:

- Linear elasticity: Geometric Algebraic Multigrid (GAMG) + CG
  Use of coarse meshes to smoothen high frequency errors

Domain decomposition methods

Physics dependent preconditioners:

- Linear elasticity: Geometric Algebraic Multigrid (GAMG) + CG
  Use of coarse meshes to smoothen high frequency errors
- Thermal conduction : hypre solver (use also boomerAmg or algebraic multigric)

Physics dependent preconditioners:

- Linear elasticity: Geometric Algebraic Multigrid (GAMG) + CG
  Use of coarse meshes to smoothen high frequency errors
- Thermal conduction : hypre solver (use also boomerAmg or algebraic multigric)
- Navier-Stokes equations : fieldsplit preconditioner (using a Schur complement method) and divergence penalization for the Oseen problem (Moulin, Jolivet, Marquet 2019)

Domain decomposition methods

Physics dependent preconditioners:

- Linear elasticity: Geometric Algebraic Multigrid (GAMG) + CG
  Use of coarse meshes to smoothen high frequency errors
- Thermal conduction : hypre solver (use also boomerAmg or algebraic multigric)
- Navier-Stokes equations : fieldsplit preconditioner (using a Schur complement method) and divergence penalization for the Oseen problem (Moulin, Jolivet, Marquet 2019)

Domain decomposition methods

Physics dependent preconditioners:

- Linear elasticity: Geometric Algebraic Multigrid (GAMG) + CG
  Use of coarse meshes to smoothen high frequency errors
- Thermal conduction : hypre solver (use also boomerAmg or algebraic multigric)
- Navier-Stokes equations : fieldsplit preconditioner (using a Schur complement method) and divergence penalization for the Oseen problem (Moulin, Jolivet, Marquet 2019)

One can also use approximate algebraic methods on the submatrices A_i: incomplete LU, MUMPS solver, a fixed number of iterations of CG or GMRES, etc...

Domain decomposition methods

To summarize:

- Assemble matrices in parallel

Domain decomposition methods

To summarize:

- Assemble matrices in parallel
- Use an iterative method (CG or GMRES) and a block preconditioner for the distributed matrix A

Domain decomposition methods

To summarize:

- Assemble matrices in parallel
- Use an iterative method (CG or GMRES) and a block preconditioner for the distributed matrix A
- Use factorization or iterative methods with adapted preconditioners for the block matrices A_i.

Domain decomposition methods

To summarize:

- Assemble matrices in parallel
- Use an iterative method (CG or GMRES) and a block preconditioner for the distributed matrix A
- Use factorization or iterative methods with adapted preconditioners for the block matrices A_i.
- with a “good” preconditioner, global iteration scheme converges in less than approx. 100 iterations.

To summarize:

- Assemble matrices in parallel
- Use an iterative method (CG or GMRES) and a block preconditioner for the distributed matrix A
- Use factorization or iterative methods with adapted preconditioners for the block matrices A_i.
- with a “good” preconditioner, global iteration scheme converges in less than approx. 100 iterations.
- The solution will be known in parallel $\rightarrow$ reconstruct it on the global mesh (use partition of unity) or assemble quantities of interest in paralellel.

Domain decomposition methods

To summarize:

- Assemble matrices in parallel
- Use an iterative method (CG or GMRES) and a block preconditioner for the distributed matrix A
- Use factorization or iterative methods with adapted preconditioners for the block matrices A_i.
- with a “good” preconditioner, global iteration scheme converges in less than approx. 100 iterations.
- The solution will be known in parallel $\rightarrow$ reconstruct it on the global mesh (use partition of unity) or assemble quantities of interest in paralellel.

To summarize:

- Assemble matrices in parallel
- Use an iterative method (CG or GMRES) and a block preconditioner for the distributed matrix A
- Use factorization or iterative methods with adapted preconditioners for the block matrices A_i.
- with a “good” preconditioner, global iteration scheme converges in less than approx. 100 iterations.
- The solution will be known in parallel $\rightarrow$ reconstruct it on the global mesh (use partition of unity) or assemble quantities of interest in paralellel.

In FreeFEM, these operations can be achieved rather easily with:

1. PETSc (Portable, Extensible Toolkit for Scientific Computation): flexible and very powerful library for solving FEM problem

To summarize:

- Assemble matrices in parallel
- Use an iterative method (CG or GMRES) and a block preconditioner for the distributed matrix A
- Use factorization or iterative methods with adapted preconditioners for the block matrices A_i.
- with a "good" preconditioner, global iteration scheme converges in less than approx. 100 iterations.
- The solution will be known in parallel $\rightarrow$ reconstruct it on the global mesh (use partition of unity) or assemble quantities of interest in paralellel.

In FreeFEM, these operations can be achieved rather easily with:

1. PETSc (Portable, Extensible Toolkit for Scientific Computation): flexible and very powerful library for solving FEM problem
2. the interface FreeFEM/PETSc written by Pierre Jolivet which allows to perform all the domain decomposition and preconditioning with minimum knowledge.