Introduction to FreeFEM

with an emphasis on parallel computing

Pierre Jolivet

`https://joliv.et/FreeFem-tutorial`

Browser shortcuts:

- Ctrl + f fit to width/height
- Ctrl + g go to a page #
- Ctrl + PgDwn next page
- Ctrl + PgUp previous page

v2023.08

Introduction

Acknowledgements

- University of Tsukuba, Japan
- Institute of Mathematics, University of Seville, Spain
- CMAP, École Polytechnique, France
- Cybermedia Center, Osaka University, Japan
- Musashino University, Japan

Prerequisites (the more the better)

- FreeFEM [Hecht 2012] `https://github.com/FreeFem/FreeFem-sources`
- Gmsh [Geuzaine and Remacle 2009] `http://gmsh.info`
- ParaView `https://www.paraview.org`
- MPI `https://www.mpi-forum.org`
- HPDDM [Jolivet, Hecht, Nataf, et al. 2013] `https://github.com/hpddm/hpddm`
- PETSc [Balay et al. 1997] `https://gitlab.com/petsc/petsc`
- SLEPc [Hernandez et al. 2005] `https://gitlab.com/slepc/slepc`

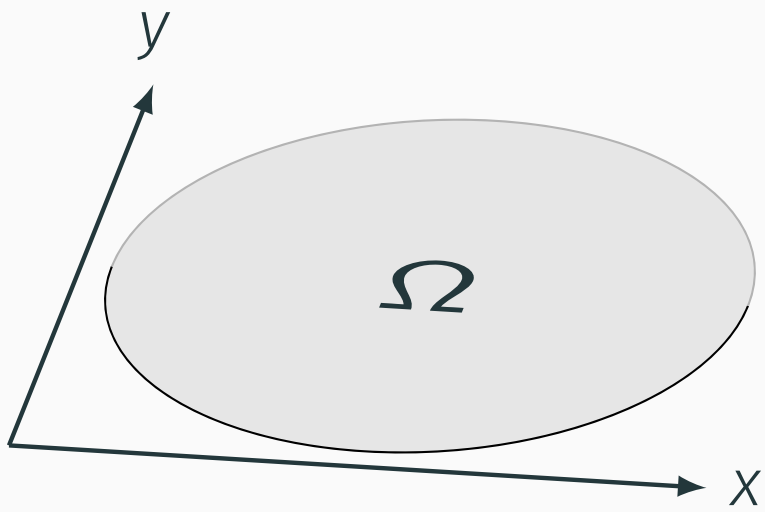
Program of the lecture

1. Introduction
2. Finite elements
3. FreeFEM
4. Shared-memory parallelism
5. Distributed-memory parallelism
6. Overlapping Schwarz methods
7. Substructuring methods
8. PETSc
9. SLEPc
10. Applications

Finite elements

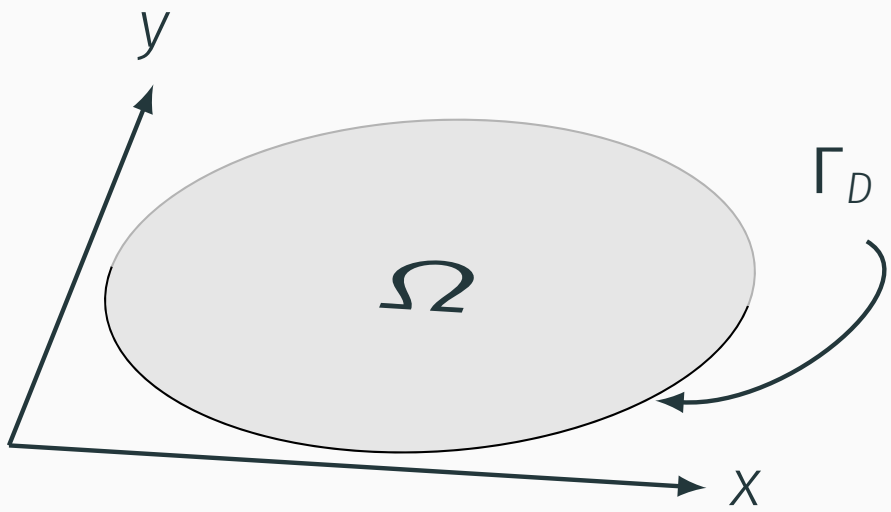
Model problem

$$-\Delta u = f \qquad \text{in } \Omega$$



Model problem

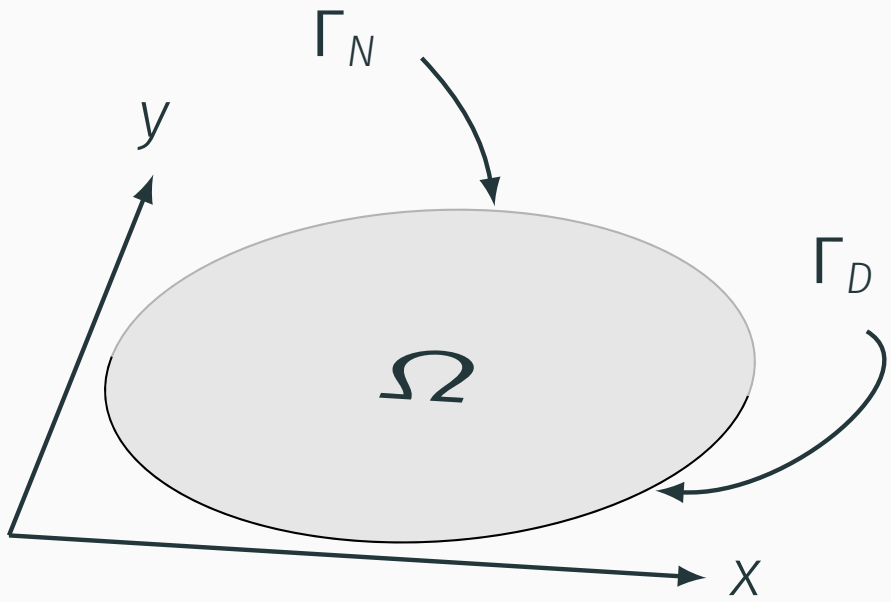
$$-\Delta u = f \qquad \text{in } \Omega$$
$$u = g_D \qquad \text{on } \Gamma_D$$



- essential boundary conditions

Model problem

$$
\begin{aligned}
-\Delta u &= f && \text{in } \Omega \\
u &= g_D && \text{on } \Gamma_D \\
\partial_n u &= g_N && \text{on } \Gamma_N
\end{aligned}
$$



- essential boundary conditions
- natural boundary conditions

Variational formulation

Green's theorem

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

$$\int_\Omega \nabla u \cdot \nabla v = \int_\Omega f\, v + \int_{\Gamma_N} g_N\, v, \quad \forall v \in H^1_{\Gamma_D}(\Omega)$$

$$u = g_D \quad \text{on } \Gamma_D$$

Variational formulation

Green's theorem

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

$$\int_\Omega \nabla u \cdot \nabla v = \int_\Omega f\, v + \int_{\Gamma_N} g_N\, v, \quad \forall v \in H^1_{\Gamma_D}(\Omega)$$

$$u = g_D \quad \text{on } \Gamma_D$$

- unknown function u
- test function v

Mesh and finite elements

- Ω discretized by Ω_h with n_h elements
- u discretized by $u_h = \sum_{i=1}^{N_h} u_h(i)\varphi_i$, smooth w.r.t. Ω_h

Mesh and finite elements

- Ω discretized by Ω_h with n_h elements
- u discretized by $u_h = \sum_{i=1}^{N_h} u_h(i)\varphi_i$, smooth w.r.t. Ω_h

$$u_h \in H^1(\Omega_h) \iff u_h \text{ is continuous}$$

Mesh and finite elements

- Ω discretized by Ω_h with n_h elements
- u discretized by $u_h = \sum_{i=1}^{N_h} u_h(i)\varphi_i$, smooth w.r.t. Ω_h

$$u_h \in H^1(\Omega_h) \iff u_h \text{ is continuous}$$

- basis functions $\{\varphi_i\}_{i=1}^{N_h}$

Assembly procedures

Matrix

$$\forall (i,j) \in [\![1; N_h]\!]^2, A_{ij} = \int_{\Omega_h} \nabla \varphi_j \cdot \nabla \varphi_i$$

$\Longrightarrow$ only integrate in the intersection of both supports

Assembly procedures

Matrix

$$\forall (i,j) \in [\![1; N_h]\!]^2, A_{ij} = \int_{\Omega_h} \nabla \varphi_j \cdot \nabla \varphi_i$$

$\Longrightarrow$ only integrate in the intersection of both supports

Right-hand side

$$\forall i \in [\![1; N_h]\!], b_i = \int_{\Omega_h} f\, \varphi_i + \int_{\Gamma_N^h} g_N\, \varphi_i$$

$\Longrightarrow$ numerical integration using quadrature rules

Essential boundary conditions

G_D subset of unknowns associated to Dirichlet BC

- nonsymmetric elimination

$$Ax = \begin{bmatrix} A_{\overline{G_D}\overline{G_D}} & A_{\overline{G_D}G_D} \\ 0 & I_{G_DG_D} \end{bmatrix} \begin{bmatrix} x_{\overline{G_D}} \\ x_{G_D} \end{bmatrix} = \begin{bmatrix} b_{\overline{G_D}} \\ g_D \end{bmatrix}$$

Essential boundary conditions

G_D subset of unknowns associated to Dirichlet BC

- nonsymmetric elimination

$$Ax = \begin{bmatrix} A_{\overline{G_D}\overline{G_D}} & A_{\overline{G_D}G_D} \\ 0 & I_{G_DG_D} \end{bmatrix} \begin{bmatrix} x_{\overline{G_D}} \\ x_{G_D} \end{bmatrix} = \begin{bmatrix} b_{\overline{G_D}} \\ g_D \end{bmatrix}$$

- symmetric elimination

$$Ax = \begin{bmatrix} A_{\overline{G_D}\overline{G_D}} & 0 \\ 0 & I_{G_DG_D} \end{bmatrix} \begin{bmatrix} x_{\overline{G_D}} \\ x_{G_D} \end{bmatrix} = \begin{bmatrix} b_{\overline{G_D}} - A_{\overline{G_D}G_D} g_D \\ g_D \end{bmatrix}$$

Essential boundary conditions

G_D subset of unknowns associated to Dirichlet BC

- nonsymmetric elimination

$$Ax = \begin{bmatrix} A_{\overline{G_D}\overline{G_D}} & A_{\overline{G_D}G_D} \\ 0 & I_{G_DG_D} \end{bmatrix} \begin{bmatrix} x_{\overline{G_D}} \\ x_{G_D} \end{bmatrix} = \begin{bmatrix} b_{\overline{G_D}} \\ g_D \end{bmatrix}$$

- symmetric elimination

$$Ax = \begin{bmatrix} A_{\overline{G_D}\overline{G_D}} & 0 \\ 0 & I_{G_DG_D} \end{bmatrix} \begin{bmatrix} x_{\overline{G_D}} \\ x_{G_D} \end{bmatrix} = \begin{bmatrix} b_{\overline{G_D}} - A_{\overline{G_D}G_D} g_D \\ g_D \end{bmatrix}$$

- penalization

$$Ax = \begin{bmatrix} A_{\overline{G_D}\overline{G_D}} & A_{\overline{G_D}G_D} \\ A_{G_D\overline{G_D}} & A_{G_DG_D} + 10^{30} I_{G_DG_D} \end{bmatrix} \begin{bmatrix} x_{\overline{G_D}} \\ x_{G_D} \end{bmatrix} = \begin{bmatrix} b_{\overline{G_D}} \\ 10^{30} g_D \end{bmatrix}$$

FreeFEM

History

1987 MacFem/PCFem by Pironneau

1992 FreeFem by Pironneau, Bernardi, Hecht, and Prud'homme

1996 FreeFem+ by Pironneau, Bernardi, and Hecht

1998 FreeFem++ by Hecht, Pironneau, and Ohtsuka

2008 version 3

2014 version 3.34 with distributed-memory parallelism

2018 moving to GitHub

2019 version 4, rebranded as FreeFEM

2023 version 4.13, composite spaces

Installation

- packages available for most OSes
  - Windows
  - macOS
  - Debian
- compilation from the sources for more flexibility
  - custom PETSc/SLEPc installation
  - develop branch
- on the cloud
  - `https://www.rescale.com`
  - `http://qarnot.com`

Installation

- packages available for most OSes
  - Windows
  - macOS
  - Debian
- compilation from the sources for more flexibility
  - custom PETSc/SLEPc installation
  - develop branch
- on the cloud
  - `https://www.rescale.com`
  - `http://qarnot.com`

$\Longrightarrow$ `https://community.freefem.org`

Standard compilation process I

- remove any previous instance of FreeFEM
- install gcc/clang and gfortran
- make sure you have a working MPI implementation

Standard compilation process II

```
> git clone https://github.com/FreeFem/FreeFem-sources
> cd FreeFem-sources
> git checkout develop
> autoreconf -i
> ./configure --enable-download --without-hdf5
  --prefix=${PWD}
> cd 3rdparty/ff-petsc
> make petsc-slepc
> cd -
> ./reconfigure
> make
```

Binaries

- FreeFem++
- FreeFem++-mpi
- ffglut
- ffmedit
- ff-c++

Basic parameters

- `-ns` script not printed
- `-nw`/`-wg` graphical output deactivated
- `-v` 0 level of verbosity

mesh AND mesh3

[EXAMPLE2.EDP]

Structured meshes

- in 2D, `square`
- in 3D, `cube`

Unstructured meshes

- `buildmesh`
- interface with Gmsh, TetGen [Si 2013], and MMG

Online visualization

- mostly for debugging purposes
- prefer `medit` over `plot` in 3D

`fespace` [example3.edp]

- formal relationship between a `mesh` and a FE
- one accessible member `.ndof`
- may be used to define finite element functions

fespace

[example3.edp]

- formal relationship between a `mesh` and a FE
- one accessible member `.ndof`
- may be used to define finite element functions

varf AND on

[example4.edp]

- bilinear forms
- linear forms
- boundary conditions
- dynamic definition
- instantiated to assemble matrices or vectors
- `qforder` to change integration rules

`varf` AND on

[EXAMPLE4.EDP]

- bilinear forms
- linear forms
- boundary conditions
- dynamic definition
- instantiated to assemble matrices or vectors
- `qforder` to change integration rules

`matrix` AND `set` [EXAMPLE5.EDP]

- assemble a `varf` using a pair of `fespace`s
- variable `sym` to assemble the upper triangular part
- matrix–vector products
- specify a solver for linear systems with `set`

`matrix` AND `set` [EXAMPLE5.EDP]

- assemble a `varf` using a pair of `fespace`s
- variable `sym` to assemble the upper triangular part
- matrix–vector products
- specify a solver for linear systems with `set`

Essential boundary conditions

- `tgv` = -1 for nonsymmetric elimination
- `tgv` = -2 for symm. elim. (careful about the RHS)
- `tgv` = 1e+30 for penalization

`real[int]` AND OTHER ARRAYS [EXAMPLE6.EDP]

- `real[int,int]`
- `real[int][int]`
- `complex[int]`
- `matrix[int]`, `string[int]`
- formal array of finite element functions

Common members and methods

- `.n` and `.m`
- `.resize`
- =, +=, /=

Additional plugins [example7.edp]

- keyword `load`
- `load "something"` $\Longrightarrow$ ff-c++ -auto something.cpp
- FreeFEM objects manipulated in C++/Fortran

func

[example7.edp]

- user-defined functions
- may be passed to external codes
- useful for matrix-free computations
- `LinearCG`, `EigenValue`

macro

[example7.edp]

- evaluated when parsing input files
- defined on the command line `-DmacroName=value`
- conditional statements using `IFMACRO`

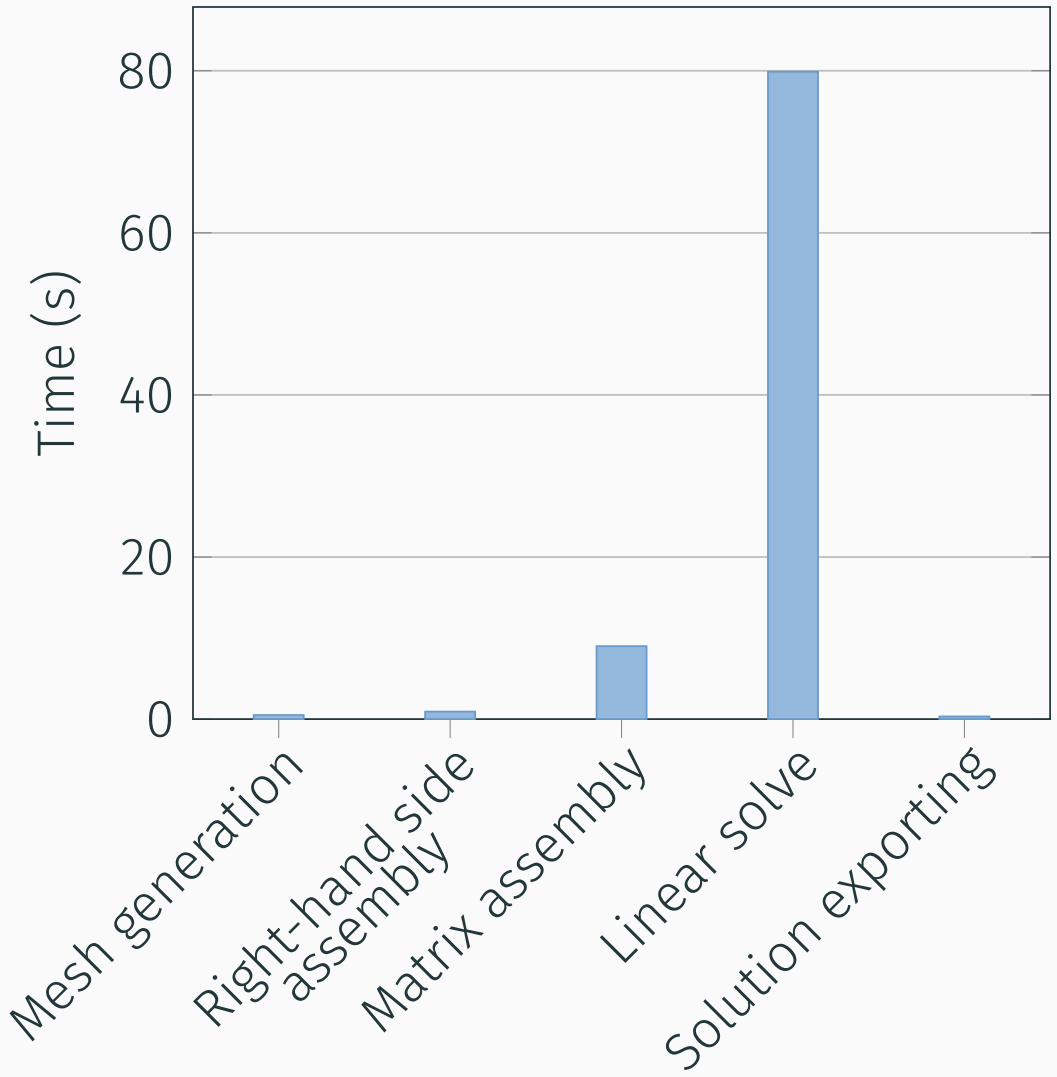
Submesh

[example8.edp]

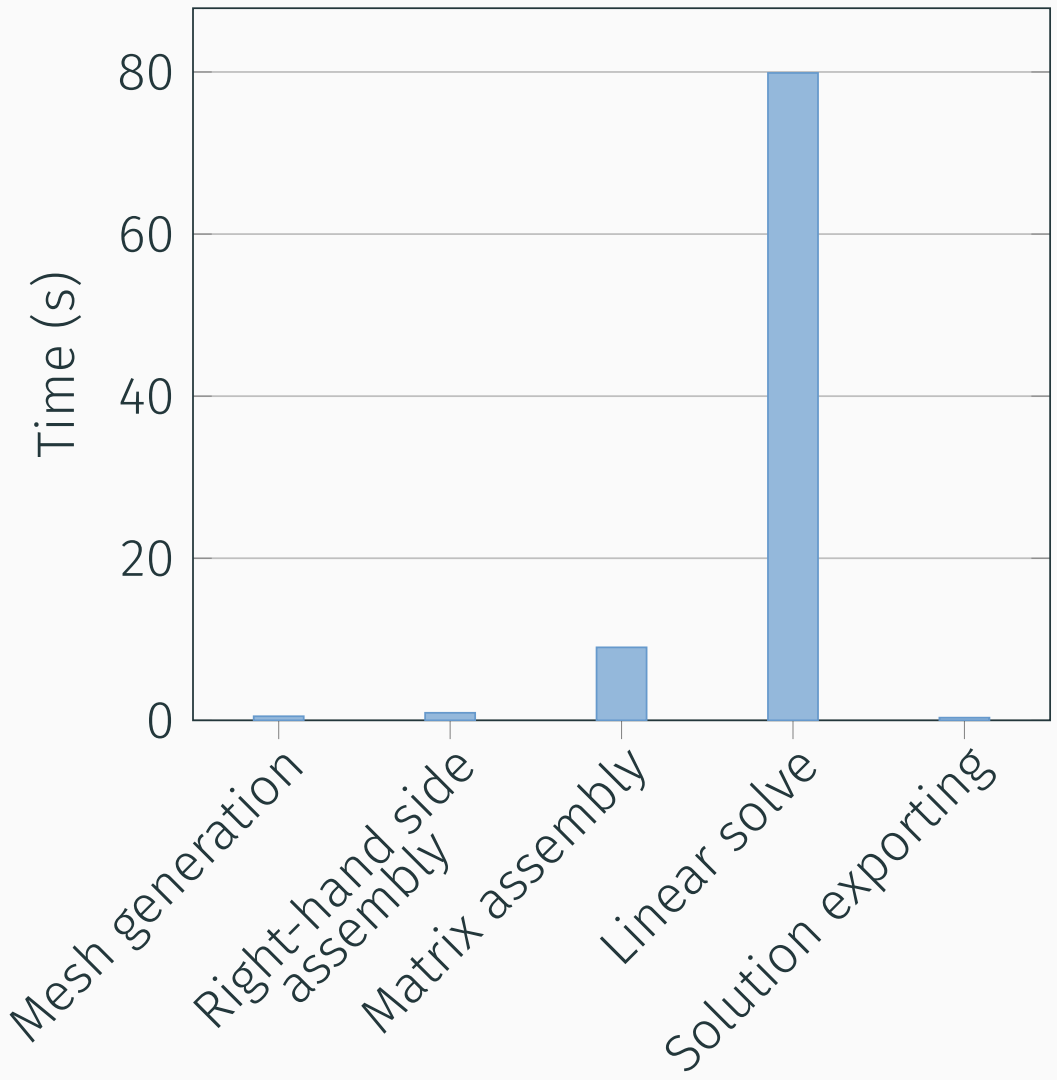
- `trunc` optional parameter `new2old`
- + `restrict` to go from one `fespace` to another
- useful to avoid (costly) interpolations
- meshes in multiphysics, e.g., solid + fluid domains

Shared-memory parallelism

Motivation


Time (s)
0
20
40
60
80
Mesh generation
Right-hand side assembly
Matrix assembly
Linear solve
Solution exporting

Motivation



Parallelism is key for performance

Impact on the finite element method [example1.edp]

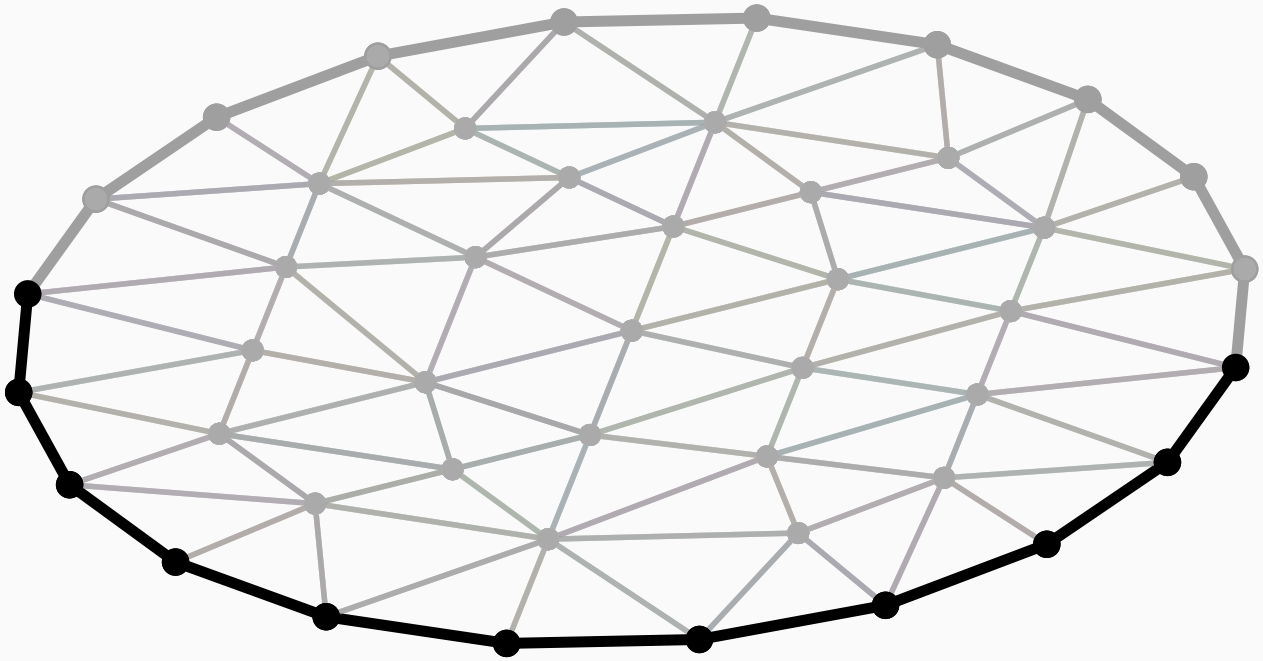
Shared-memory parallelism (pthreads, OpenMP)

- global address space
- minimize critical sections
- mostly suited to small-scale architectures

Impact on the finite element method [example1.edp]

Shared-memory parallelism (pthreads, OpenMP)

- global address space
- minimize critical sections
- mostly suited to small-scale architectures

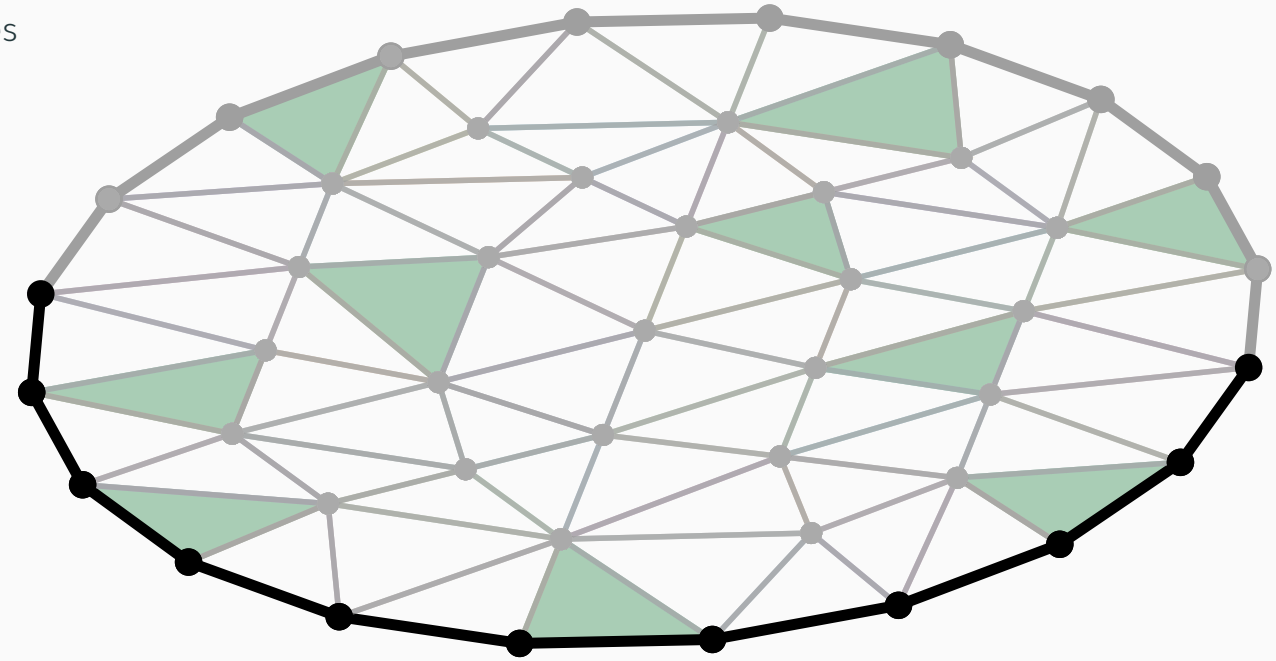


Impact on the finite element method [example1.edp]

Shared-memory parallelism (pthreads, OpenMP)

- global address space
- minimize critical sections
- mostly suited to small-scale architectures

Color #1: 10 triangles

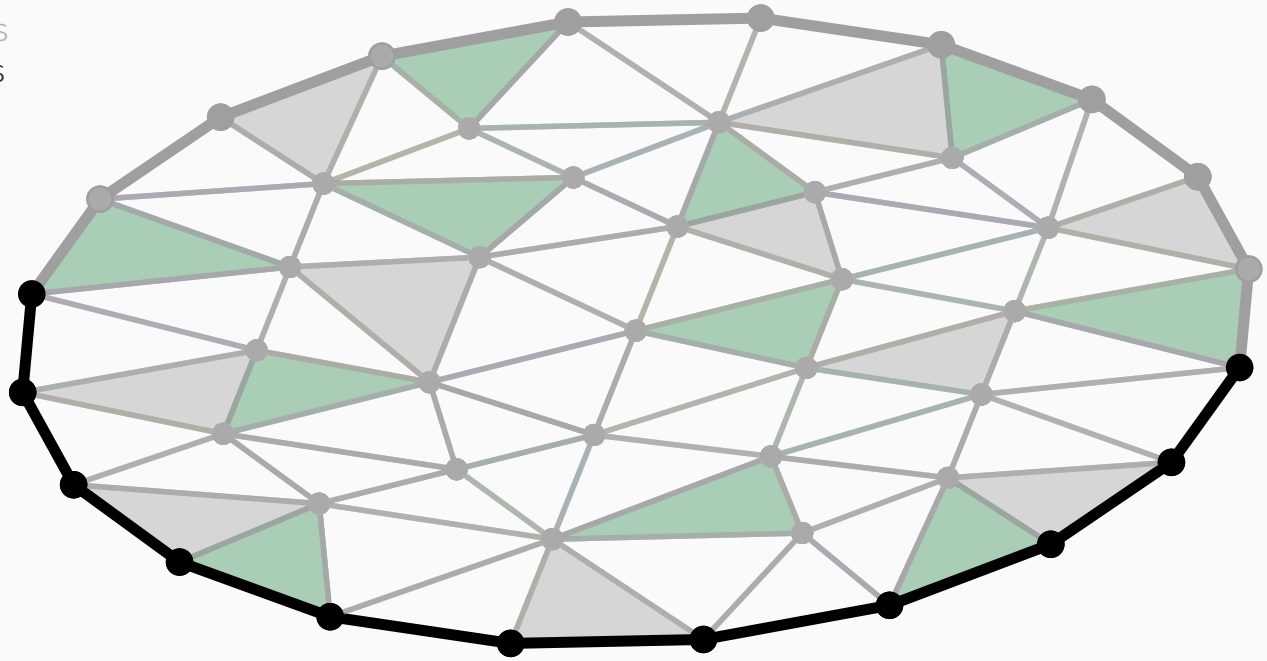


Impact on the finite element method [example1.edp]

Shared-memory parallelism (pthreads, OpenMP)

- global address space
- minimize critical sections
- mostly suited to small-scale architectures

Color #1: 10 triangles
Color #2: 11 triangles

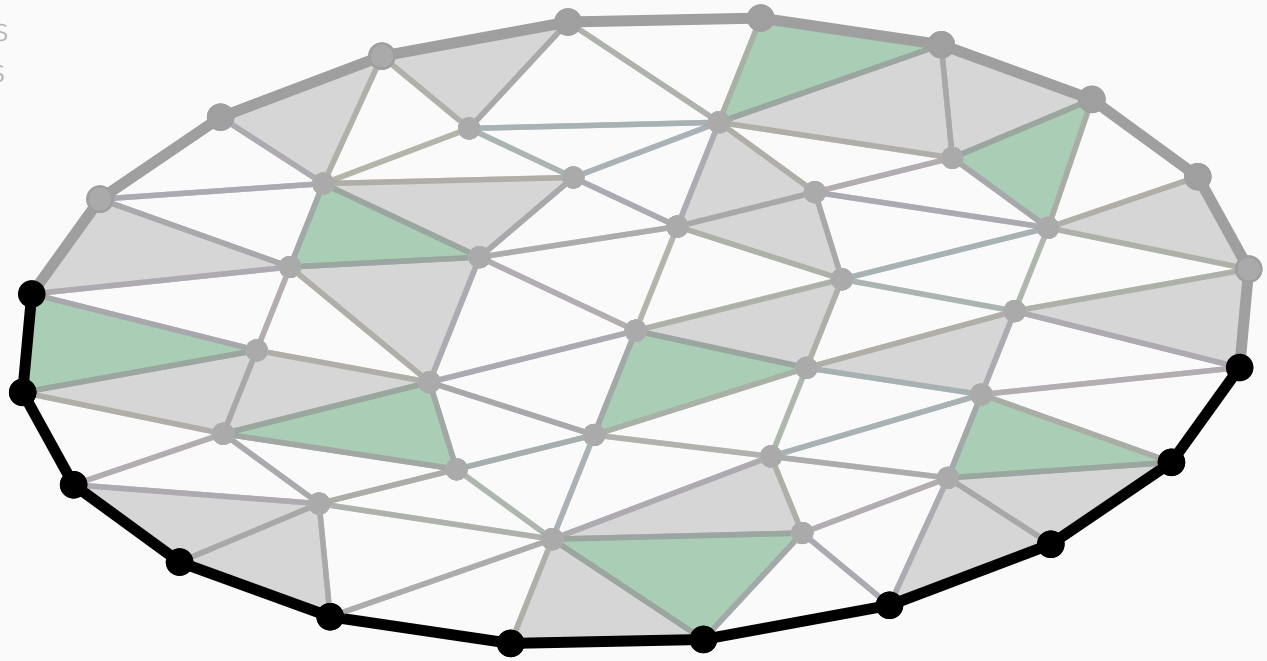


Impact on the finite element method [example1.edp]

Shared-memory parallelism (pthreads, OpenMP)

- global address space
- minimize critical sections
- mostly suited to small-scale architectures

Color #1: 10 triangles
Color #2: 11 triangles
Color #3: 8 triangles

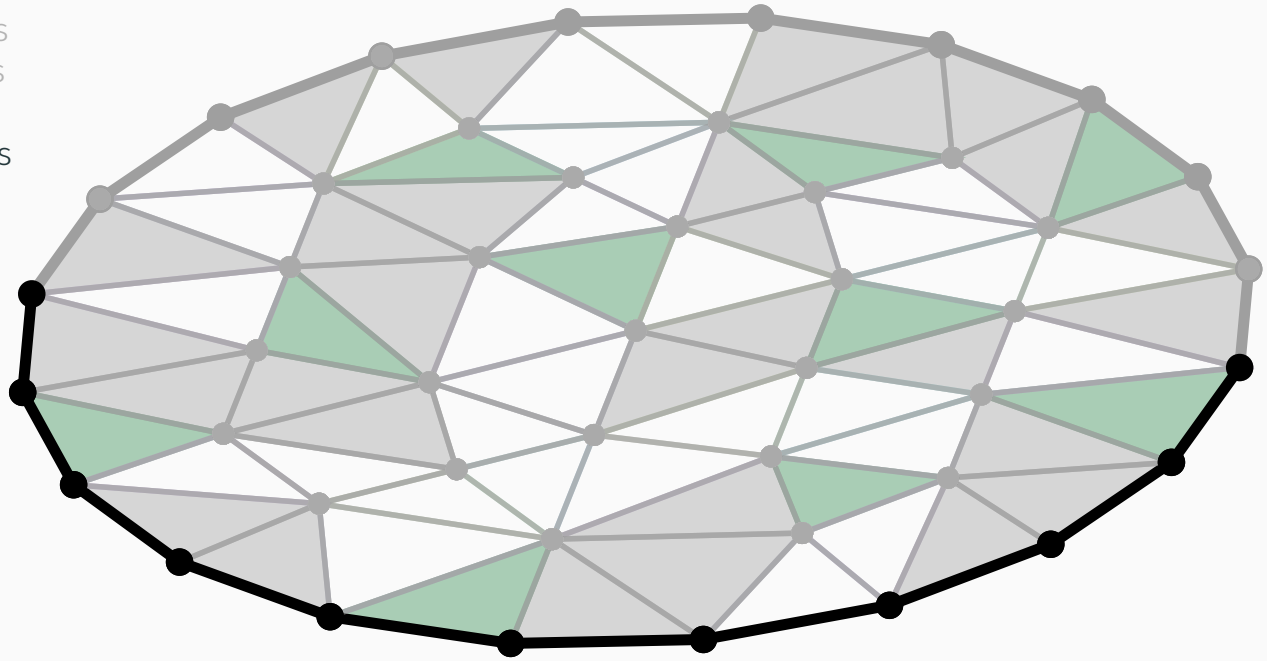


Impact on the finite element method [example1.edp]

Shared-memory parallelism (pthreads, OpenMP)

- global address space
- minimize critical sections
- mostly suited to small-scale architectures

Color #1: 10 triangles
Color #2: 11 triangles
Color #3: 8 triangles
Color #4: 10 triangles

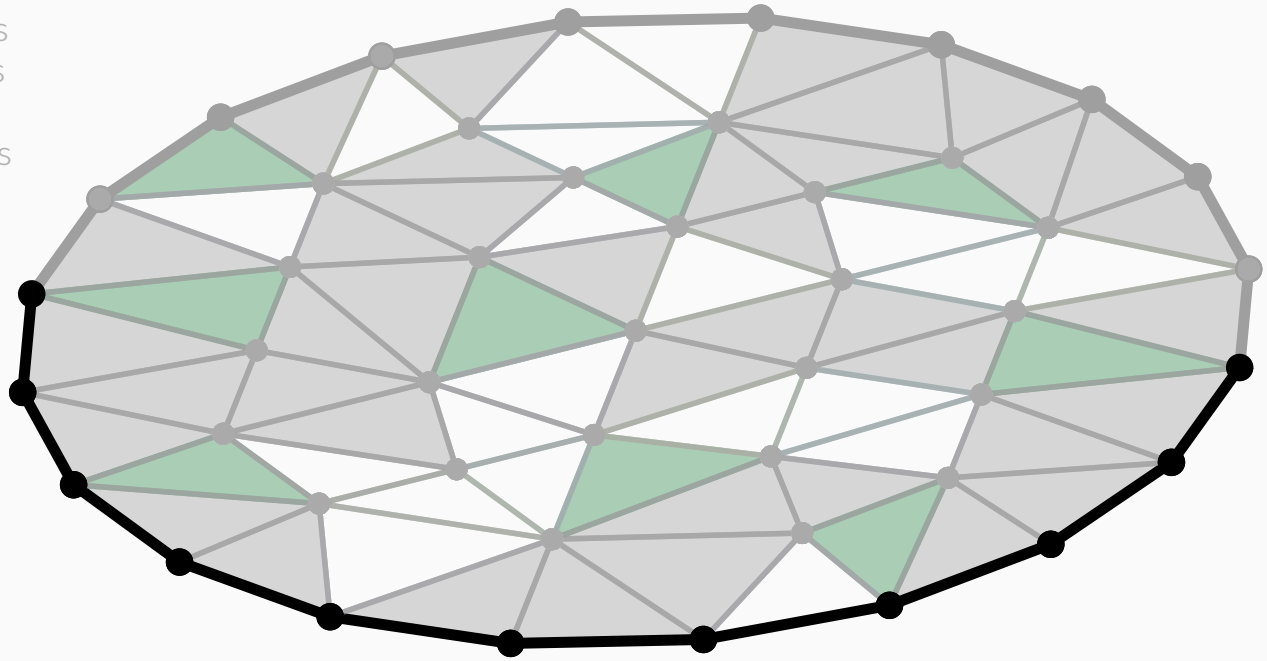


Impact on the finite element method [example1.edp]

Shared-memory parallelism (pthreads, OpenMP)

- global address space
- minimize critical sections
- mostly suited to small-scale architectures

Color #1: 10 triangles
Color #2: 11 triangles
Color #3: 8 triangles
Color #4: 10 triangles
Color #5: 9 triangles

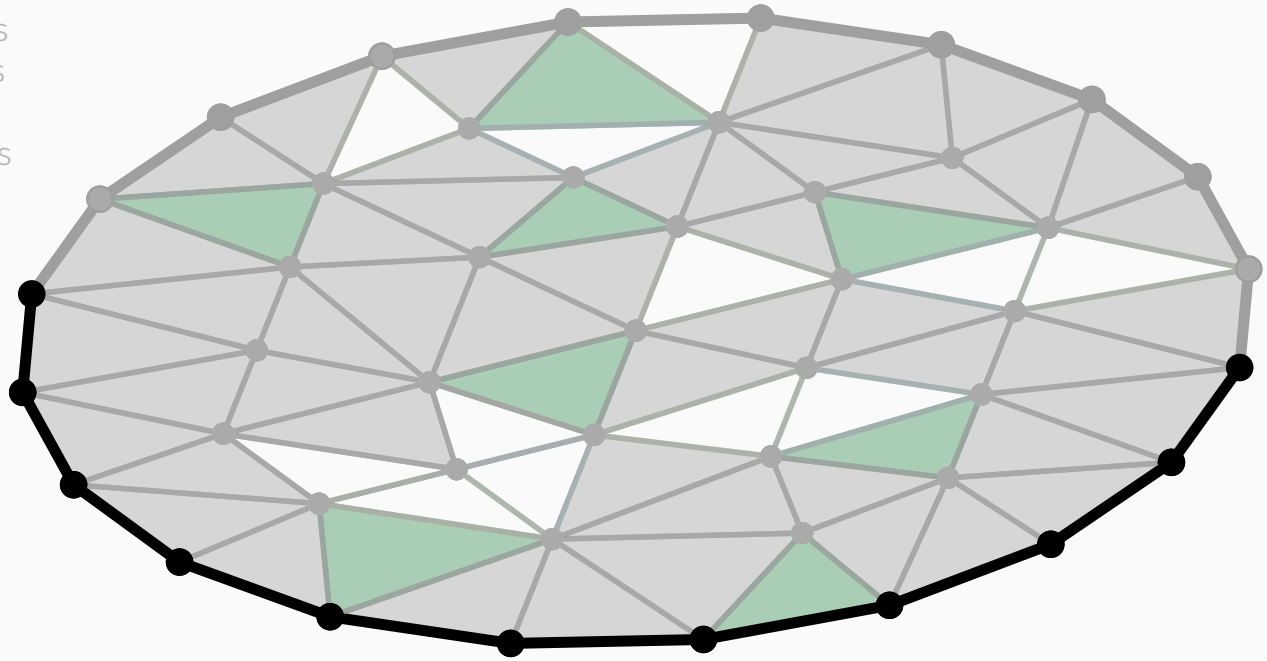


Impact on the finite element method [example1.edp]

Shared-memory parallelism (pthreads, OpenMP)

- global address space
- minimize critical sections
- mostly suited to small-scale architectures

Color #1: 10 triangles
Color #2: 11 triangles
Color #3: 8 triangles
Color #4: 10 triangles
Color #5: 9 triangles
Color #6: 8 triangles

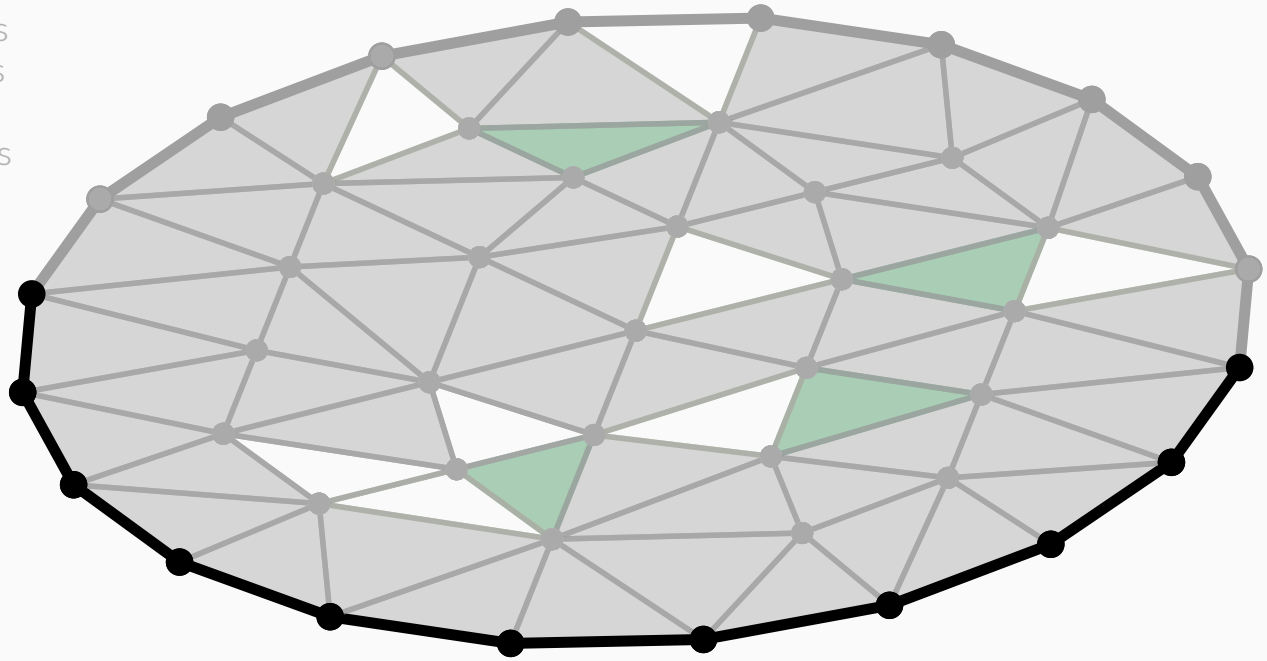


Impact on the finite element method [example1.edp]

Shared-memory parallelism (pthreads, OpenMP)

- global address space
- minimize critical sections
- mostly suited to small-scale architectures

Color #1: 10 triangles
Color #2: 11 triangles
Color #3: 8 triangles
Color #4: 10 triangles
Color #5: 9 triangles
Color #6: 8 triangles
Color #7: 4 triangles

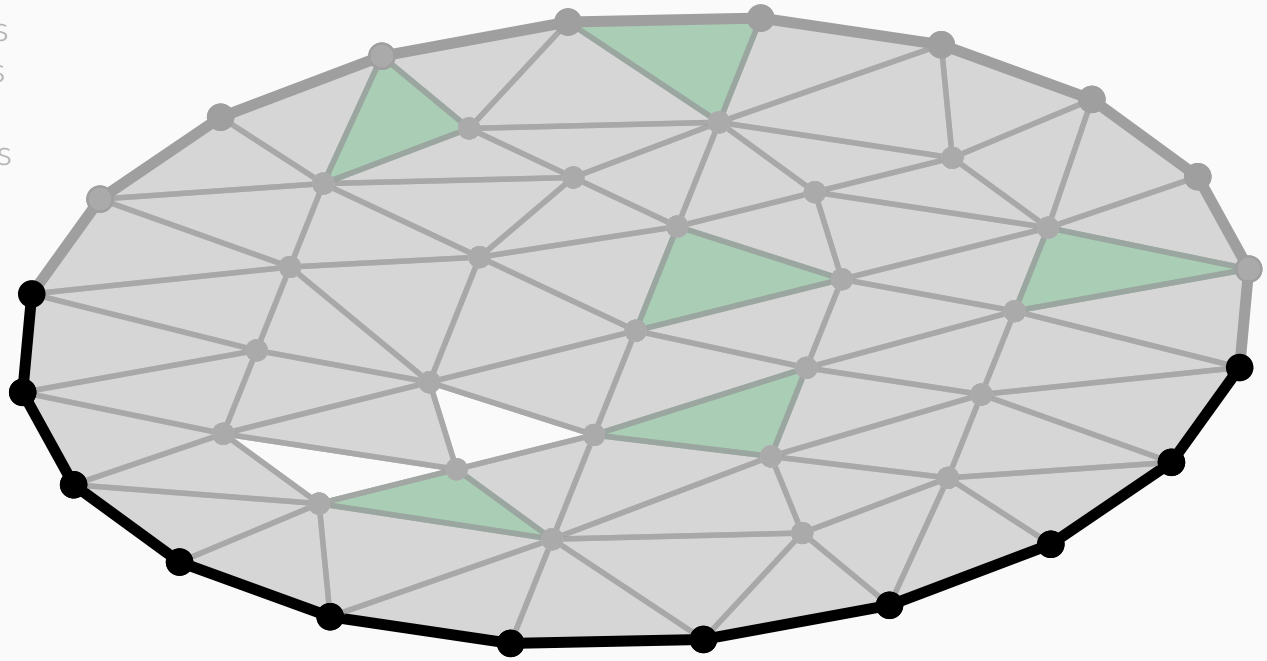


Impact on the finite element method [example1.edp]

Shared-memory parallelism (pthreads, OpenMP)

- global address space
- minimize critical sections
- mostly suited to small-scale architectures

Color #1: 10 triangles
Color #2: 11 triangles
Color #3: 8 triangles
Color #4: 10 triangles
Color #5: 9 triangles
Color #6: 8 triangles
Color #7: 4 triangles
Color #8: 6 triangles

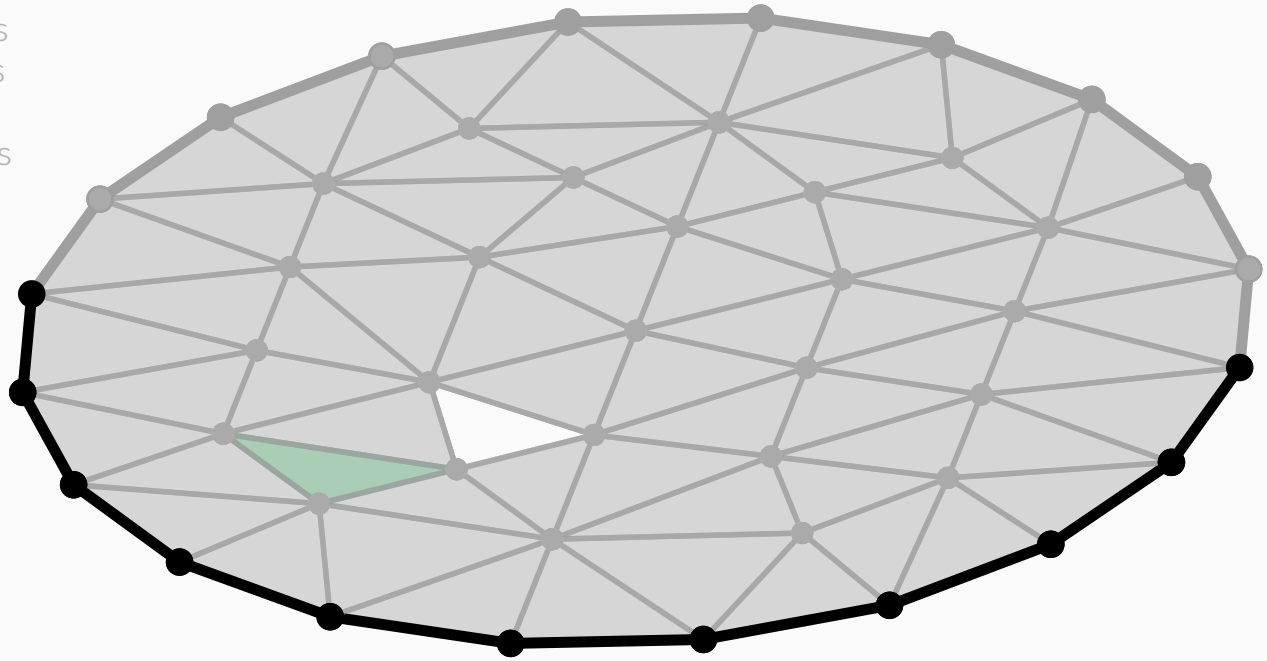


Impact on the finite element method [example1.edp]

Shared-memory parallelism (pthreads, OpenMP)

- global address space
- minimize critical sections
- mostly suited to small-scale architectures

Color #1: 10 triangles
Color #2: 11 triangles
Color #3: 8 triangles
Color #4: 10 triangles
Color #5: 9 triangles
Color #6: 8 triangles
Color #7: 4 triangles
Color #8: 6 triangles
Color #9: 1 triangle

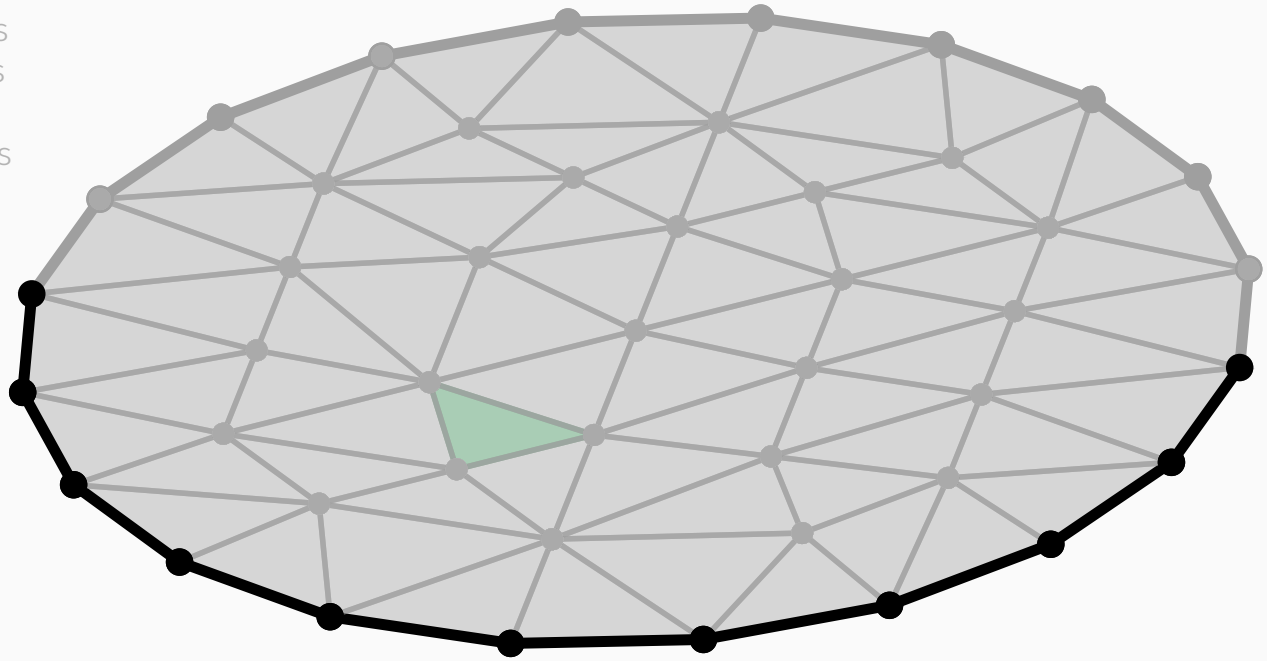


Impact on the finite element method [example1.edp]

Shared-memory parallelism (pthreads, OpenMP)

- global address space
- minimize critical sections
- mostly suited to small-scale architectures

Color #1: 10 triangles
Color #2: 11 triangles
Color #3: 8 triangles
Color #4: 10 triangles
Color #5: 9 triangles
Color #6: 8 triangles
Color #7: 4 triangles
Color #8: 6 triangles
Color #9: 1 triangle
Color #10: 1 triangle

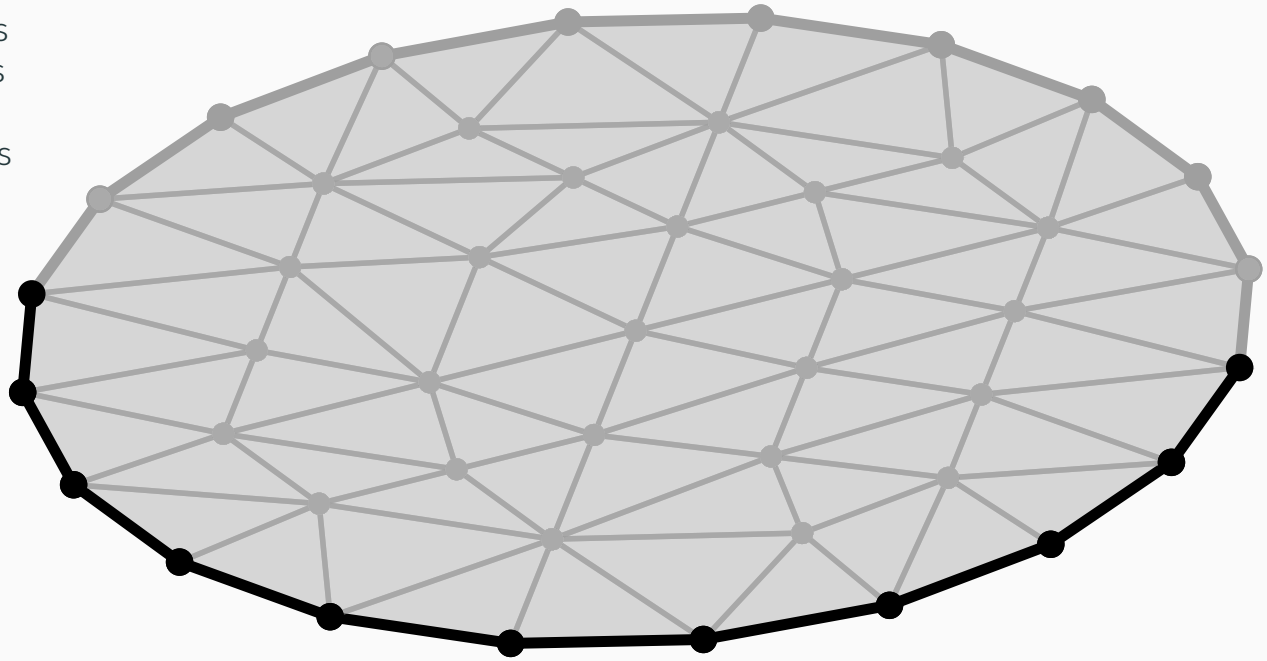


Impact on the finite element method [example1.edp]

Shared-memory parallelism (pthreads, OpenMP)

- global address space
- minimize critical sections
- mostly suited to small-scale architectures

Color #1: 10 triangles
Color #2: 11 triangles
Color #3: 8 triangles
Color #4: 10 triangles
Color #5: 9 triangles
Color #6: 8 triangles
Color #7: 4 triangles
Color #8: 6 triangles
Color #9: 1 triangle
Color #10: 1 triangle



Other kernels

- efficient assembly is hard (especially for low-order FE)
- linear algebra
- exact factorizations (LU or LDL^H)

Direct solvers and BLAS

- three options
  - MKL PARDISO, Intel [example2.edp]
  - Dissection [Suzuki and Roux 2014]
  - MUMPS_seq
- MKL for dense linear algebra [example3.edp]
- be careful with `OMP_NUM_THREADS`

Distributed-memory parallelism

Impact on the finite element method [example1.edp]

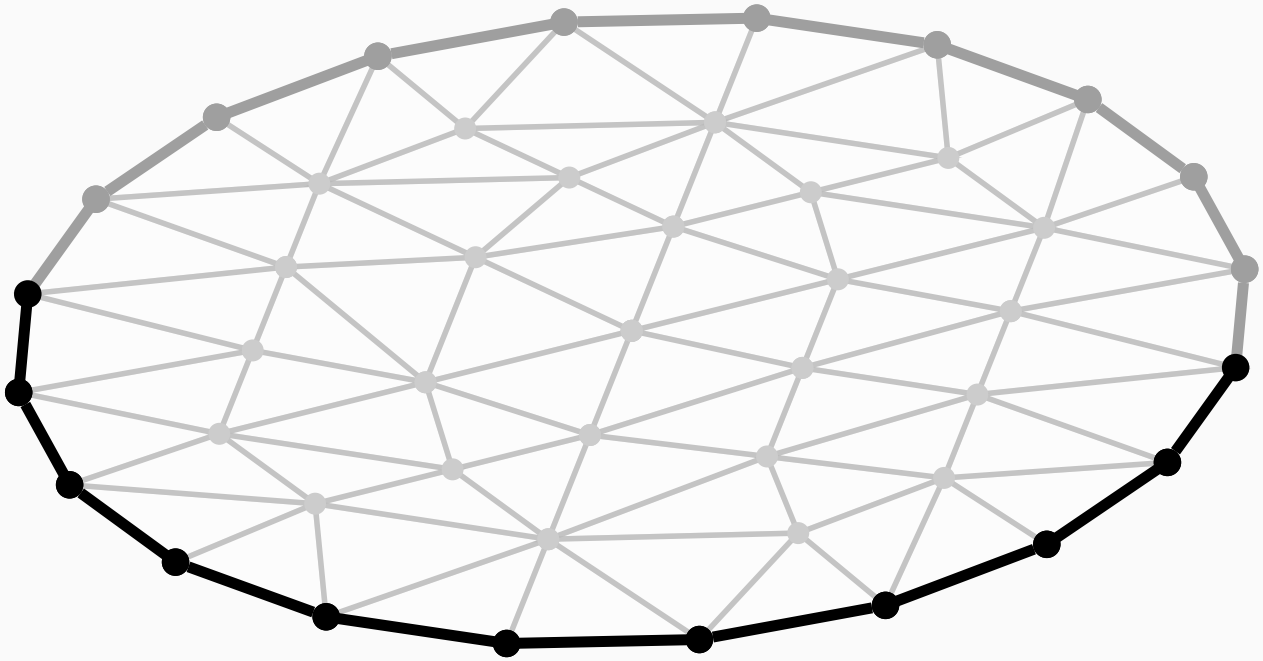
Distributed-memory parallelism (MPI)

- local address space
- distribute data efficiently to minimize communication
- orthogonal to shared-memory parallelism

Impact on the finite element method [example1.edp]

Distributed-memory parallelism (MPI)

- local address space
- distribute data efficiently to minimize communication
- orthogonal to shared-memory parallelism

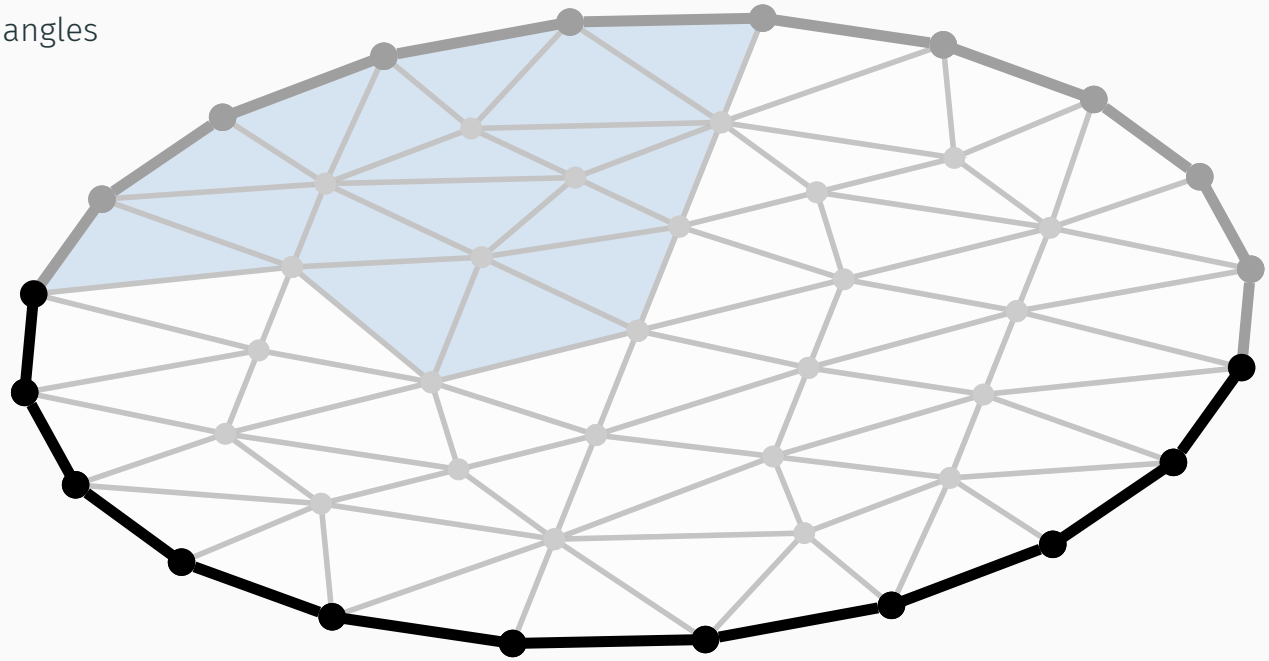


Impact on the finite element method [example1.edp]

Distributed-memory parallelism (MPI)

- local address space
- distribute data efficiently to minimize communication
- orthogonal to shared-memory parallelism

Subdomain #1: 17 triangles

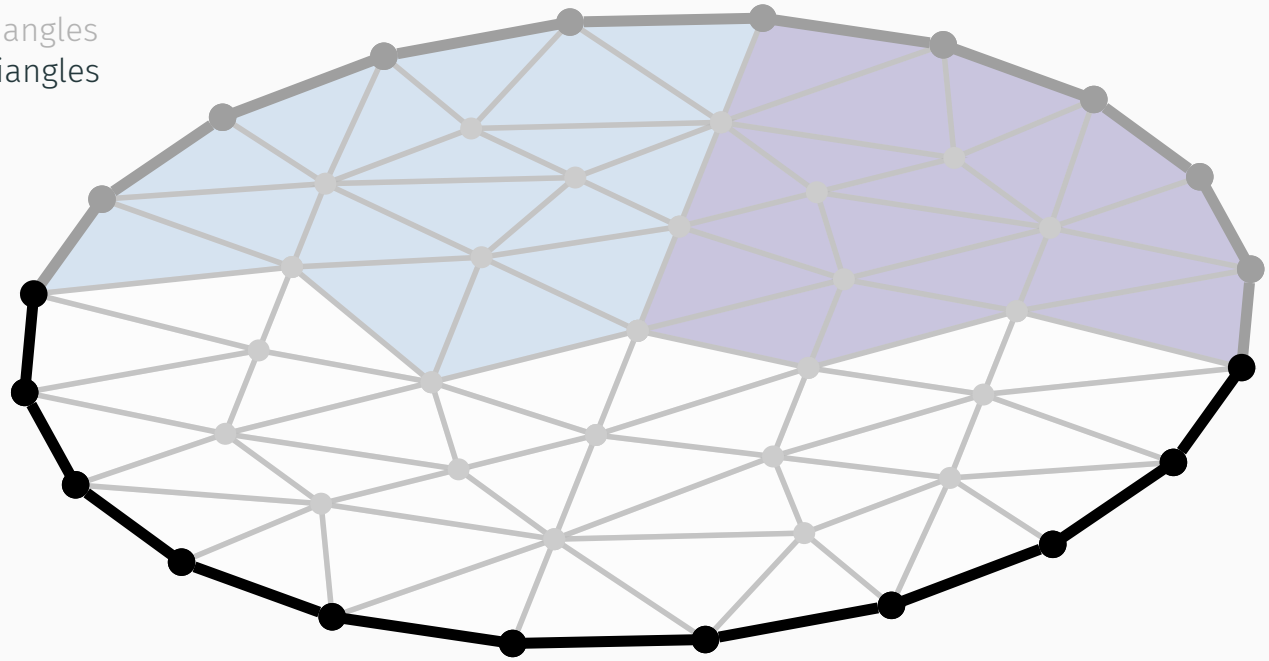


Impact on the finite element method [example1.edp]

Distributed-memory parallelism (MPI)

- local address space
- distribute data efficiently to minimize communication
- orthogonal to shared-memory parallelism

Subdomain #1: 17 triangles
Subdomain #2: 17 triangles

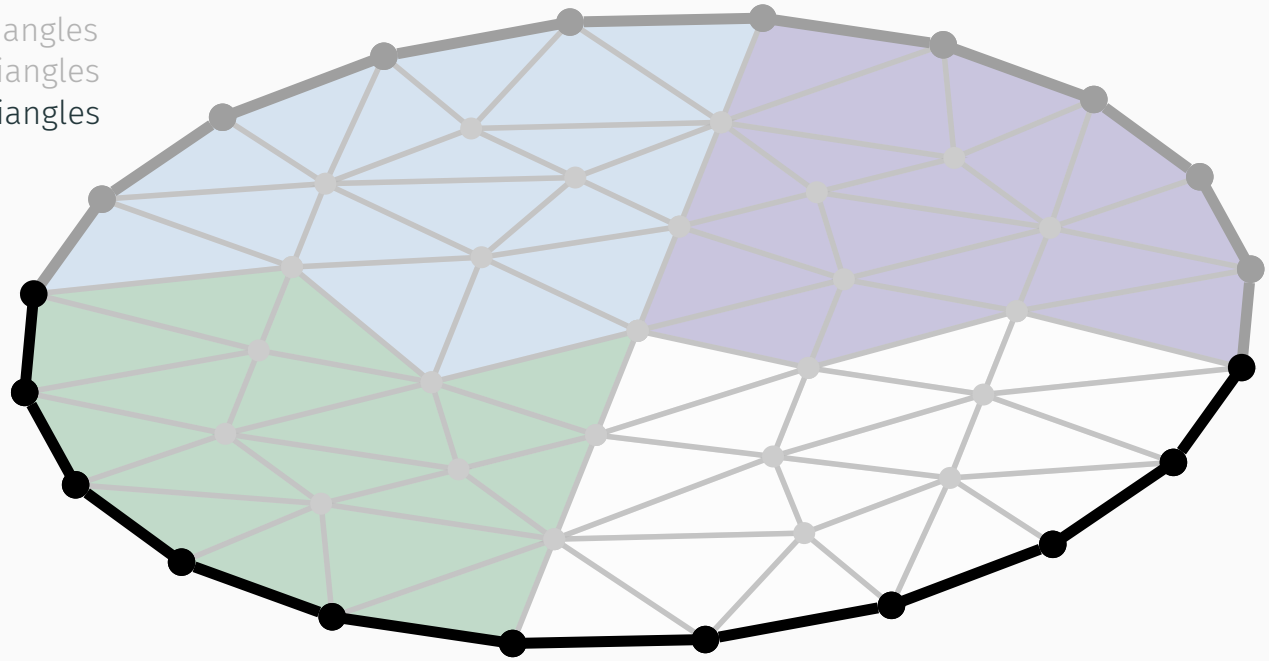


Impact on the finite element method [example1.edp]

Distributed-memory parallelism (MPI)

- local address space
- distribute data efficiently to minimize communication
- orthogonal to shared-memory parallelism

Subdomain #1: 17 triangles
Subdomain #2: 17 triangles
Subdomain #3: 17 triangles

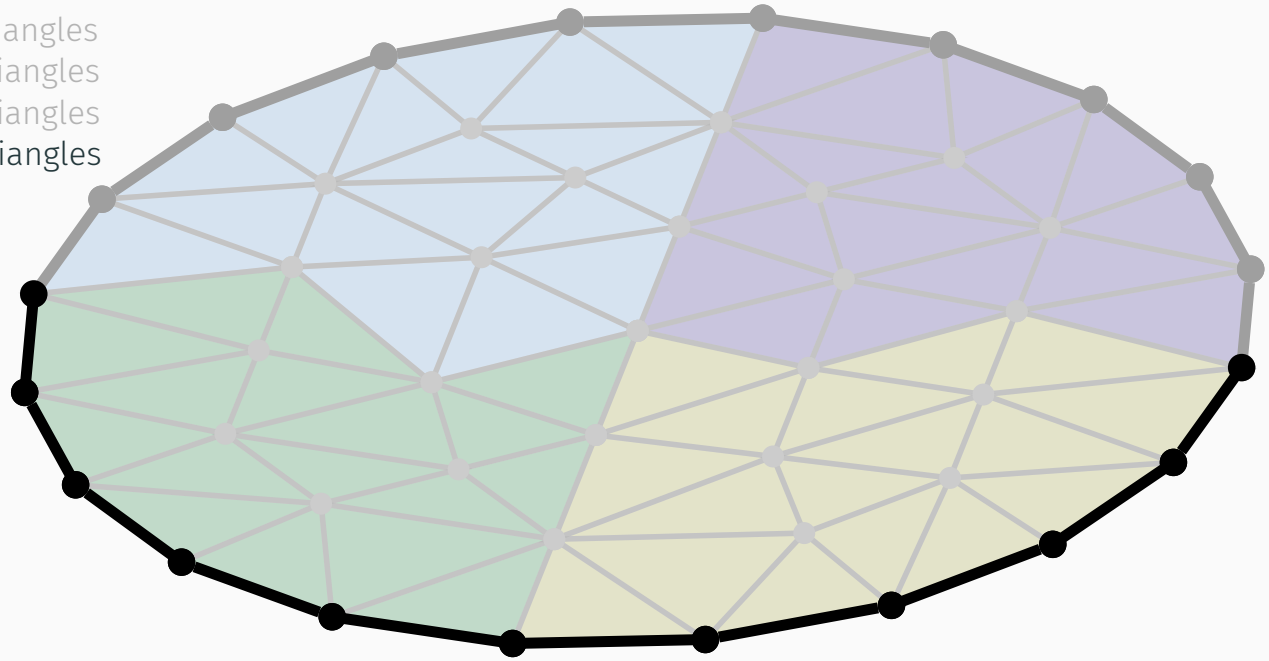


Impact on the finite element method [example1.edp]

Distributed-memory parallelism (MPI)

- local address space
- distribute data efficiently to minimize communication
- orthogonal to shared-memory parallelism

Subdomain #1: 17 triangles
Subdomain #2: 17 triangles
Subdomain #3: 17 triangles
Subdomain #4: 17 triangles

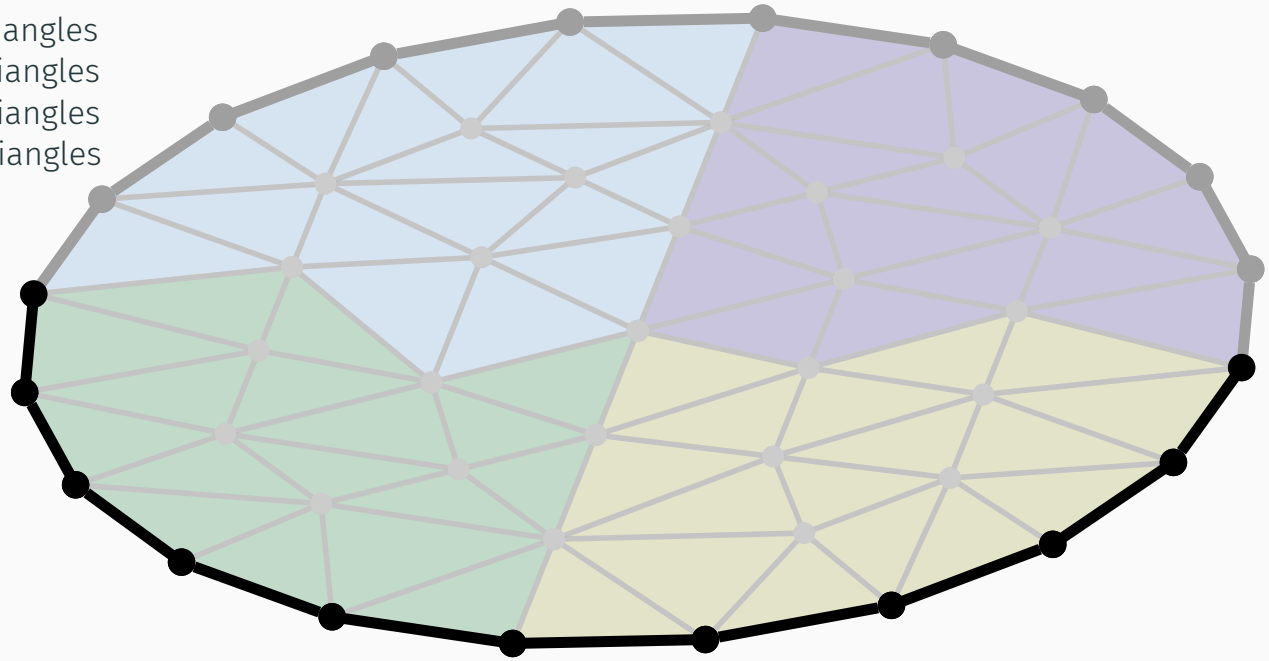


Impact on the finite element method [example1.edp]

Distributed-memory parallelism (MPI)

- local address space
- distribute data efficiently to minimize communication
- orthogonal to shared-memory parallelism

Subdomain #1: 17 triangles
Subdomain #2: 17 triangles
Subdomain #3: 17 triangles
Subdomain #4: 17 triangles



FreeFem++-mpi

- just like FreeFem++, but with MPI
- a friendly approach to message passing (like mpi4py)
- some new types like `mpiComm` or `mpiRequest`
- no problem with `mesh`, `matrix`, arrays

Assembly and direct solvers

- naive approach to distributed computing
- assuming some global variables may be replicated

Assembly and direct solvers

- naive approach to distributed computing
- assuming some global variables may be replicated

Assembly and direct solvers

- naive approach to distributed computing
- assuming some global variables may be replicated

Legacy interfaces

- MUMPS [Amestoy et al. 2001]
- PaStiX [Hénon et al. 2002]
- SuperLU_DIST [Li 2005]

Assembly and direct solvers

- naive approach to distributed computing
- assuming some global variables may be replicated

Legacy interfaces $\Leftarrow$ use at your own risk!

- MUMPS [Amestoy et al. 2001]
- PaStiX [Hénon et al. 2002]
- SuperLU_DIST [Li 2005]

ParaView

- more powerful postprocessing tool
- handle distributed solutions
- generate movies if needed
- `savevtk("sol.vtu",Th,sol)`

Example 3: heat equation [example3.edp]

Aim

Solve the transient PDE

$$\frac{\partial u}{\partial t} - \Delta u = 1 \quad \text{in } \Omega \times [0; T]$$
$$u(x, y, z, 0) = 1 \quad \text{in } \Omega$$
$$u(x, y, z, t) = 1 \quad \text{on } \Gamma \times [0; T]$$

Example 3: heat equation [example3.edp]

Aim

Solve the transient PDE

$$\begin{aligned} \frac{\partial u}{\partial t} - \Delta u &= 1 \quad && \text{in } \Omega \times [0;T] \\ u(x,y,z,0) &= 1 \quad && \text{in } \Omega \\ u(x,y,z,t) &= 1 \quad && \text{on } \Gamma \times [0;T] \end{aligned}$$

Implicit Euler scheme

$$\int_\Omega \frac{u^{n+1} - u^n}{\mathrm{d}t} w + \nabla u^n \nabla w = \int_\Omega w$$

Example 3: heat equation [example3.edp]

Aim

Solve the transient PDE

$$\begin{aligned} \frac{\partial u}{\partial t} - \Delta u &= 1 \quad && \text{in } \Omega \times [0;T] \\ u(x,y,z,0) &= 1 \quad && \text{in } \Omega \\ u(x,y,z,t) &= 1 \quad && \text{on } \Gamma \times [0;T] \end{aligned}$$

Implicit Euler scheme

$$\int_\Omega \frac{u^{n+1} - u^n}{\mathrm{d}t} w + \nabla u^n \nabla w = \int_\Omega w$$

$$\Downarrow$$

$$(M + \mathrm{d}t \cdot A) u^{n+1} = M u^n + \mathrm{d}t \cdot b$$

Overlapping Schwarz methods

History

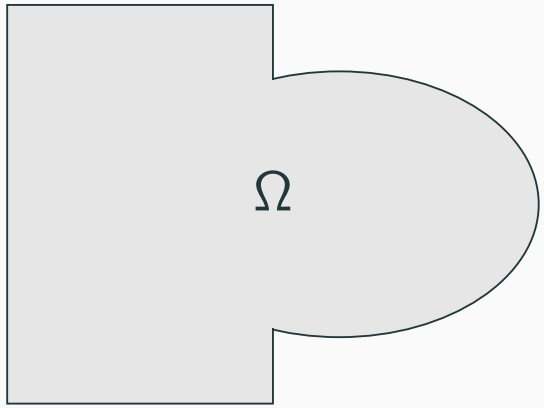
- initially focused on domain decomposition methods
- new developments around iterative methods [Jolivet and Tournier 2016]

- only interfaced with FreeFEM at first
- low-level languages (C/C++, Fortran, Python)

- first commit in late 2011
- open-sourced in late 2014
- `https://github.com/hpddm/hpddm`
- integrated in PETSc in 2019

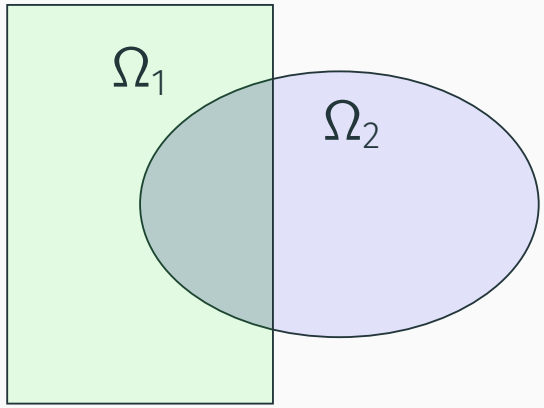
Historical method

- due to Schwarz (1870)
- how to solve Poisson equation on “complex” geometries?



Historical method

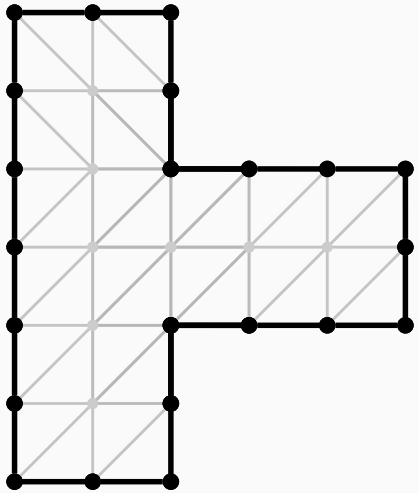
- due to Schwarz (1870)
- how to solve Poisson equation on “complex” geometries?



- by using solvers that work on simpler subdomains

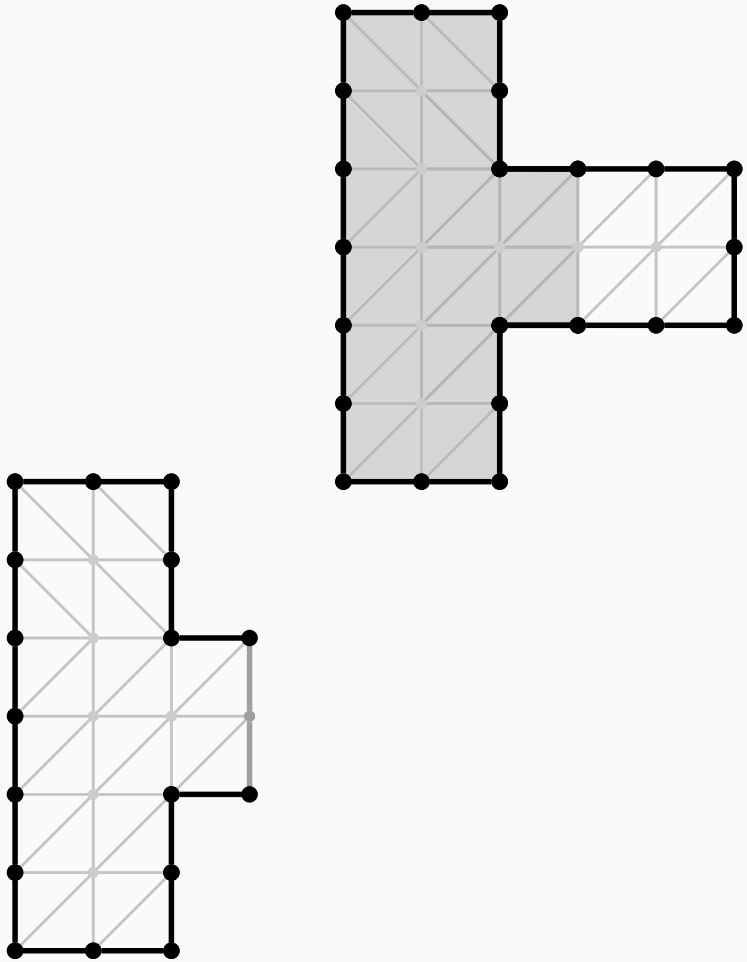
Discretized model problem

[example1.edp]



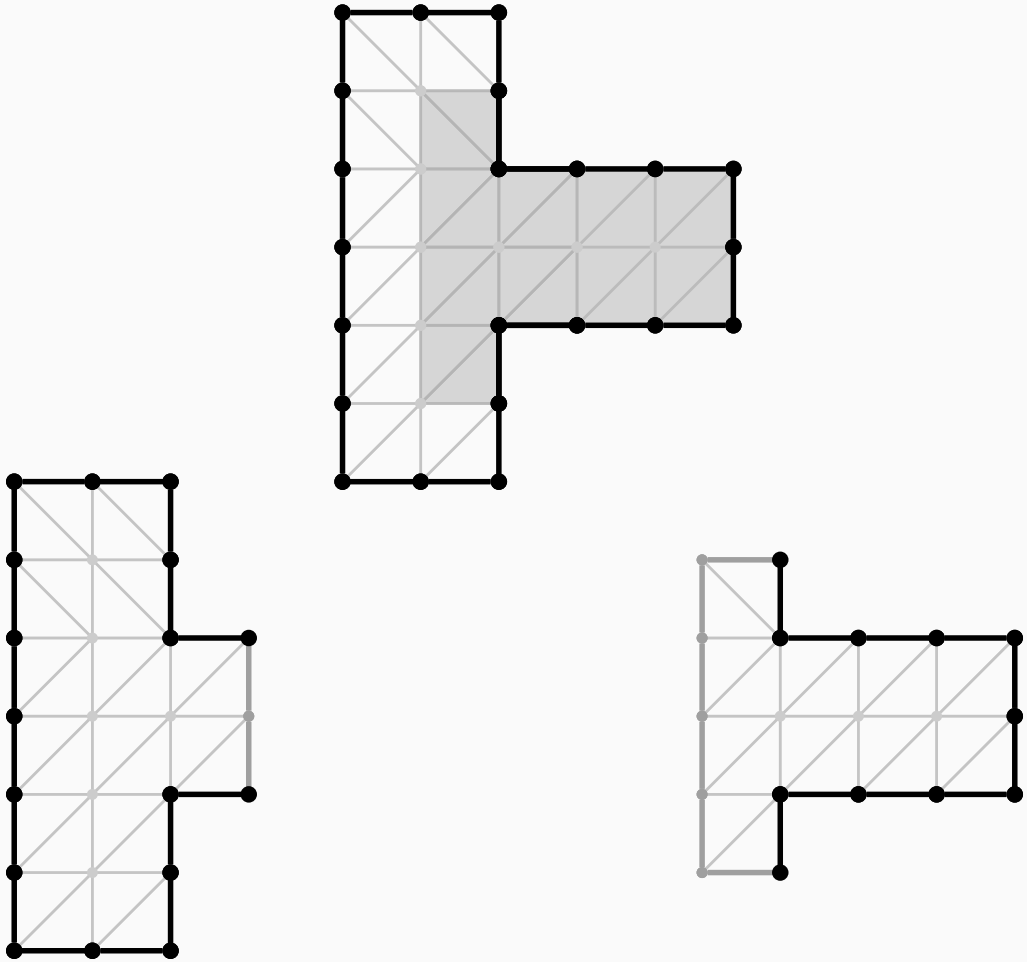
Discretized model problem

[example1.edp]



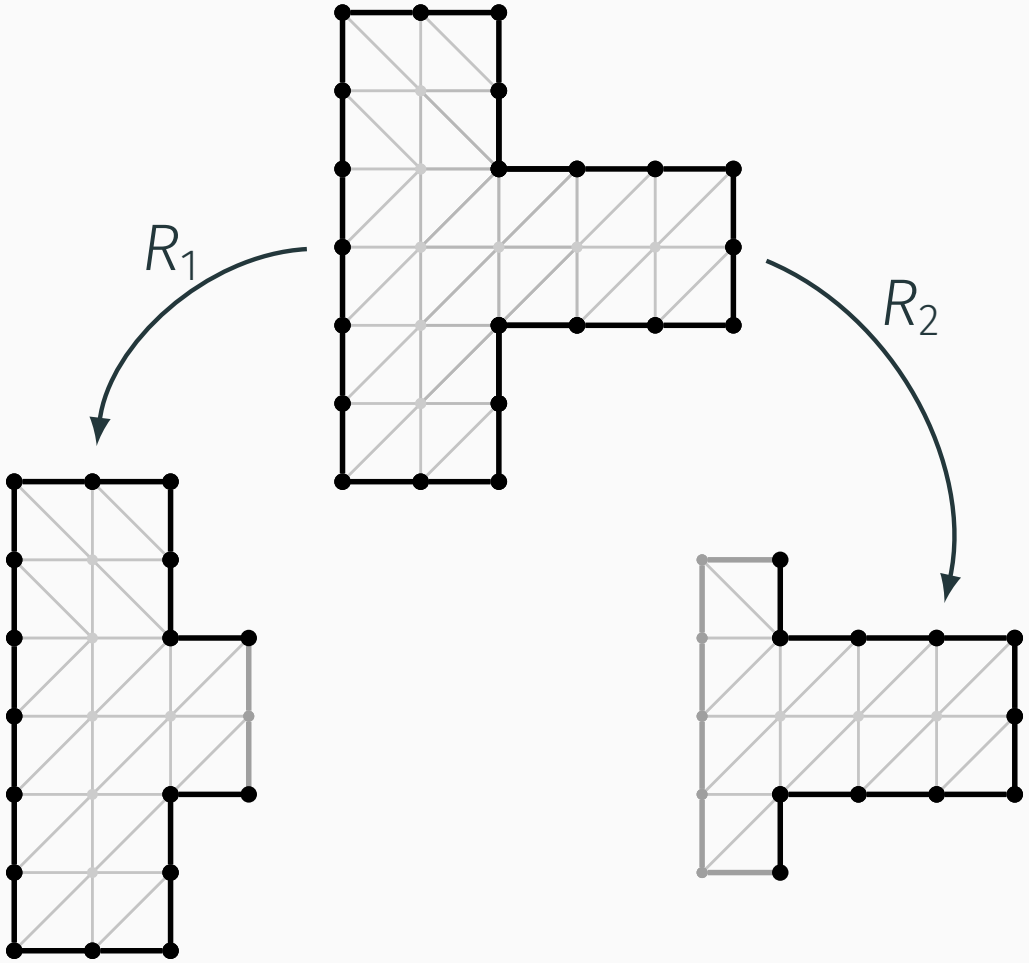
Discretized model problem

[example1.edp]



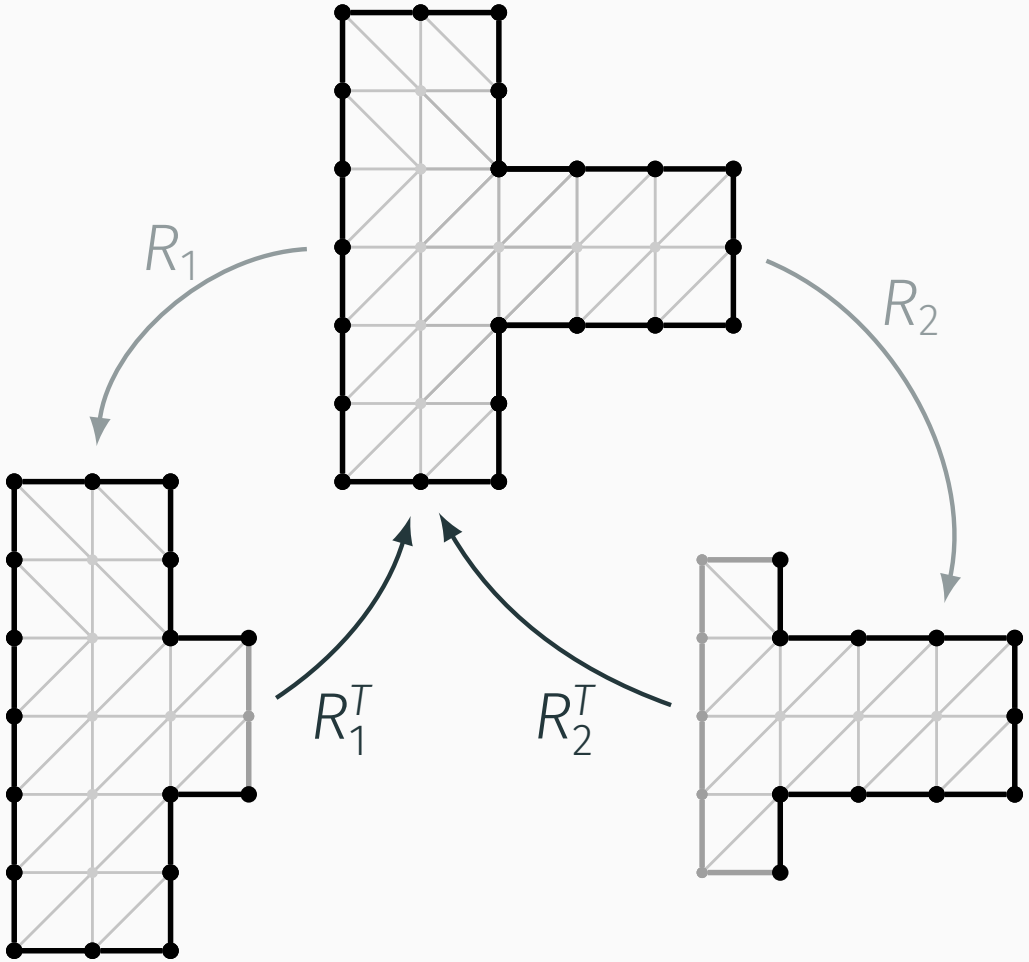
Discretized model problem

[example1.edp]

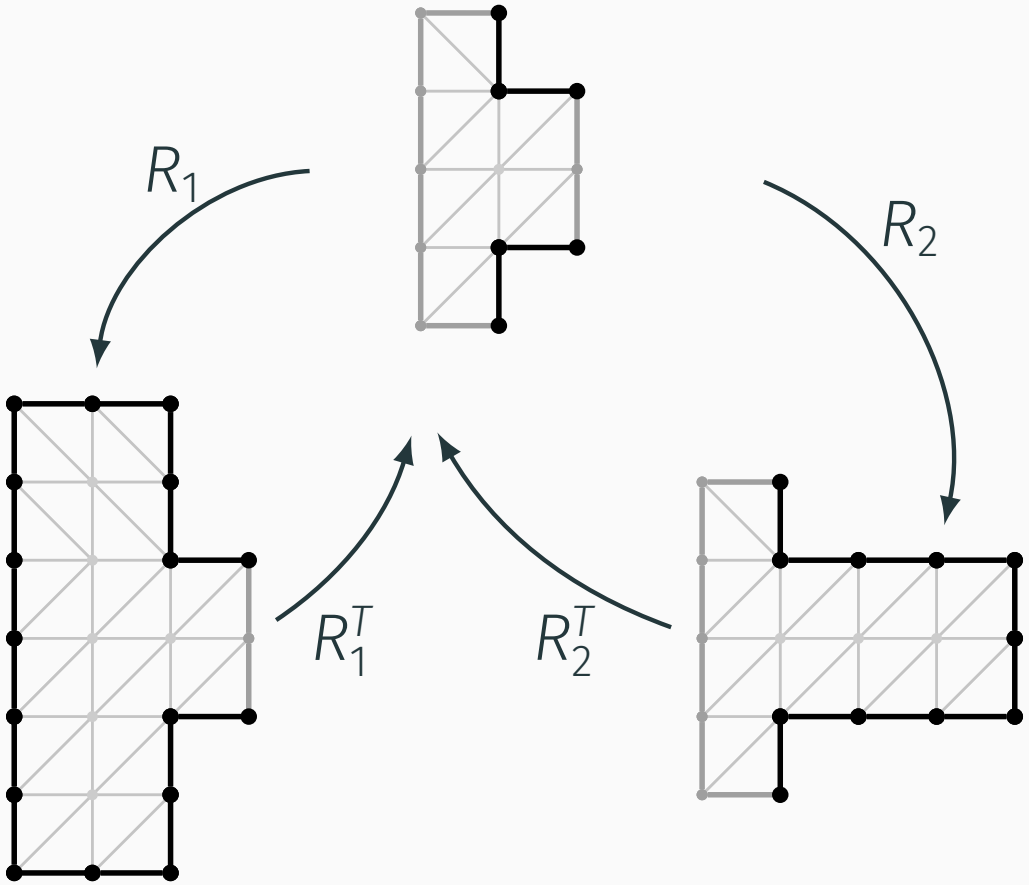


Discretized model problem

[example1.edp]



Discretized model problem [example1.edp]



- no need for the complete mesh
- neighboring numberings on the overlaps

Unpreconditioned iterative methods

- partition of unity

$$I = \sum_{i=1}^{N} R_i^T D_i R_i$$

Unpreconditioned iterative methods

- partition of unity $$I = \sum_{i=1}^{N} R_i^T D_i R_i$$
- scalar product $$(u, v) = \sum_{i=1}^{N} (R_i u, D_i R_i v)$$

Unpreconditioned iterative methods

- partition of unity $$I = \sum_{i=1}^{N} R_i^T D_i R_i$$
- scalar product $$(u, v) = \sum_{i=1}^{N} (R_i u, D_i R_i v)$$
- matrix–vector product $$R_i A u = R_i \sum_{j=1}^{N} R_j^T R_j A R_j^T D_j R_j u$$

Unpreconditioned iterative methods

- partition of unity $$I = \sum_{i=1}^{N} R_i^T D_i R_i$$
- scalar product $$(u, v) = \sum_{i=1}^{N} (R_i u, D_i R_i v)$$
- matrix–vector product $$R_i A u = R_i \sum_{j=1}^{N} R_j^T R_j A R_j^T D_j R_j u$$

$\Longrightarrow$ subdomains only require “local data” [example2.edp]

Overlapping preconditioners [example3.edp]

- additive Schwarz

$$M_{\text{ASM}}^{-1} = \sum_{i=1}^{N} R_i^T \left(R_i A R_i^T \right)^{-1} R_i$$

Overlapping preconditioners [example3.edp]

- additive Schwarz

$$M_{\text{ASM}}^{-1} = \sum_{i=1}^{N} R_i^T \left(R_i A R_i^T \right)^{-1} R_i$$

- restricted additive Schwarz [Cai et al. 2003]

$$M_{\text{RAS}}^{-1} = \sum_{i=1}^{N} R_i^T D_i \left(R_i A R_i^T \right)^{-1} R_i$$

Overlapping preconditioners [example3.edp]

- additive Schwarz

$$M_{\text{ASM}}^{-1} = \sum_{i=1}^{N} R_i^T \left(R_i A R_i^T \right)^{-1} R_i$$

- restricted additive Schwarz [Cai et al. 2003]

$$M_{\text{RAS}}^{-1} = \sum_{i=1}^{N} R_i^T D_i \left(R_i A R_i^T \right)^{-1} R_i$$

- optimized restricted additive Schwarz [St-Cyr et al. 2007]

$$M_{\text{ORAS}}^{-1} = \sum_{i=1}^{N} R_i^T D_i B_i^{-1} R_i$$

Some utility routines

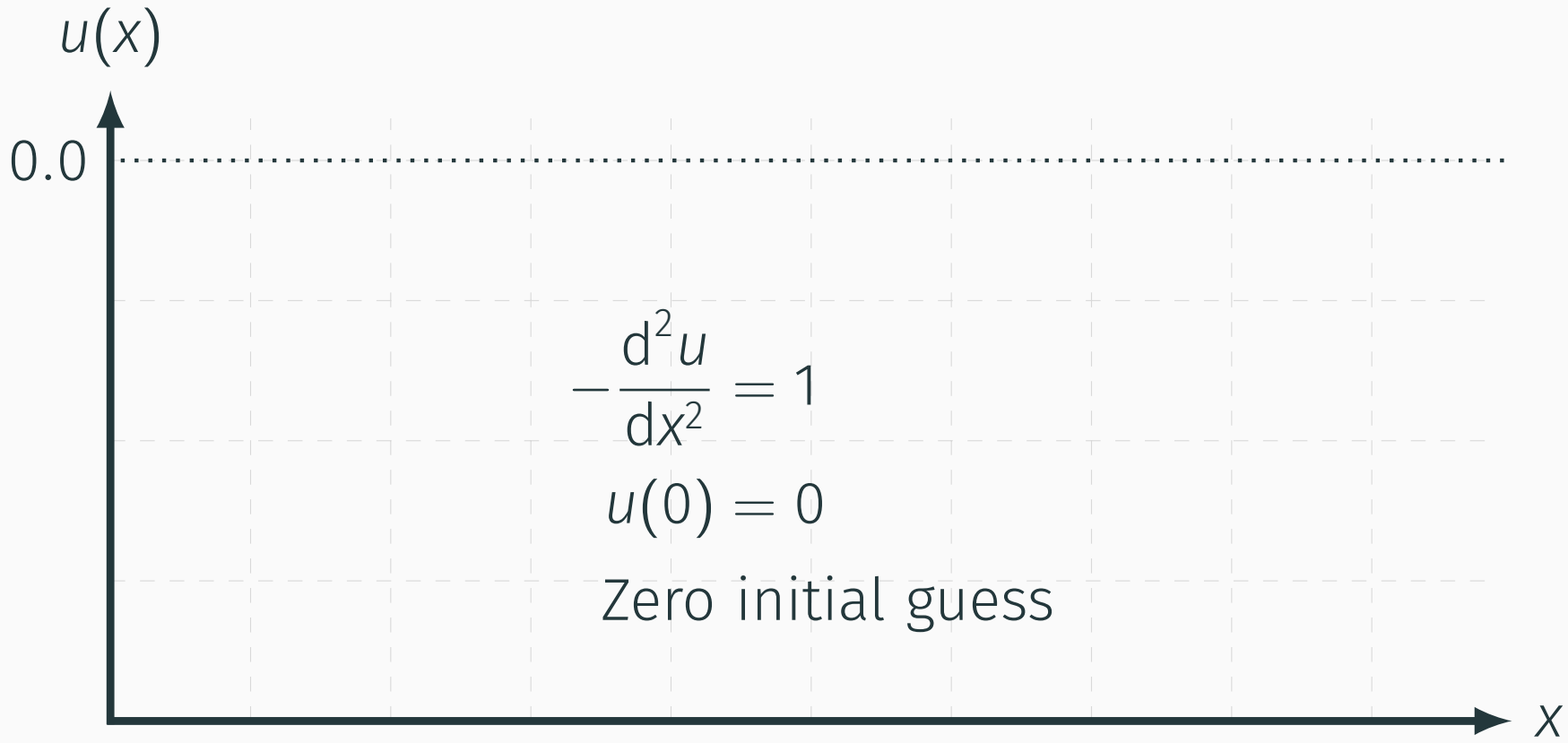
- `exchange` for consistency in the overlap
- `statistics` to get some metrics about the DD
- `A(u,v)` to compute weighted dot products
- `ChangeOperator` to update a local matrix

Coarse grid preconditioners

- convergence independently of the decomposition

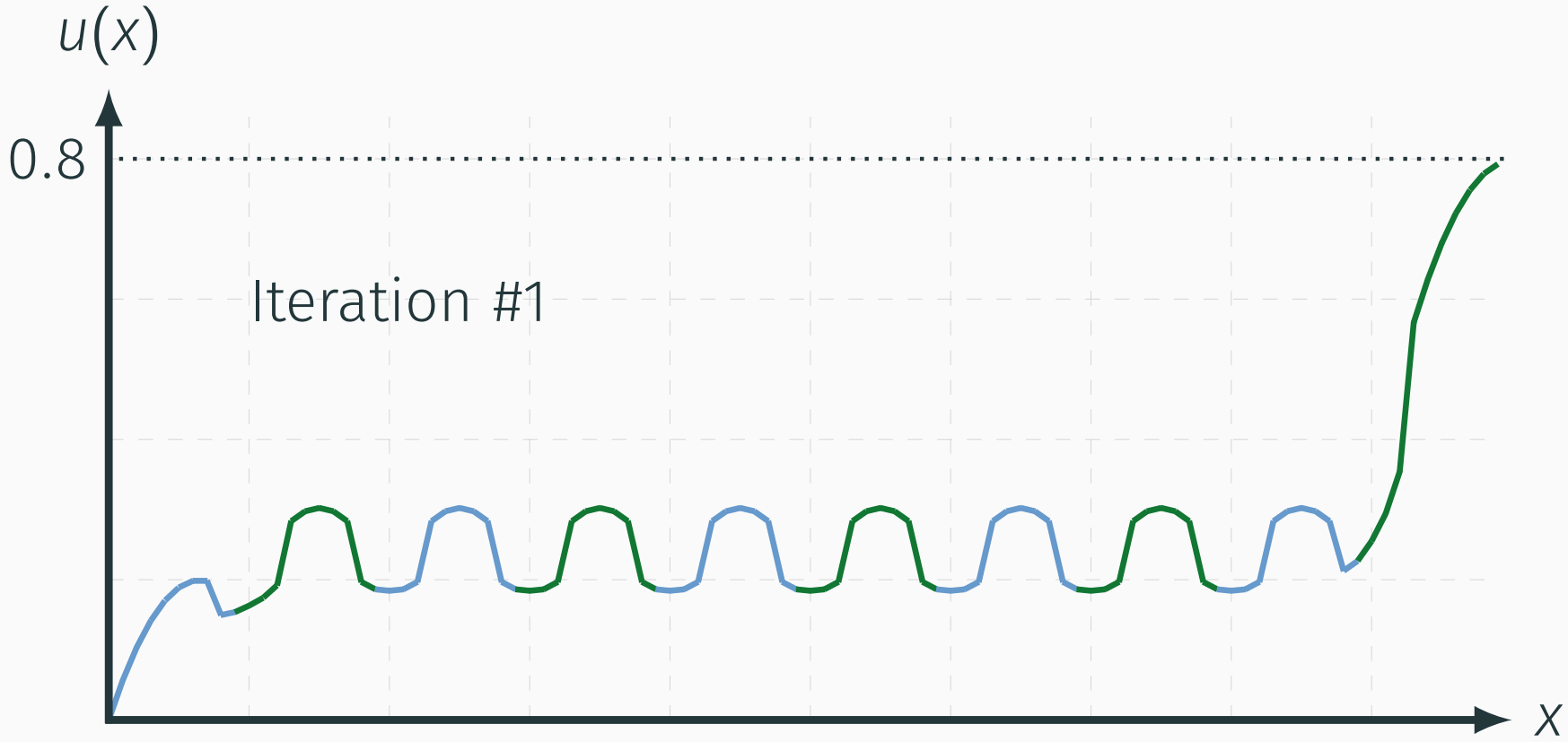
Coarse grid preconditioners

- convergence independently of the decomposition



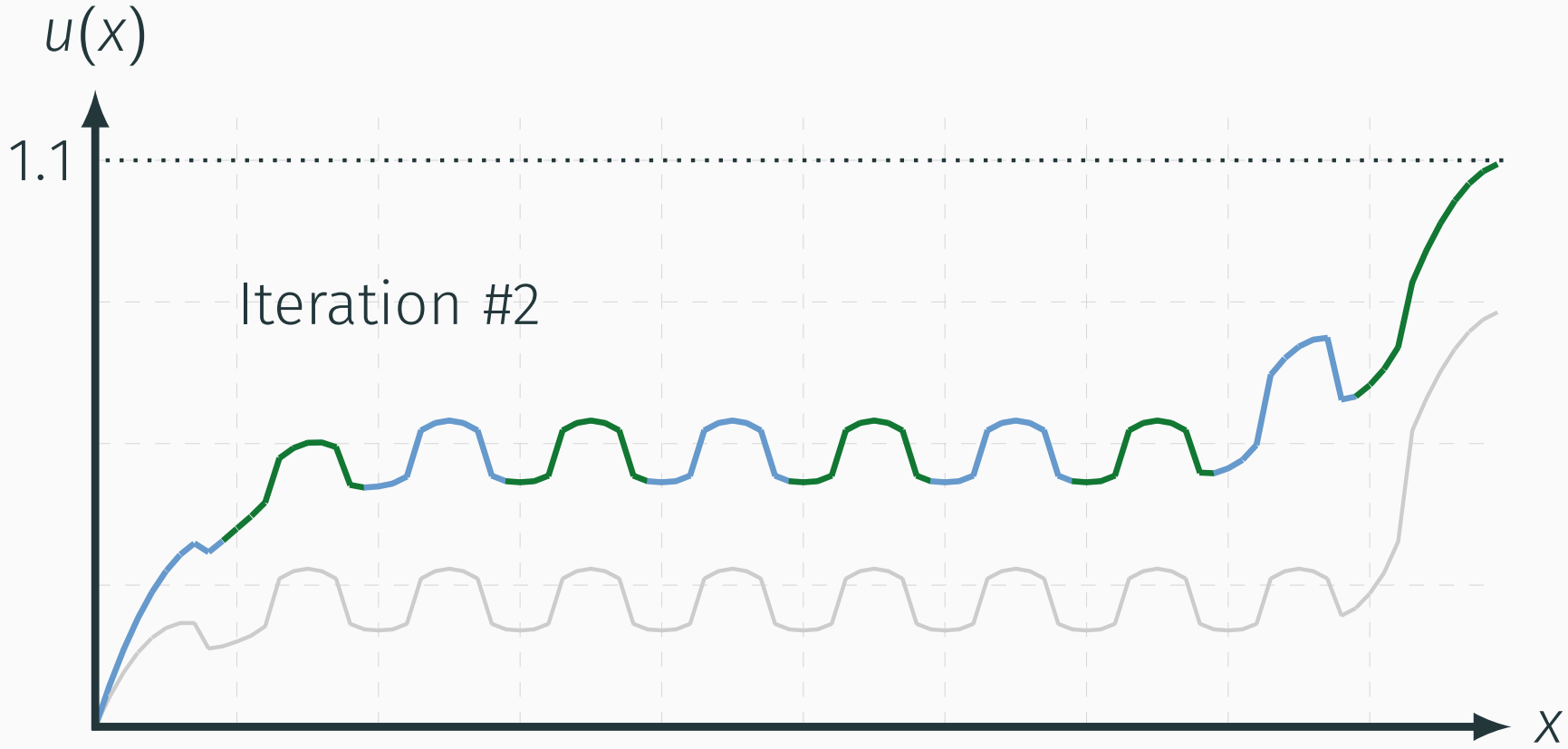
Coarse grid preconditioners

- convergence independently of the decomposition



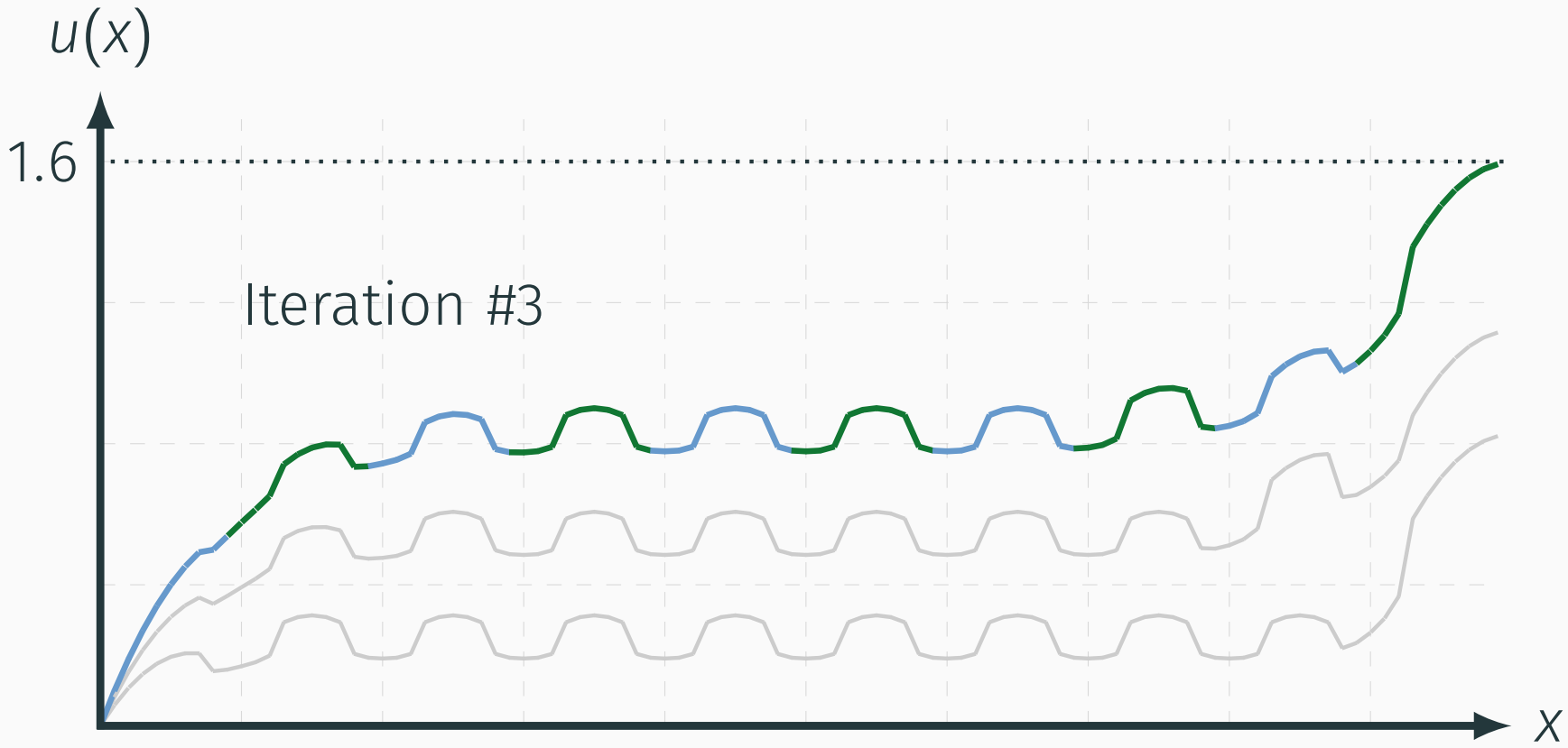
Coarse grid preconditioners

- convergence independently of the decomposition



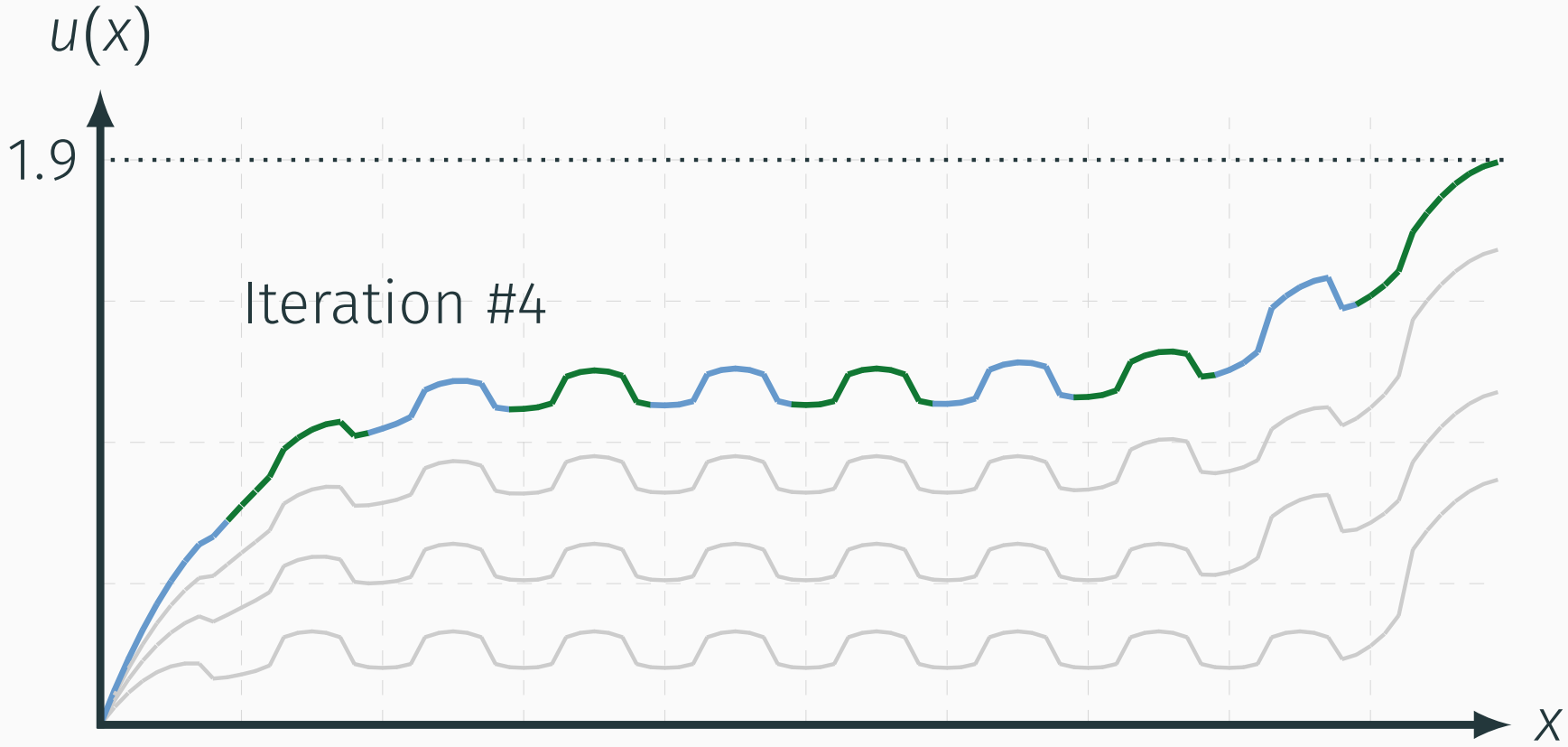
Coarse grid preconditioners

- convergence independently of the decomposition



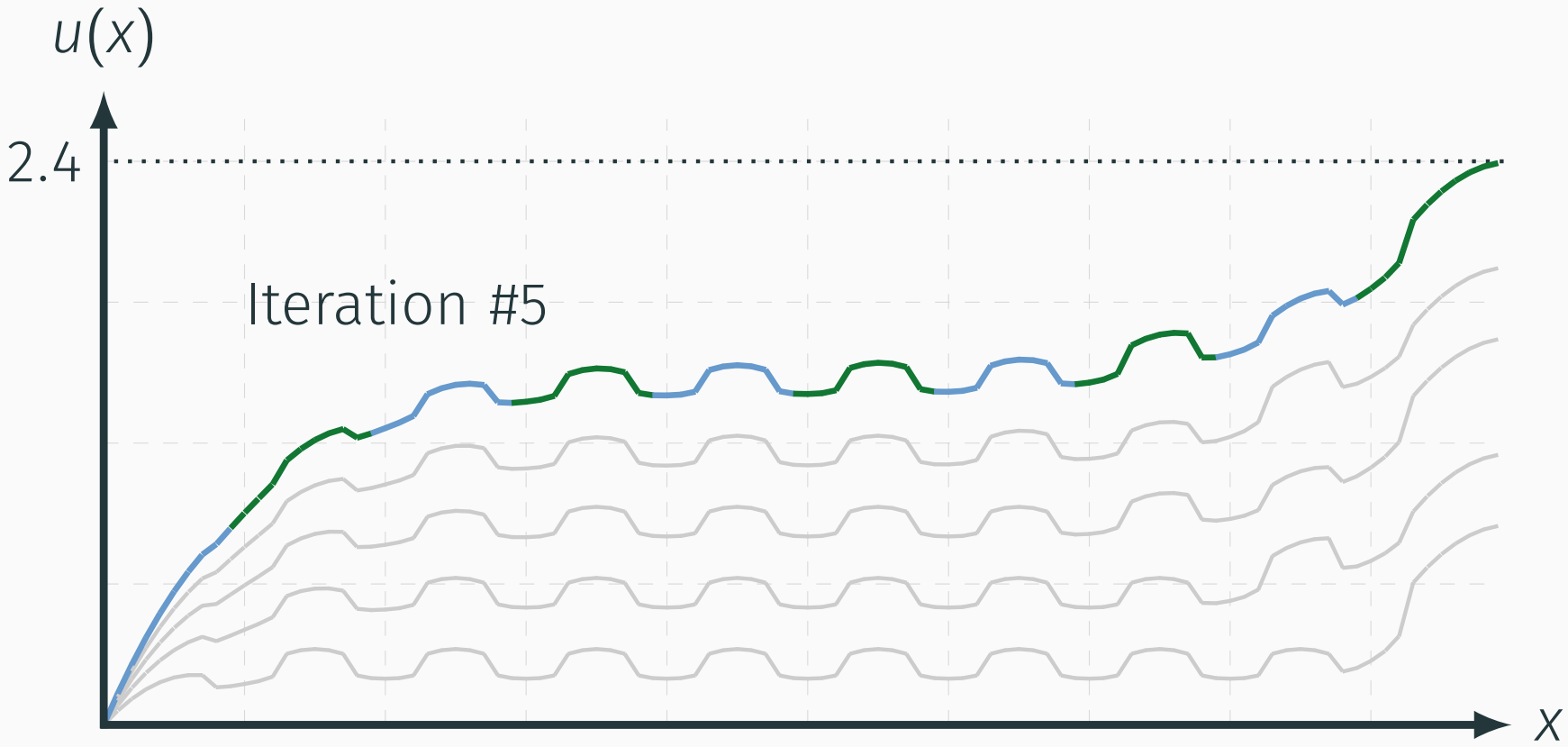
Coarse grid preconditioners

- convergence independently of the decomposition



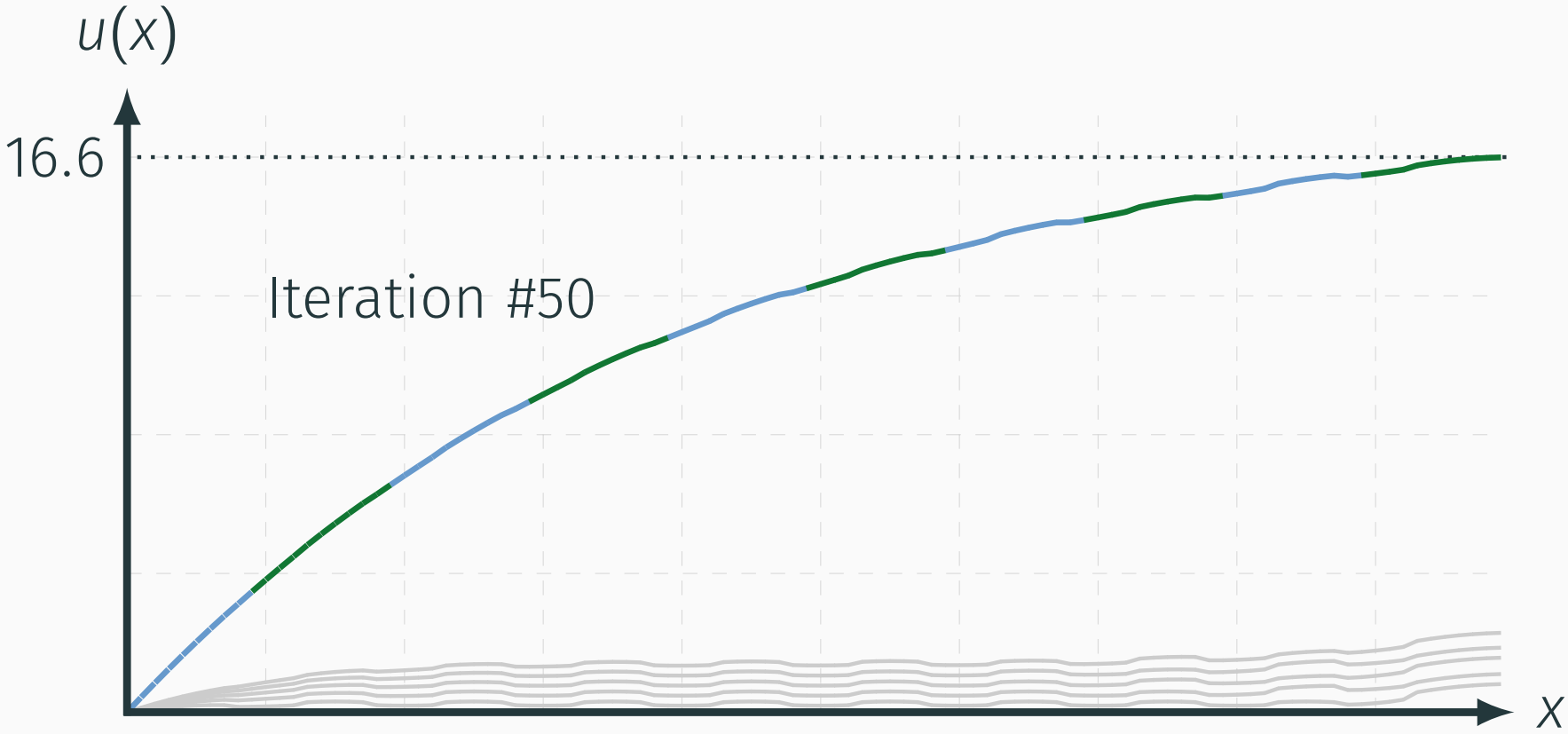
Coarse grid preconditioners

- convergence independently of the decomposition



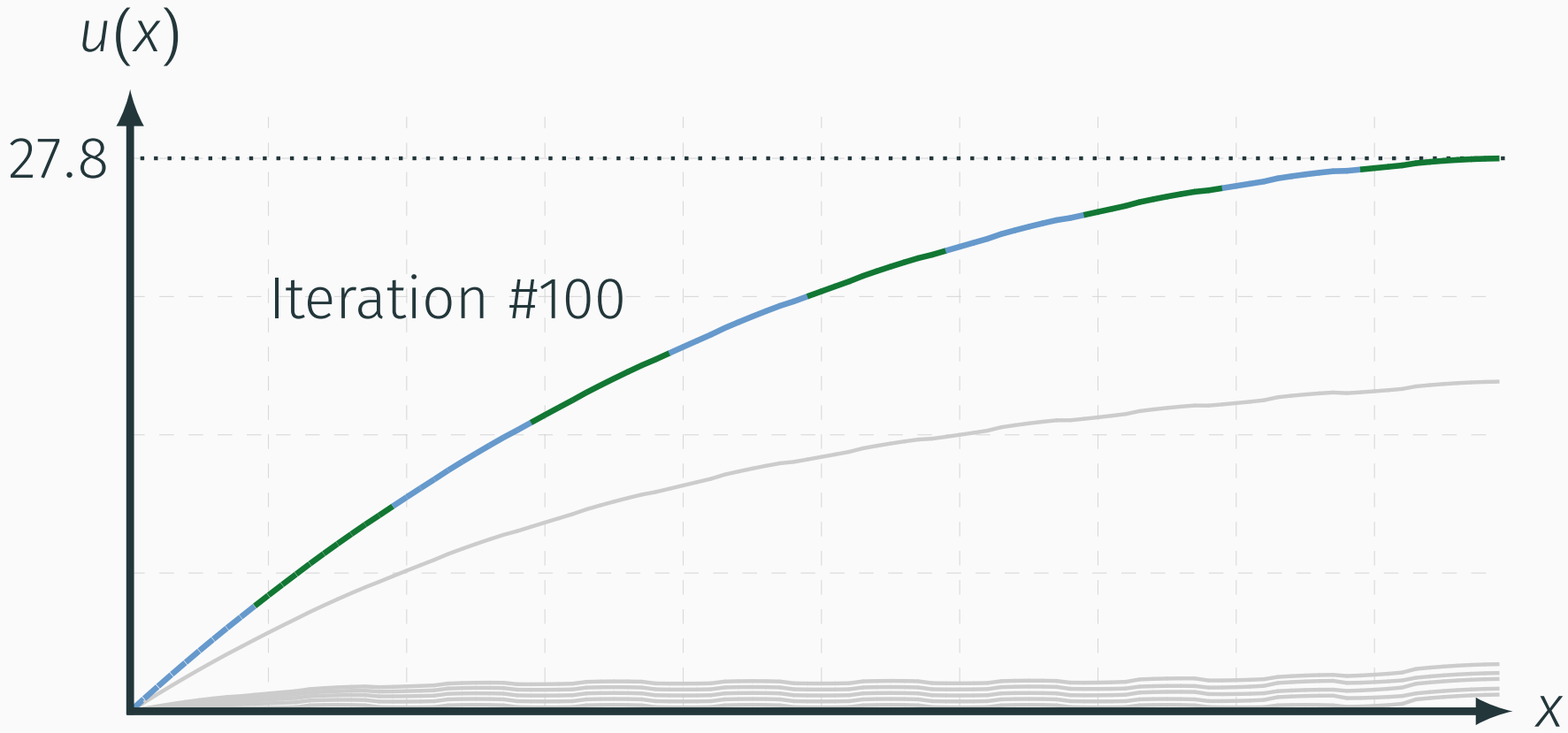
Coarse grid preconditioners

- convergence independently of the decomposition



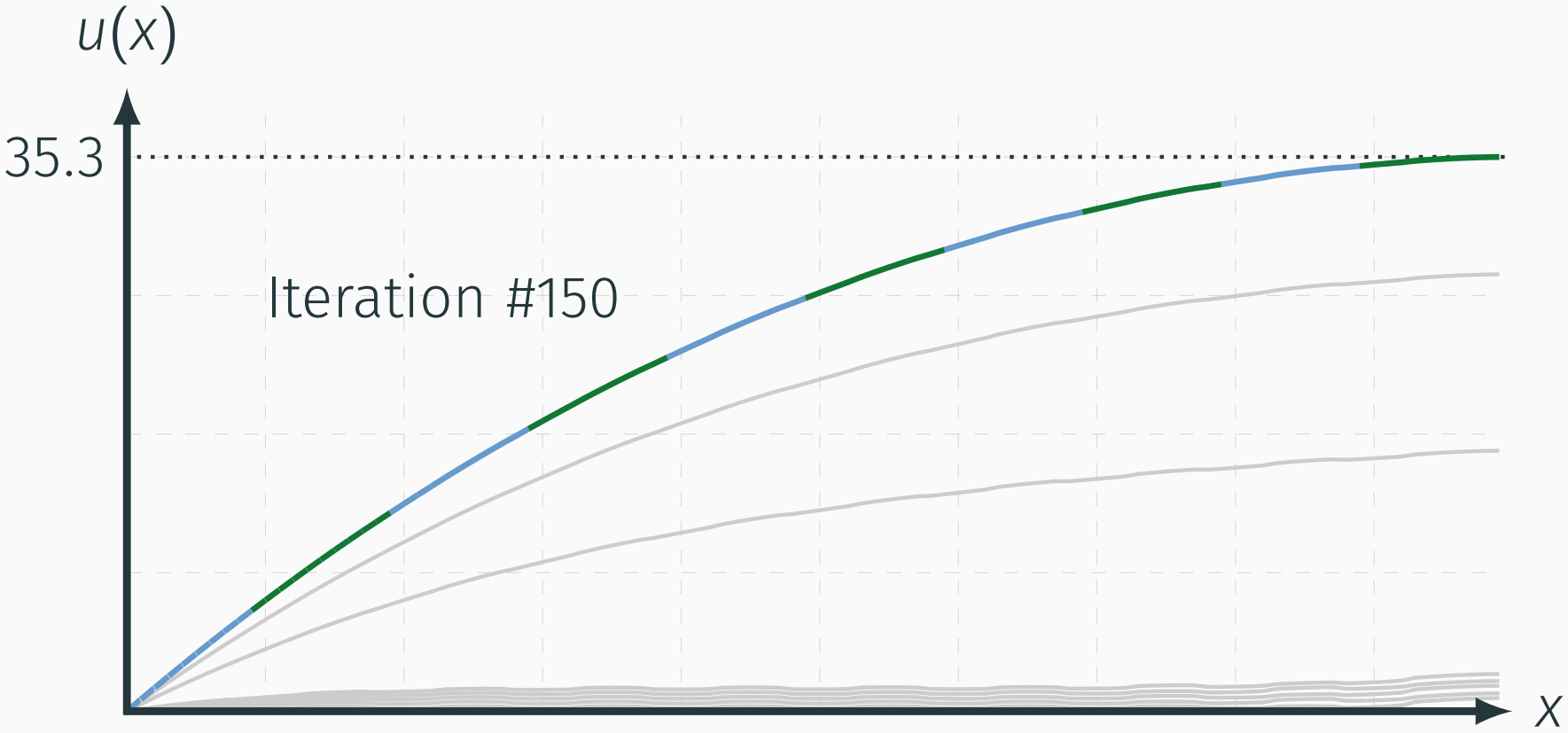
Coarse grid preconditioners

- convergence independently of the decomposition



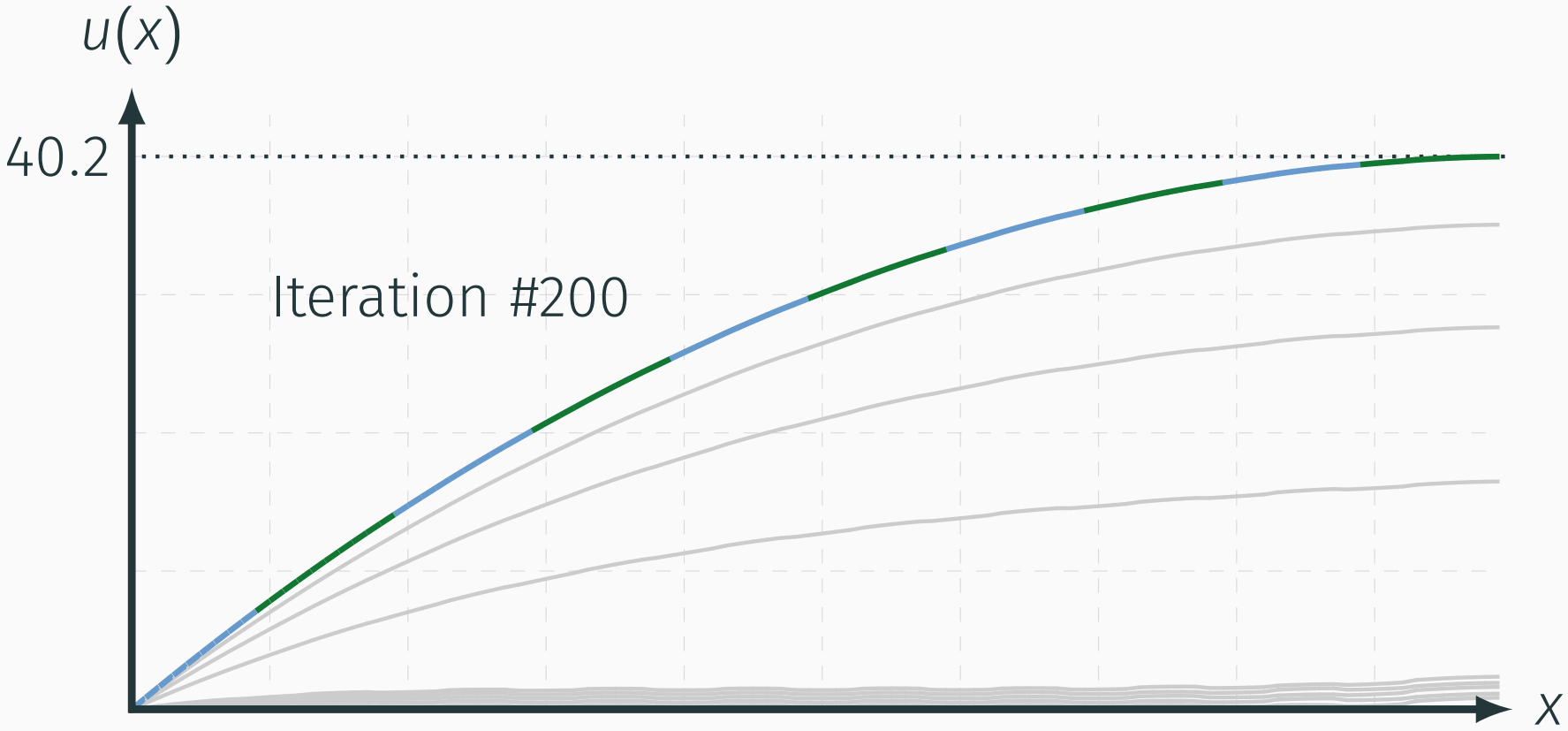
Coarse grid preconditioners

- convergence independently of the decomposition



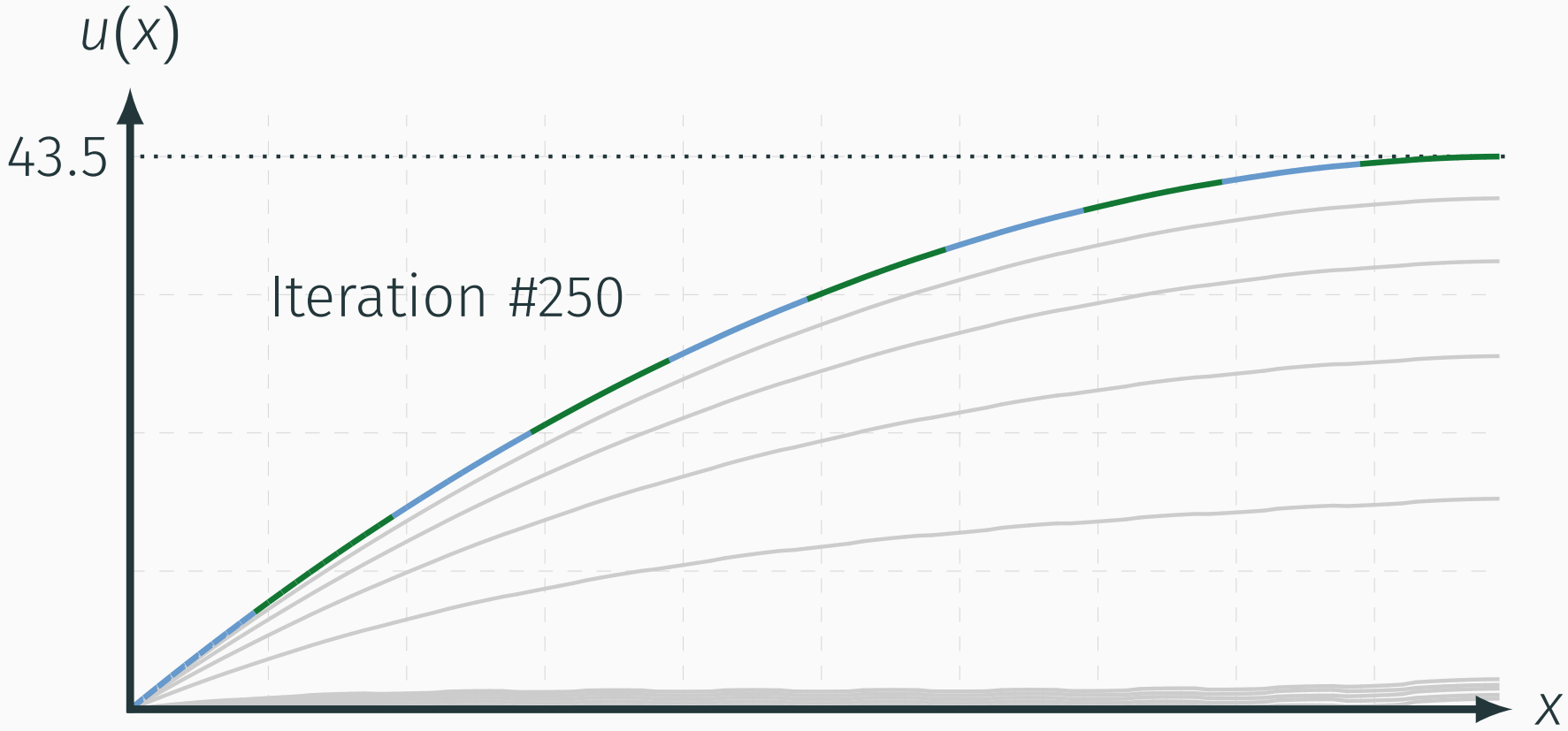
Coarse grid preconditioners

- convergence independently of the decomposition



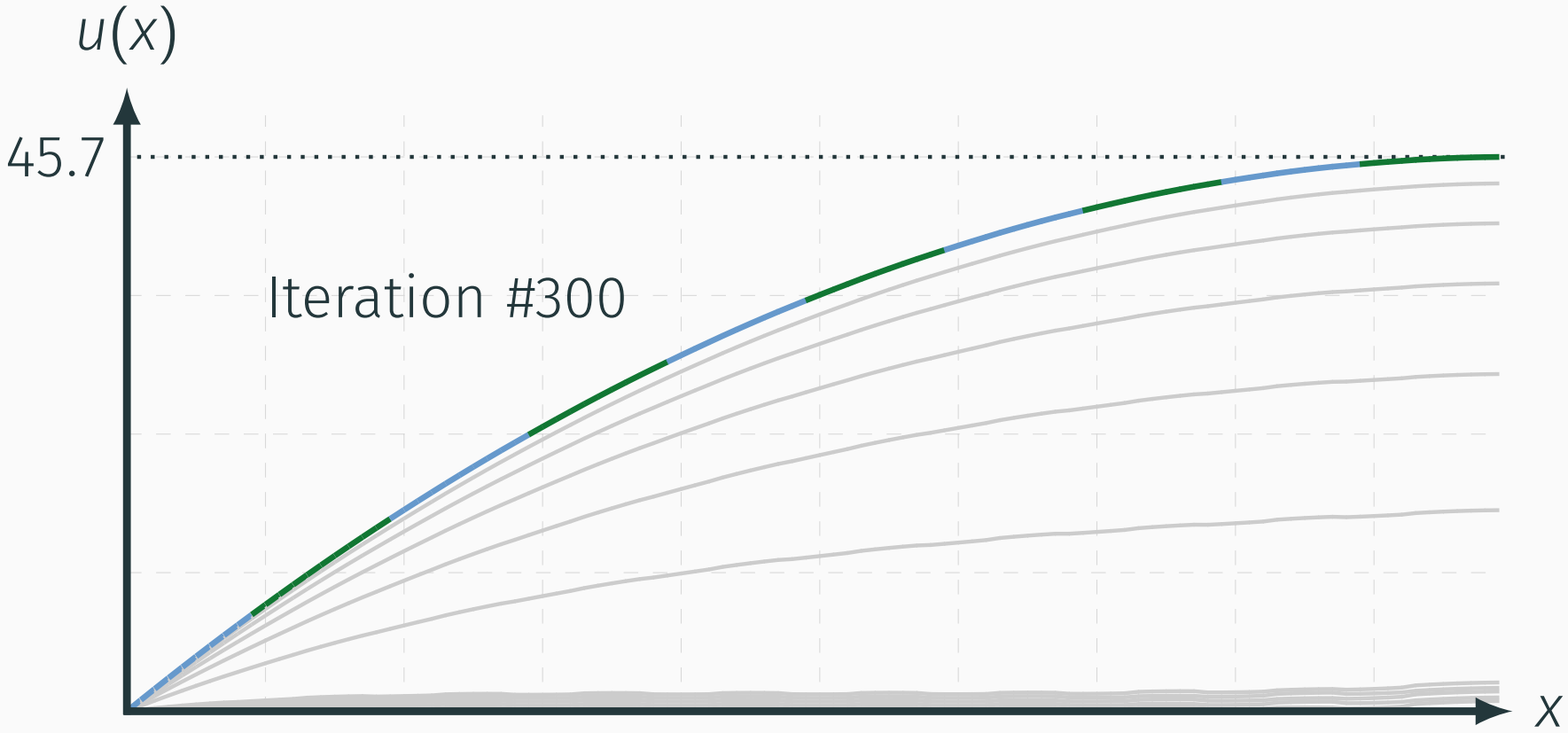
Coarse grid preconditioners

- convergence independently of the decomposition



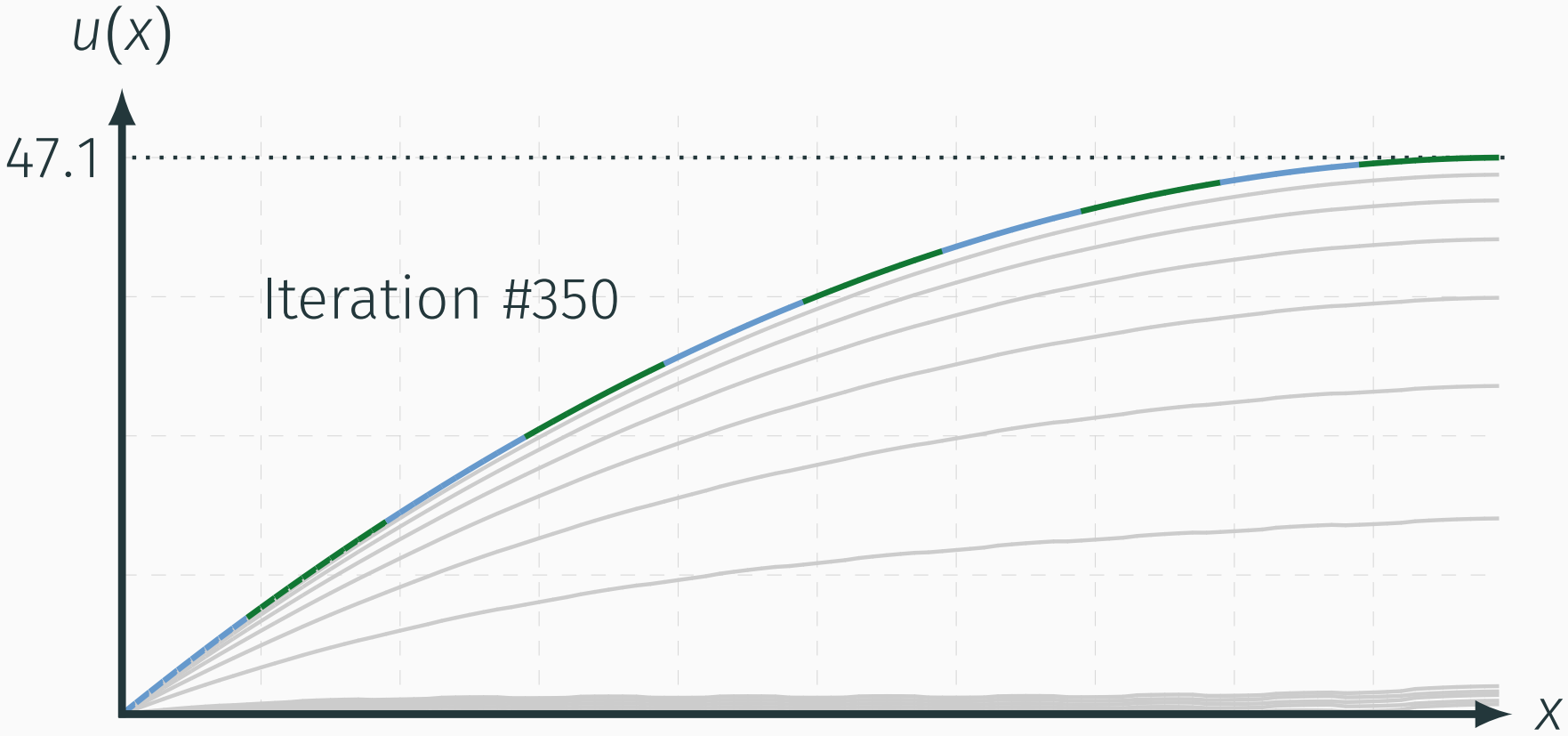
Coarse grid preconditioners

- convergence independently of the decomposition



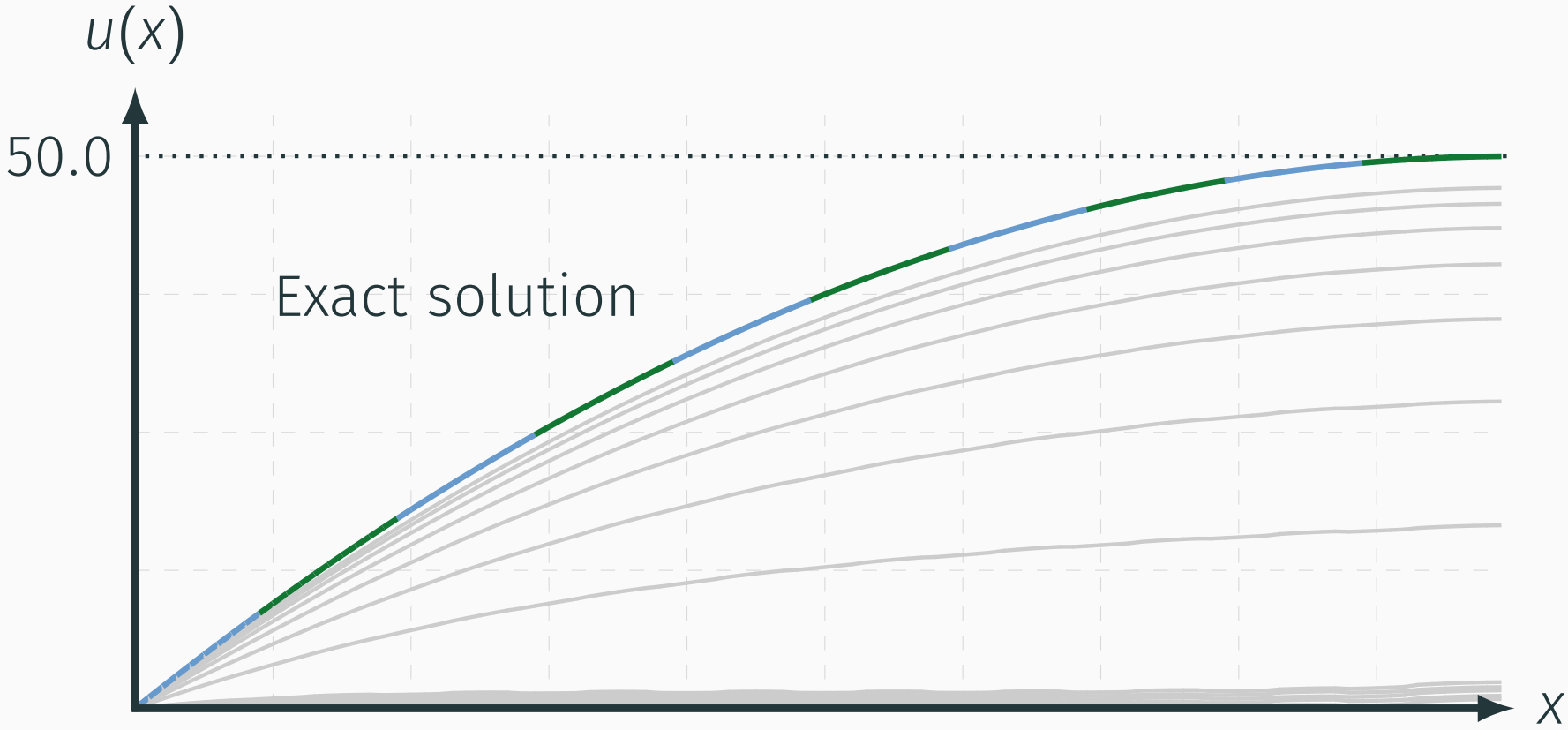
Coarse grid preconditioners

- convergence independently of the decomposition



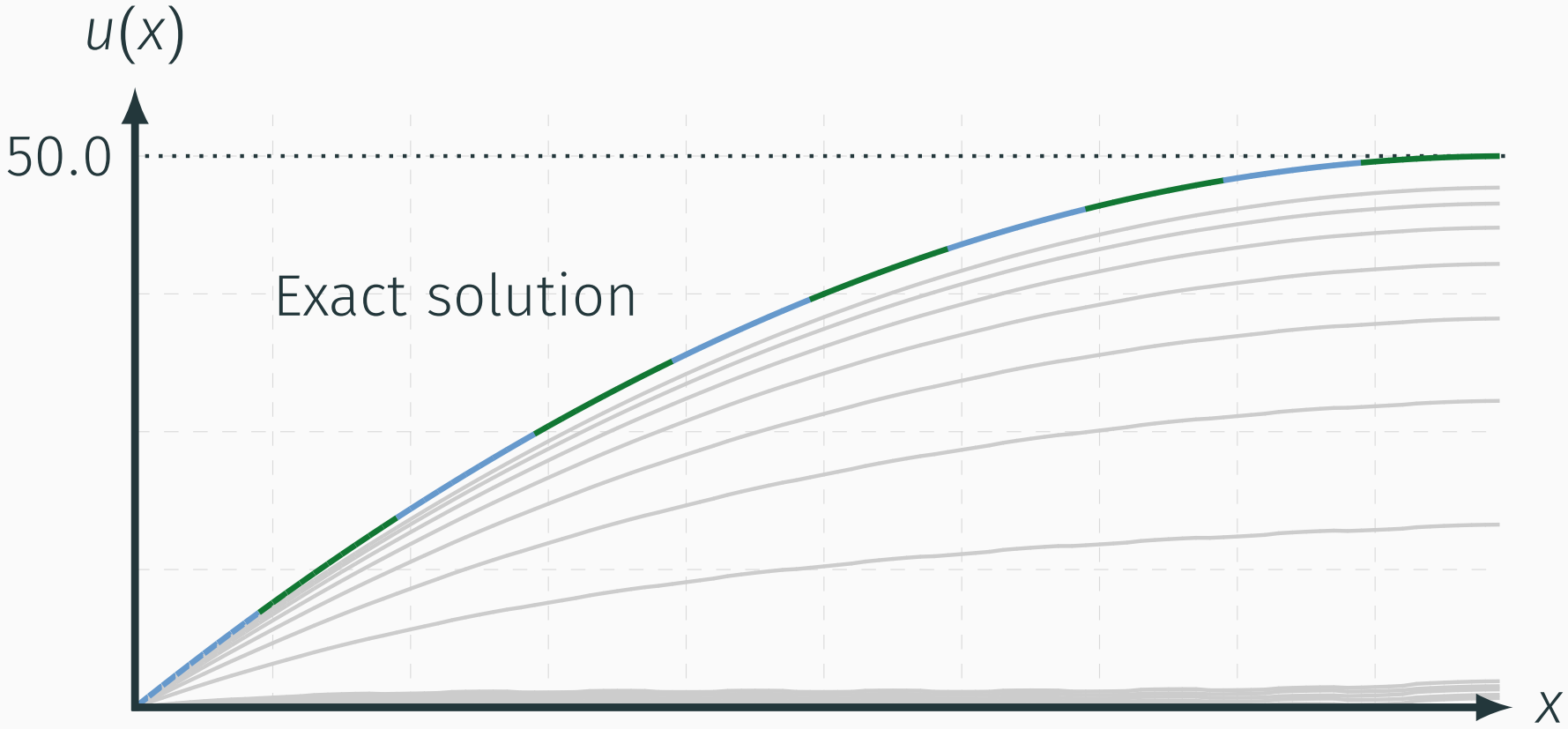
Coarse grid preconditioners

- convergence independently of the decomposition



Coarse grid preconditioners

- convergence independently of the decomposition



- `AttachCoarseOperator` not always trivial to define

Example 4: GenEO coarse operator [example4.edp]

Aim

- robust DDM [Spillane et al. 2013]
- $\sim$ incompressible elasticity [Haferssas et al. 2017]
- highly parametrizable [Jolivet, Hecht, Nataf, et al. 2013]
- scalable [Al Daas et al. 2021]

Example 4: GenEO coarse operator [example4.edp]

Aim

- robust DDM [Spillane et al. 2013]
- $\sim$ incompressible elasticity [Haferssas et al. 2017]
- highly parametrizable [Jolivet, Hecht, Nataf, et al. 2013]
- scalable [Al Daas et al. 2021]

- `-hpddm_schwarz_method` [asm|ras|osm]
- `-hpddm_geneo_nu` *n*
- `-hpddm_level_2_p` *p*

Substructuring methods

Algebraic decomposition

$$A = \begin{bmatrix} A_{11} & 0 & A_{1\Gamma} \\ 0 & A_{22} & A_{2\Gamma} \\ A_{\Gamma 1} & A_{\Gamma 2} & A_{\Gamma\Gamma} \end{bmatrix} \qquad b = \begin{bmatrix} b_1 \\ b_2 \\ b_\Gamma \end{bmatrix}$$

$$\Downarrow$$

$$\begin{aligned} \left(A_{\Gamma\Gamma} - A_{\Gamma 1}A_{11}^{-1}A_{1\Gamma} - A_{\Gamma 2}A_{22}^{-1}A_{2\Gamma}\right) x_\Gamma &= b_\Gamma - A_{\Gamma 1}A_{11}^{-1}b_1 - A_{\Gamma 2}A_{22}^{-1}b_2 \\ &= g_\Gamma = g_\Gamma^{(1)} + g_\Gamma^{(2)} \end{aligned}$$

Algebraic decomposition

$$A = \begin{bmatrix} A_{11} & 0 & A_{1\Gamma} \\ 0 & A_{22} & A_{2\Gamma} \\ A_{\Gamma 1} & A_{\Gamma 2} & A_{\Gamma\Gamma} \end{bmatrix} \qquad b = \begin{bmatrix} b_1 \\ b_2 \\ b_\Gamma \end{bmatrix}$$

$$\Downarrow$$

$$\begin{aligned} \left(A_{\Gamma\Gamma} - A_{\Gamma 1}A_{11}^{-1}A_{1\Gamma} - A_{\Gamma 2}A_{22}^{-1}A_{2\Gamma}\right) x_\Gamma &= b_\Gamma - A_{\Gamma 1}A_{11}^{-1}b_1 - A_{\Gamma 2}A_{22}^{-1}b_2 \\ &= g_\Gamma = g_\Gamma^{(1)} + g_\Gamma^{(2)} \end{aligned}$$

Schur complement(s)

$$\begin{aligned} S_p &= A_{\Gamma\Gamma}^{(1)} - A_{\Gamma 1}A_{11}^{-1}A_{1\Gamma} + A_{\Gamma\Gamma}^{(2)} - A_{\Gamma 2}A_{22}^{-1}A_{2\Gamma} \\ &= S_p^{(1)} + S_p^{(2)} \end{aligned}$$

Condensed system

Preconditioning $S_p x_\Gamma = g_\Gamma$

$$S_p^{(1)} = S_p^{(2)} \implies M^{-1} = \frac{1}{4}\left(S_p^{(1)^{-1}} + S_p^{(2)^{-1}}\right)$$

Condensed system

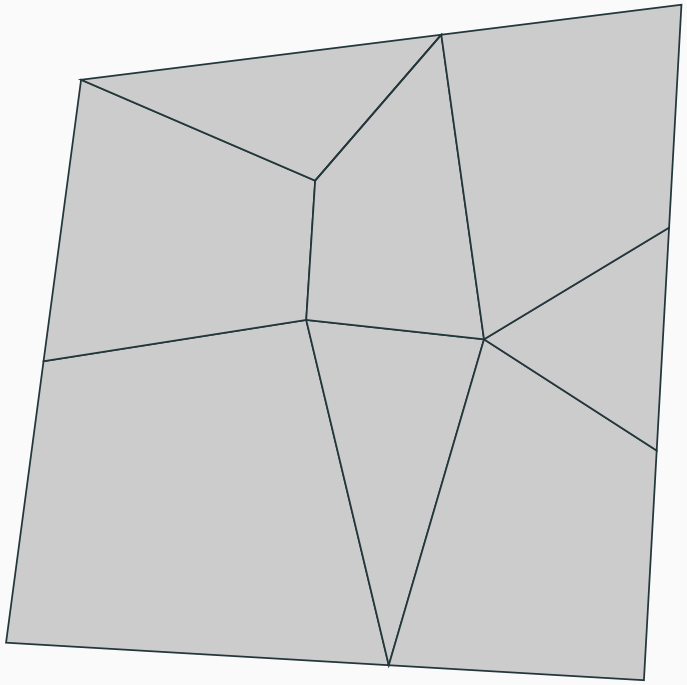
Preconditioning $S_p x_\Gamma = g_\Gamma$

$$S_p^{(1)} = S_p^{(2)} \implies M^{-1} = \frac{1}{4}\left(S_p^{(1)^{-1}} + S_p^{(2)^{-1}}\right)$$

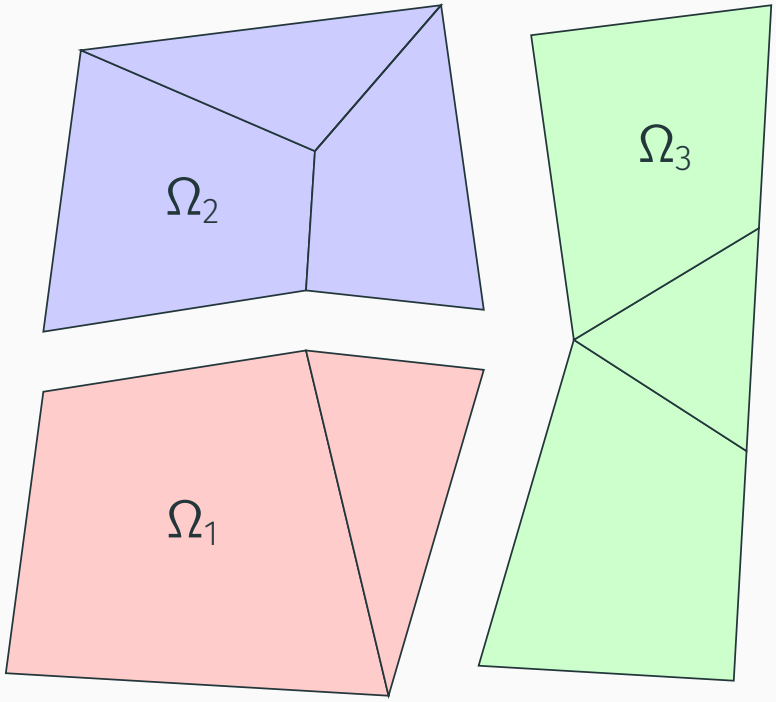
$$A^{(i)} = \begin{bmatrix} A_{ii} & A_{i\Gamma} \\ A_{\Gamma i} & A_{\Gamma\Gamma}^{(i)} \end{bmatrix}$$

$$= \begin{bmatrix} I & 0 \\ A_{\Gamma i}{A_{ii}}^{-1} & I \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & S_p^{(i)} \end{bmatrix} \begin{bmatrix} I & {A_{\Gamma\Gamma}^{(i)}}^{-1} A_{i\Gamma} \\ 0 & I \end{bmatrix}$$

General notations

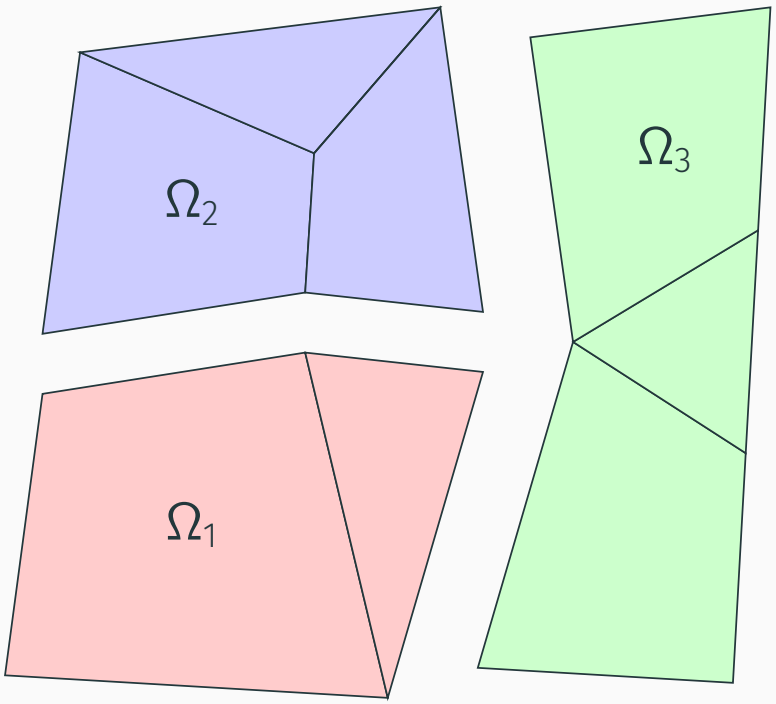


[Gosselet and Rey 2006]

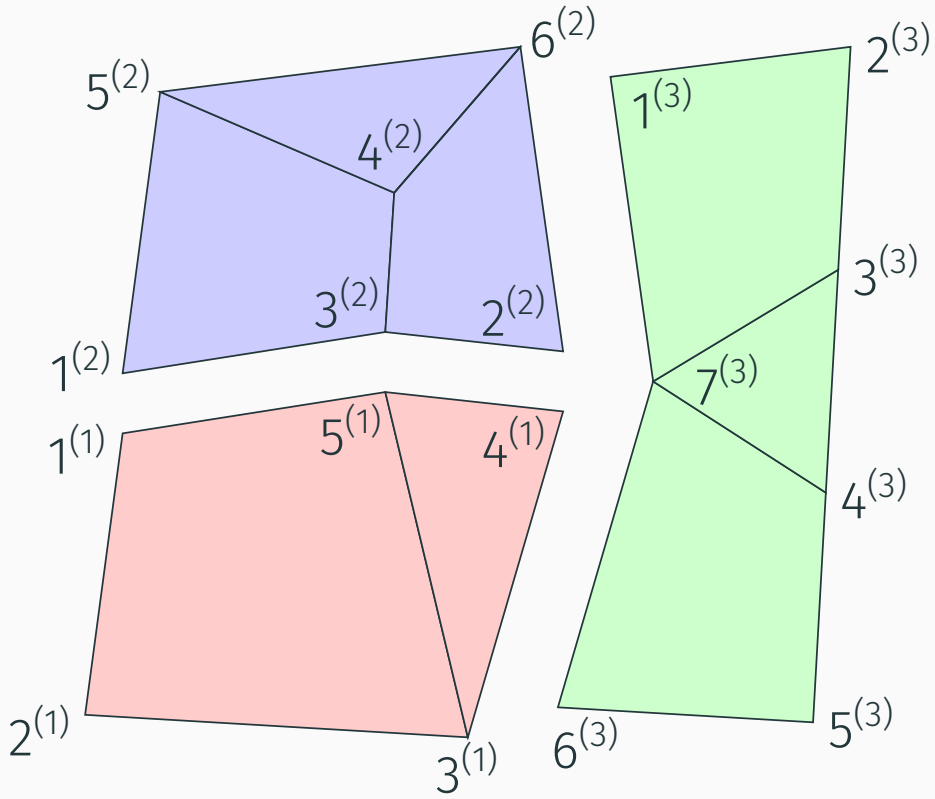


Subdomain tearing

General notations



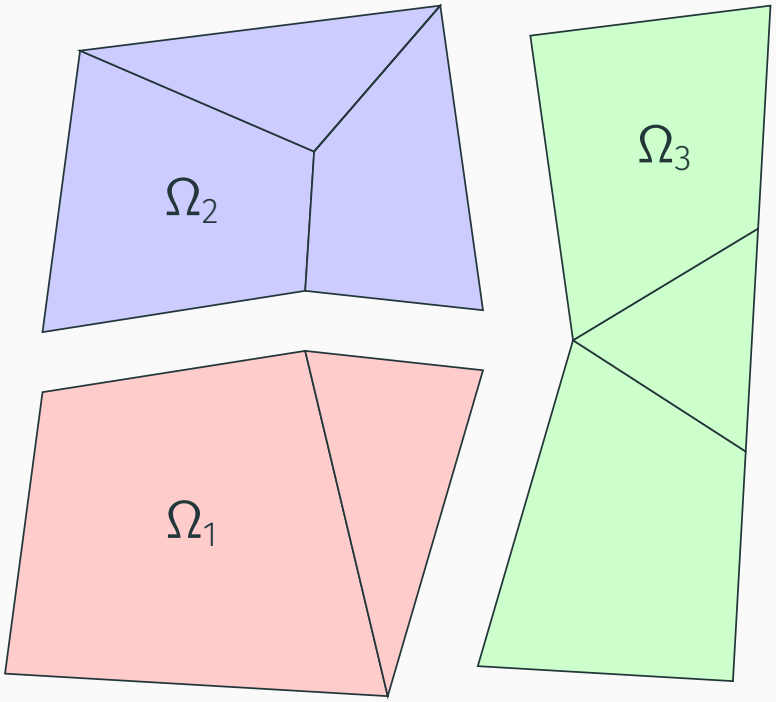
[Gosselet and Rey 2006]



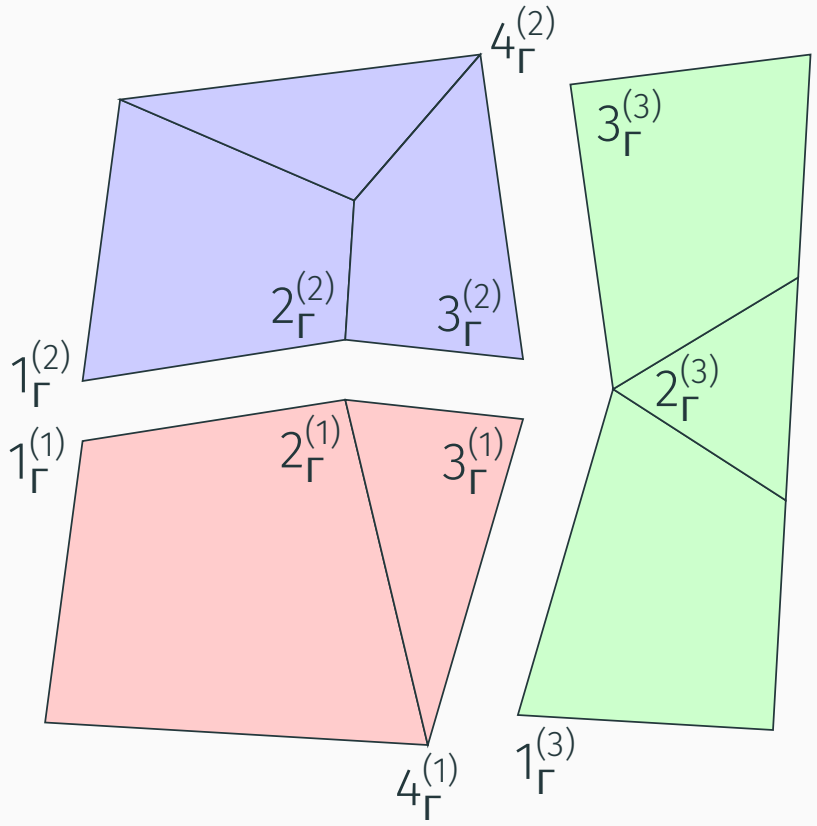
Local numbering

$$A^{(i)} = \begin{bmatrix} A_{ii} & A_{i\Gamma} \\ A_{\Gamma i} & A_{\Gamma\Gamma}^{(i)} \end{bmatrix}$$

General notations



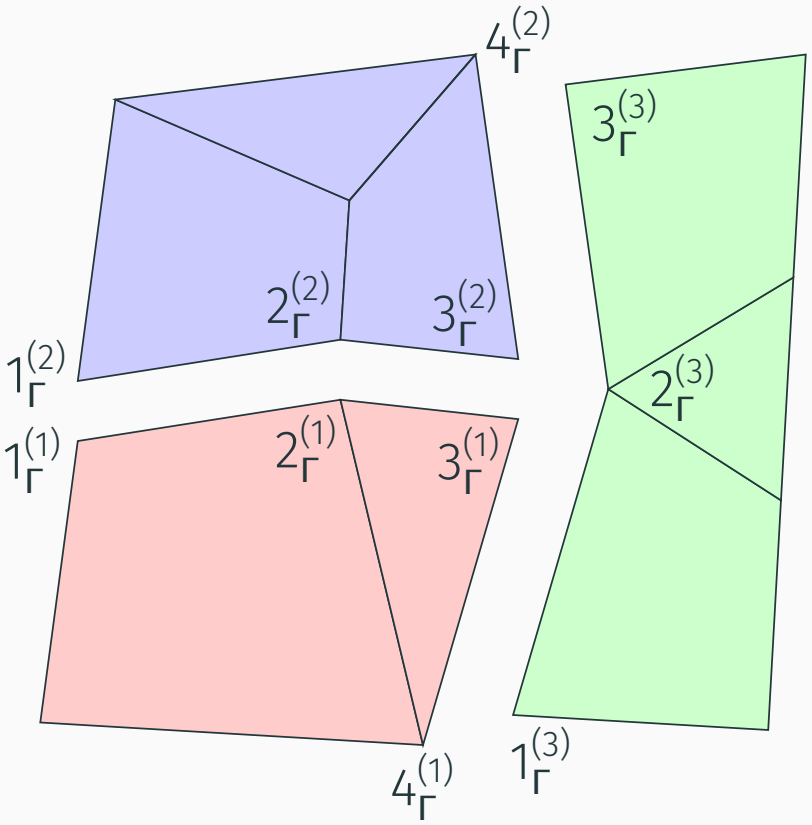
[Gosselet and Rey 2006]



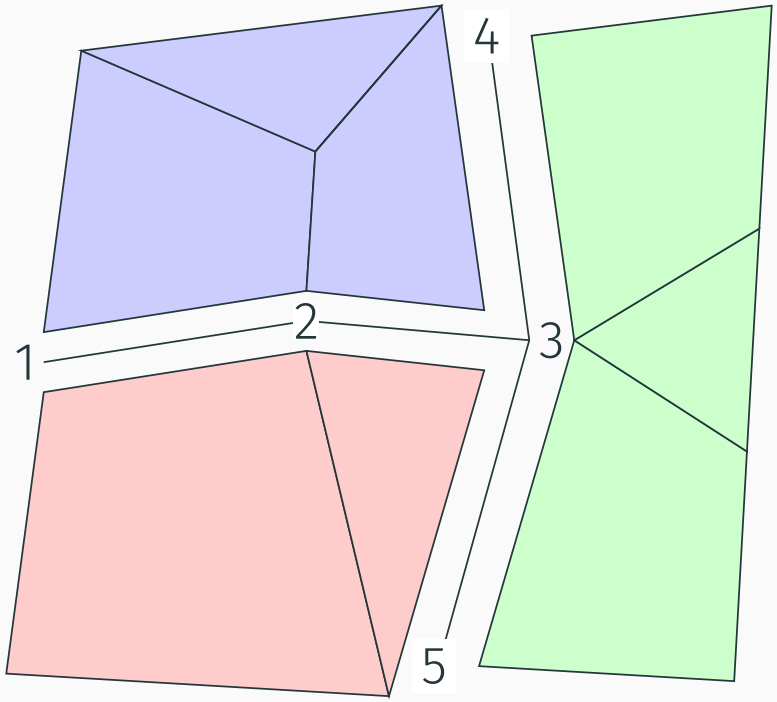
Elimination of interior d.o.f.

$$S_p^{(i)} = A_{\Gamma\Gamma}^{(i)} - A_{\Gamma i} A_{ii}^{-1} A_{i\Gamma}$$

General notations

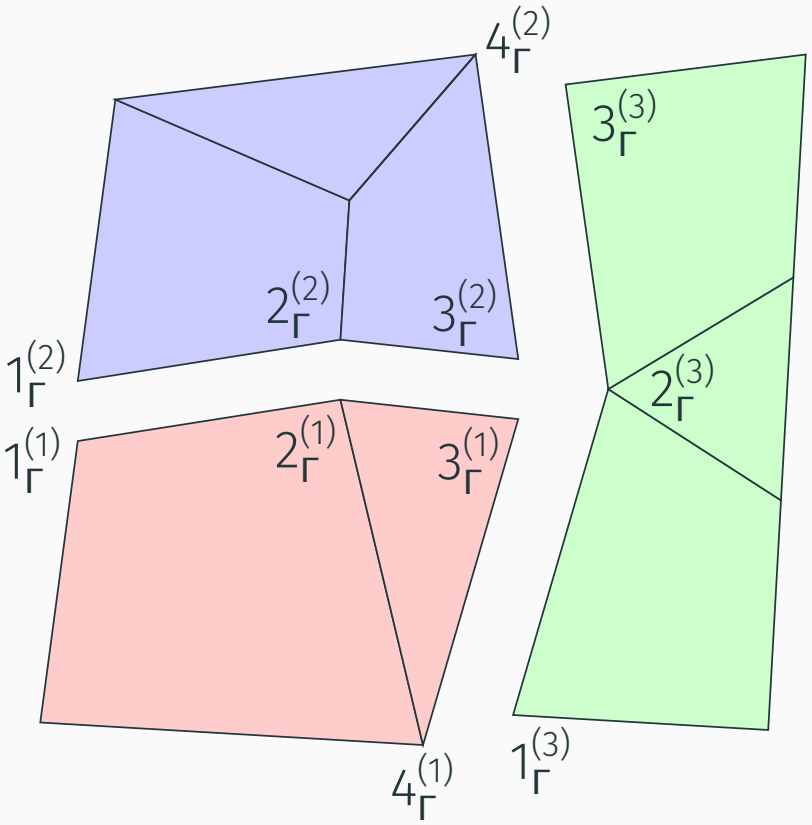


Jump operators $\{B^{(i)}\}_{i=1}^{3}$

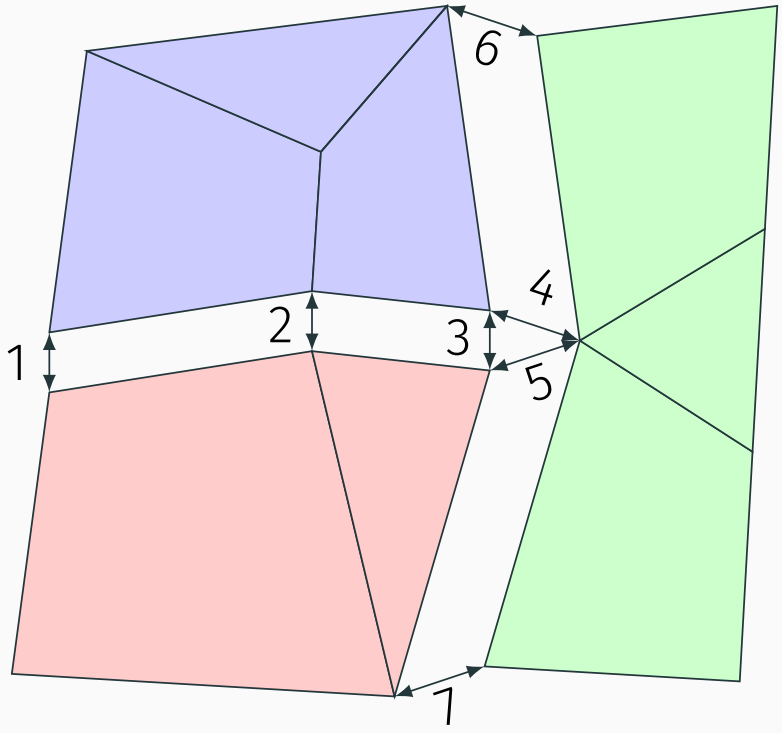


Primal constraints
[Mandel 1993]

General notations



Jump operators $\{\underline{B}^{(i)}\}_{i=1}^{3}$



Dual constraints
[Farhat and Roux 1991]

Condensed system

$$\forall i \in [\![1; N]\!], S_p^{(i)} x_\Gamma^{(i)} = g_\Gamma^{(i)} + \lambda_\Gamma^{(i)}$$

Condensed system

$$\forall i \in [\![1; N]\!], S_p^{(i)} x_\Gamma^{(i)} = g_\Gamma^{(i)} + \lambda_\Gamma^{(i)}$$

$$\sum_{i=1}^{N} \underline{B}^{(i)} x_\Gamma^{(i)} = 0$$

Condensed system

$$\forall i \in [\![1; N]\!], S_p^{(i)} x_\Gamma^{(i)} = g_\Gamma^{(i)} + \lambda_\Gamma^{(i)}$$

$$\sum_{i=1}^{N} \underline{B}^{(i)} x_\Gamma^{(i)} = 0$$

$$\sum_{i=1}^{N} B^{(i)} \lambda_\Gamma^{(i)} = 0$$

Condensed system

$$\forall i \in [\![1; N]\!], S_p^{(i)} x_\Gamma^{(i)} = g_\Gamma^{(i)} + \lambda_\Gamma^{(i)}$$

$$R_\Gamma^{(i)^T} \lambda_\Gamma^{(i)} = 0$$

$$\sum_{i=1}^{N} \underline{B}^{(i)} x_\Gamma^{(i)} = 0$$

$$\sum_{i=1}^{N} B^{(i)} \lambda_\Gamma^{(i)} = 0$$

Unique displacement/eliminated reactions

- unknown $x_\Gamma \implies x_\Gamma^{(i)} = {B^{(i)}}^T x_\Gamma$

Unique displacement/eliminated reactions

- unknown $x_\Gamma \implies x_\Gamma^{(i)} = {B^{(i)}}^T x_\Gamma$
- system of equations

$$\sum_{i=1}^{N} B^{(i)} S_p^{(i)} {B^{(i)}}^T x_\Gamma = \sum_{i=1}^{N} B^{(i)} g_\Gamma^{(i)}$$

Unique displacement/eliminated reactions

- unknown $x_\Gamma \implies x_\Gamma^{(i)} = {B^{(i)}}^T x_\Gamma$
- system of equations

$$\sum_{i=1}^{N} B^{(i)} S_p^{(i)} {B^{(i)}}^T x_\Gamma = \sum_{i=1}^{N} B^{(i)} g_\Gamma^{(i)}$$

- preconditioner

$$M^{-1} = \sum_{i=1}^{N} B^{(i)} D_p^{(i)} {S_p^{(i)}}^\dagger D_p^{(i)} {B^{(i)}}^T$$

Unique displacement/eliminated reactions

- unknown $x_\Gamma \implies x_\Gamma^{(i)} = {B^{(i)}}^T x_\Gamma$
- system of equations

$$\sum_{i=1}^{N} B^{(i)} S_p^{(i)} {B^{(i)}}^T x_\Gamma = \sum_{i=1}^{N} B^{(i)} g_\Gamma^{(i)}$$

- preconditioner

$$M^{-1} = \sum_{i=1}^{N} B^{(i)} D_p^{(i)} {S_p^{(i)}}^\dagger D_p^{(i)} {B^{(i)}}^T$$

applied to vectors in $\mathrm{Im}(S_p)$

Balanced residual

$$\sum_{i=1}^{N} {R_\Gamma^{(i)}}^T D_p^{(i)} {B^{(i)}}^T r_\Gamma = 0$$

Balanced residual

$$\sum_{i=1}^{N} {R_\Gamma^{(i)}}^T D_p^{(i)} {B^{(i)}}^T r_\Gamma = 0$$

Projection

$$\mathsf{R}_\Gamma = \begin{bmatrix} B^{(1)} D_p^{(1)} R_\Gamma^{(1)} & \dots & B^{(N)} D_p^{(N)} R_\Gamma^{(N)} \end{bmatrix}$$

$$\mathsf{S}_p = \sum_{i=1}^{N} B^{(i)} S_p^{(i)} {B^{(i)}}^T$$

Balanced residual

$$\sum_{i=1}^{N} {R_\Gamma^{(i)}}^T D_p^{(i)} {B^{(i)}}^T r_\Gamma = 0$$

Projection

$$\mathbf{R}_\Gamma = \begin{bmatrix} B^{(1)} D_p^{(1)} R_\Gamma^{(1)} & \dots & B^{(N)} D_p^{(N)} R_\Gamma^{(N)} \end{bmatrix}$$

$$\mathbf{S}_p = \sum_{i=1}^{N} B^{(i)} S_p^{(i)} {B^{(i)}}^T$$

$$\Downarrow$$

$$\mathbf{R}_\Gamma^T \mathbf{S}_p P = 0 \quad \text{with } P = I - \mathbf{R}_\Gamma \left(\mathbf{R}_\Gamma^T \mathbf{S}_p \mathbf{R}_\Gamma \right)^{-1} \mathbf{R}_\Gamma^T \mathbf{S}_p$$

Unique reaction/eliminated displacements

- unknown $\lambda_\Gamma \implies \lambda_\Gamma^{(i)} = \underline{B}^{(i)^T} \lambda_\Gamma$
- dual Schur complements $S_d^{(i)} = S_p^{(i)^\dagger}$

Unique reaction/eliminated displacements

- unknown $\lambda_\Gamma \implies \lambda_\Gamma^{(i)} = \underline{B}^{(i)^T} \lambda_\Gamma$
- dual Schur complements $S_d^{(i)} = S_p^{(i)^\dagger}$
- system of equations

$$\forall i \in [\![1; N]\!], x_\Gamma^{(i)} = S_d^{(i)}(g_\Gamma^{(i)} + \underline{B}^{(i)^T}\lambda_\Gamma) + R_\Gamma^{(i)}\alpha^{(i)}$$

$$0 = R_\Gamma^{(i)^T}(g_\Gamma^{(i)} + \underline{B}^{(i)^T}\lambda_\Gamma)$$

Unique reaction/eliminated displacements

- unknown $\lambda_\Gamma \implies \lambda_\Gamma^{(i)} = \underline{B}^{(i)^T} \lambda_\Gamma$
- dual Schur complements $S_d^{(i)} = S_p^{(i)^\dagger}$
- system of equations

$$\forall i \in [\![1; N]\!], x_\Gamma^{(i)} = S_d^{(i)}(g_\Gamma^{(i)} + \underline{B}^{(i)^T}\lambda_\Gamma) + R_\Gamma^{(i)}\alpha^{(i)}$$

$$0 = R_\Gamma^{(i)^T}(g_\Gamma^{(i)} + \underline{B}^{(i)^T}\lambda_\Gamma)$$

- saddle-point formulation

$$\begin{bmatrix} \mathsf{S}_d & \underline{\mathsf{R}}_\Gamma \\ \underline{\mathsf{R}}_\Gamma^T & 0 \end{bmatrix} \begin{bmatrix} \lambda_\Gamma \\ \alpha \end{bmatrix} = \begin{bmatrix} -\mathsf{b}_d \\ -\mathsf{g}_\Gamma \end{bmatrix}$$

Similar utility routines

- exchange
- statistics
- renumber

PETSc

Introduction

- Portable, Extensible Toolkit for Scientific Computation
- suite of data structures
  - `Vec` and `Mat`
  - `PC` and `KSP`
  - `SNES`, `TS`, and `Tao`
- useful for compiling other libraries (MUMPS, *hypre*)
- `https://petsc.org`
- `https://gitlab.com/petsc/petsc`
- interfaced with HPDDM

Application Codes

Higher-Level Libraries and Frameworks

PETSc

TS Time Steppers
- Pseudo-Transient, Runge-Kutta, IMEX, SSP, ...
- Local and Global Error Estimators
- Adaptive Timestepping
- Event Handling
- Sensitivity via Adjoints
- ...

TAO Optimization Solvers
- PDE-Constrained
- Adjoint-Based
- Derivative-Free
- Levenberg-Marquardt
- Newton's Method
- Interior Point Methods
- ...

SNES Nonlinear Solvers
- Newton Linesearch
- Newton Trust Region
- BFGS (Quasi-Newton)
- Nonlinear Gauss-Seidel
- Successive Substitution
- Nonlinear CG
- Active Set VI
- ...

KSP Linear Solvers
- CG, GMRES, BiCGStab, FGMRES, ...
- Pipelined Krylov Methods
- Hierarchical Krylov Methods
- ...

DM Domain Management
- DMDA Regular Grids
- DMPlex Unstructured Meshes
- DMForest Forest-of-octrees AMR
- DMStag Staggered Grids
- DMNetwork Networks
- DMSwarm Particles

PC Preconditioners
- ILU/ICC
- Additive Schwarz
- Fieldsplit (Block Preconditioners)
- PCMG (Geometric Multigrid)
- GAMG (Algebraic Multigrid)
- ...

Vec Vectors

IS Index Sets

Mat Linear Operators
- AIJ (Compressed Sparse Row)
- SAIJ (Symmetric)
- BAIJ (Blocked)
- Dense
- GPU Matrices
- ...

PetscSF Parallel Communication

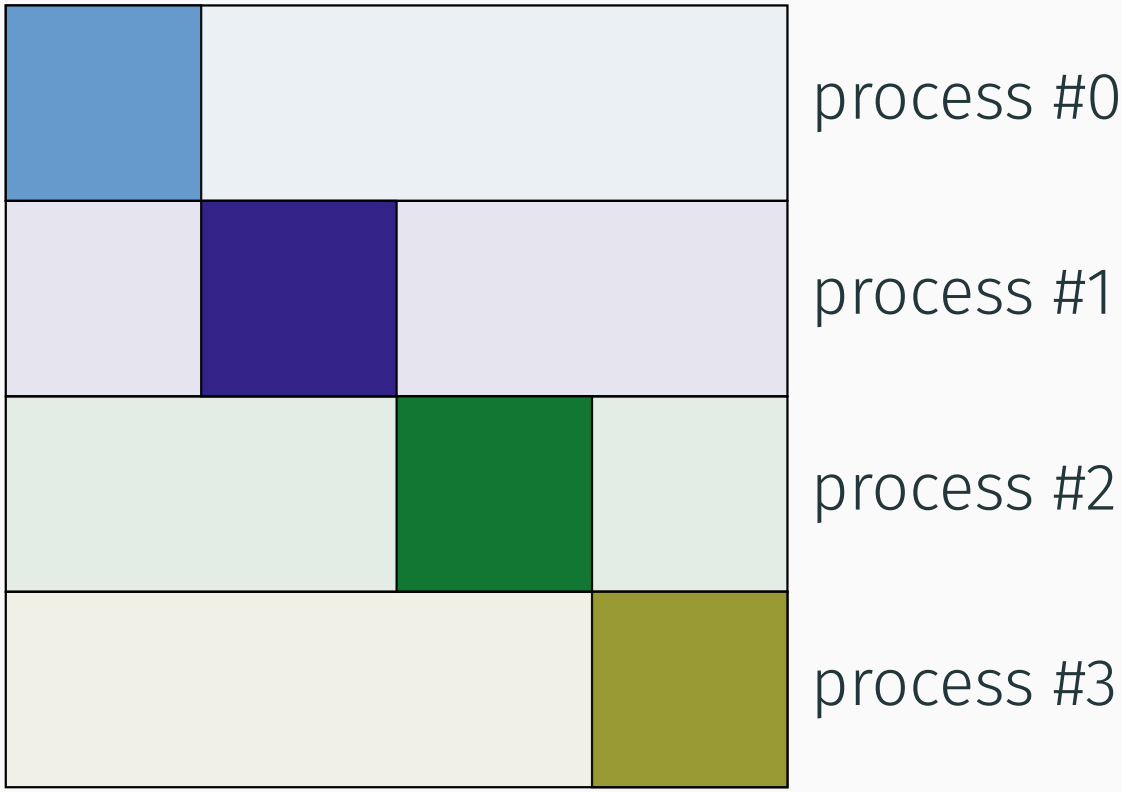
SLEPc Eigensolvers

Communication and Computational Kernels

MPI | BLAS/LAPACK | Kokkos | CUDA | . . .

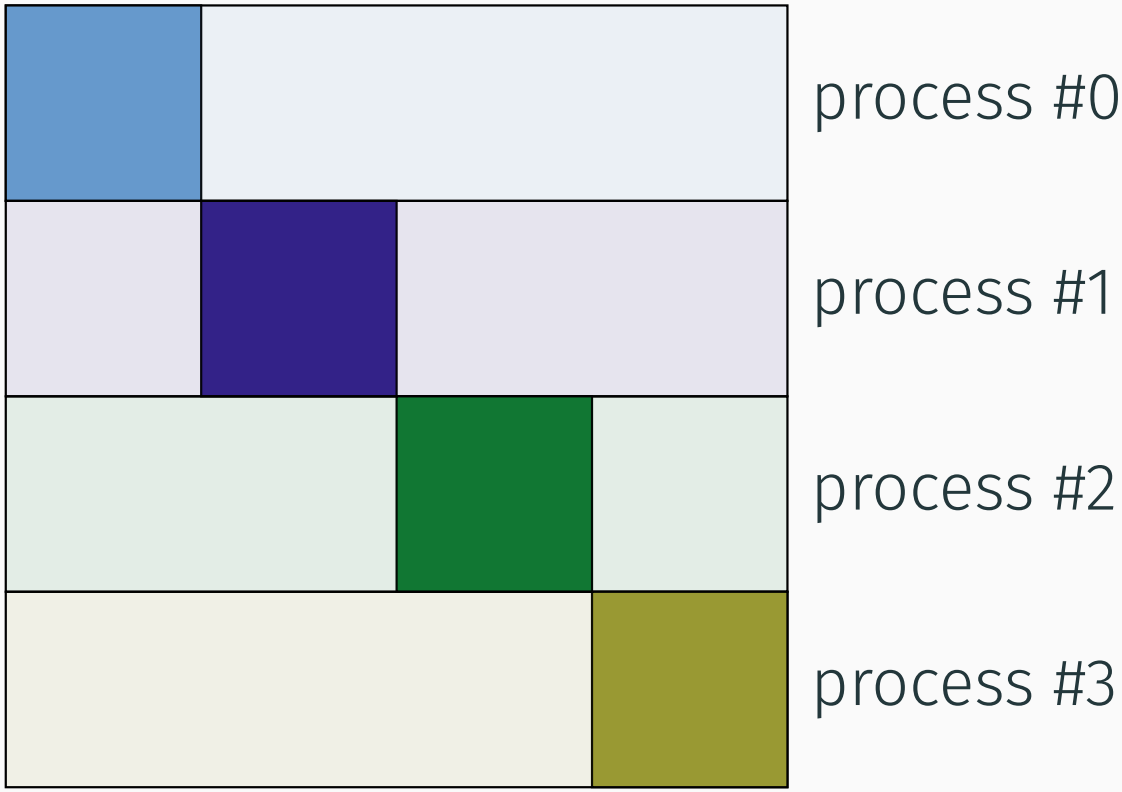
Data distribution

Operators follow a 1D row-wise contiguous distribution



Data distribution

Operators follow a 1D row-wise contiguous distribution



$\Longrightarrow$ accessible via `GlobalNumbering`

Setting up a `Mat`

- same input parameters as HPDDM types
- simplified macro `MatCreate(Th,A,Pk)`
- `Mat<complex>` for complex-valued problems

Setting up a `Mat`

- same input parameters as HPDDM types
- simplified macro `MatCreate(Th,A,Pk)`
- `Mat<complex>` for complex-valued problems

Switching between numberings

- `ChangeNumbering(Mat<K>,K[int])`
- optional parameters
  - `inverse` to go from PETSc to FreeFEM
  - `exchange` to update ghost values

`Mat` LINEAR OPERATIONS [EXAMPLE2.EDP]

Just as with FreeFEM `matrix`

- matrix–vector product `A * x`
- matrix transpose–vector product `A' * x`
- linear solve `A^-1 * x`

  transposed linear solve `A'^-1 * x`

- native operations with vectors in PETSc numbering
  - `KSPSolve`
  - `MatMatMult`
  - more at `Mat` manual pages

Mat LINEAR OPERATIONS [EXAMPLE2.EDP]

Just as with FreeFEM `matrix`

- matrix–vector product `A * x`
- matrix transpose–vector product `A' * x`
- linear solve `A^-1 * x`
- Hermitian transposed linear solve `A'^-1 * x`

- native operations with vectors in PETSc numbering
  - `KSPSolve`
  - `MatMatMult`
  - more at `Mat` manual pages

Basic preconditioning [example3.edp]

- default to BJacobi with $ILU(0)$ as a subdomain solver
- attach a KSP using `set` and `sparams`
- `-ksp_type`, `-pc_type`
- `-help` generated dynamically

Updating a `Mat`

- `Mat = matrix` if same pattern or first update
- `Mat = Mat` if different pattern

Direct factorizations

- `-pc_type` [lu|cholesky]
- `-pc_factor_mat_solver_type` [mumps|superlu]
- `-help` for fine-tuning a solver
- e.g., `-mat_mumps_icntl_4` 2

Schwarz methods

- `-pc_type` [asm|gasm]
- `-pc_asm_overlap` *n*
- `-pc_asm_type` [basic|restrict|interpolate|none]
- `-sub_pc_type`, `-sub_ksp_type`
- more at PCASM manual page

Example 4: Liouville–Bratu–Gelfand equation

Aim

Solve the nonlinear PDE

$$\Delta u + \lambda \exp^u = 0 \quad \text{in } \Omega$$

$$u(x, y) = \cos \pi x \cos \pi y \quad \text{on } \Gamma$$

Example 4: Liouville–Bratu–Gelfand equation

Aim

Solve the nonlinear PDE

$$\Delta u + \lambda \exp^u = 0 \quad \text{in } \Omega$$

$$u(x, y) = \cos \pi x \cos \pi y \quad \text{on } \Gamma$$

Newton method

$$u^{n+1} = u^n + w \quad \text{such that} \quad \mathcal{F}(u^n) + \mathrm{d}_w \mathcal{F}(u^n) = 0$$

$$\text{with} \quad \mathrm{d}_w \mathcal{F}(u^n) = \Delta w + \lambda \exp^{u^n} w$$

[example4_seq.edp] + [example4.edp]

Block matrices

Limitations of monolithic formulations

- single `varf`
- single `fespace`

Block matrices

Limitations of monolithic formulations

- single `varf`
- single `fespace`

Alternative

- “decoupled” `fespace` and multiple `varf`
- PETSc `MatNest` to match FreeFEM block syntax

Block matrices

Limitations of monolithic formulations

- single `varf`
- single `fespace`

Alternative

- "decoupled" `fespace` and multiple `varf`
- PETSc `MatNest` to match FreeFEM block syntax

Solving linear systems with `MatNest`

- no explicit representation $\Rightarrow$ no *LU*, BJacobi
- automatic convertion or `MatConvert`

Example 5: Poisson eq. with Neumann BC [example5.edp]

Aim

- indefinite system with no essential BC
- additional constraint $\int_\Omega u = 0$

Example 5: Poisson eq. with Neumann BC [example5.edp]

Aim

- indefinite system with no essential BC
- additional constraint $\int_\Omega u = 0$

$$\begin{bmatrix} A & c \\ c^T & 0 \end{bmatrix} \begin{bmatrix} u \\ \lambda \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}$$

- written as `Mat N = [[A,c],[c',0]]`
- `A` is a `Mat` and `c` follows PETSc numbering
- use either `^-1` or `MatConvert` + `KSPSolve`
- λ stored only on the process with the lowest rank

Rectangular matrices [example6.edp]

- some FreeFEM `matrix` are not square
- `matrix Loc = varf(Ph, Vh)` // Loc.n = Vh.ndof
- coupling 2D and 3D problems

Rectangular matrices [example6.edp]

- some FreeFEM `matrix` are not square
- `matrix Loc = varf(Ph, Vh)` // Loc.n = Vh.ndof
- coupling 2D and 3D problems

`fespaces` defined with the same partitioning

- one square `Mat A` for distributing `Vh`
- one square `Mat B` for distributing `Ph`
- one `Mat C(A,B,Loc)`

$\Longrightarrow$ `DmeshCreate(Th)` + `MatCreate(A|B)`

Example 7: Stokes equation [example7.edp]

Aim

Solve with a Poiseuille inflow the system

$$\begin{aligned} -\Delta u + \nabla p &= 0 \quad \text{in } \Omega \\ \nabla \cdot u &= 0 \end{aligned}$$

Example 7: Stokes equation [example7.edp]

Aim

Solve with a Poiseuille inflow the system

$$\begin{aligned} -\Delta u + \nabla p &= 0 \quad \text{in } \Omega \\ \nabla \cdot u &= 0 \end{aligned}$$

- two `varf`s
- two `fespace` distributions
- two assembled `Mat`s `A` and `B`
- coupled system `Mat N = [[A,B],[B',0]]`
- transposed operators not formed explicitly

Multigrid methods

- point-wise aggregation
  - *hypre* (--download-hypre) [Falgout and Yang 2002]
  - AMS [Hiptmair and Xu 2007]
- smoothed aggregation
  - GAMG [Adams et al. 2004]
  - ML (--download-ml) [Gee et al. 2006]

Systems of equations

- matrix block size
- `MatNullSpace`, e.g., rigid body modes

Example 8: system of linear elasticity [example8.edp]

Aim

Solve the system

$$-\operatorname{div}\sigma = f \quad \text{in } \Omega$$

$$u = 0 \quad \text{on } \Gamma_D$$

Example 8: system of linear elasticity [example8.edp]

Aim

Solve the system

$$-\operatorname{div}\sigma = f \quad \text{in } \Omega$$
$$u = 0 \quad \text{on } \Gamma_D$$

Tools

- vectorial `fespace`
- block size of three
- `MatNullSpace` with the rigid body modes
  - translations $(1,0,0)$ $(0,1,0)$ $(0,0,1)$
  - rotations $(y,-x,0)$ $(-z,0,x)$ $(0,z,-y)$

Example 9: geometric multigrid [example9.edp]

Aim

Solve the complex-valued system

$$\operatorname{curl}\operatorname{curl} E - k^2 E = f \quad \text{in } \Omega$$

$$(\operatorname{curl} E) \times n - \mathrm{i}k(E \times n) \times n = 0 \quad \text{on } \Gamma_R$$

Example 9: geometric multigrid [example9.edp]

Aim

Solve the complex-valued system

$$\begin{aligned} \operatorname{curl}\operatorname{curl} E - k^2 E &= f \quad \text{in } \Omega \\ (\operatorname{curl} E) \times n - \mathrm{i}k(E \times n) \times n &= 0 \quad \text{on } \Gamma_R \end{aligned}$$

Tools

- Nédélec edge elements
- Schwarz smoothers
- `buildMatEdgeRecursive`
- more at PCMG manual page

Fieldsplit preconditioners [example10.edp]

- separate preconditioners for each field
- unknowns are interleaved
- underlying `IS`
- `fespace Vh(Pk,Pq,Pt)` ⇒ `Vh [u,v,w] = [1,2,3]`
- new option prefixes `-fieldsplit_%d_`

Optional parameters

- prefixes customizable with a `string[int]`
- approximate Schur complement

Fieldsplit preconditioners [example10.edp]

- separate preconditioners for each field
- unknowns are interleaved
- underlying `IS`
- `fespace Vh(Pk,Pk,Pq)` ⇒ `Vh [u,v,p] = [1,1,2]`
- new option prefixes `-fieldsplit_%d_`

Optional parameters

- prefixes customizable with a `string[int]`
- approximate Schur complement

Examples for Stokes equations [Knepley 2013]

$$A = \begin{bmatrix} C & B \\ B^T & 0 \end{bmatrix} \approx \mathrm{pc}_A = \begin{bmatrix} C^{-1} & 0 \\ 0 & I \end{bmatrix}$$

-ksp_type gmres -pc_fieldsplit_type additive
-fieldsplit_velocity_pc_type lu
-fieldsplit_pressure_pc_type jacobi

Fieldsplit preconditioners [example10.edp]

- separate preconditioners for each field
- unknowns are interleaved
- underlying `IS`
- `fespace Vh(Pk,Pk,Pq)` ⇒ `Vh [u,v,p] = [1,1,2]`
- new option prefixes `-fieldsplit_%d_`

Optional parameters

- prefixes customizable with a `string[int]`
- approximate Schur complement

Examples for Stokes equations [Knepley 2013]

$$A = \begin{bmatrix} C & B \\ B^T & 0 \end{bmatrix} \approx \mathrm{pc}_A = \begin{bmatrix} \mathrm{ksp}_C & 0 \\ 0 & I \end{bmatrix}$$

-ksp_type fgmres -pc_fieldsplit_type additive
-fieldsplit_velocity_pc_type gamg
-fieldsplit_pressure_pc_type jacobi

Fieldsplit preconditioners [example10.edp]

- separate preconditioners for each field
- unknowns are interleaved
- underlying `IS`
- `fespace Vh(Pk,Pk,Pq)` ⇒ `Vh [u,v,p] = [1,1,2]`
- new option prefixes `-fieldsplit_%d_`

Optional parameters

- prefixes customizable with a `string[int]`
- approximate Schur complement

Examples for Stokes equations [Knepley 2013]

$$A = \begin{bmatrix} C & B \\ B^T & 0 \end{bmatrix} \approx \mathrm{pc}_A = \begin{bmatrix} \mathrm{ksp}_C & B \\ 0 & I \end{bmatrix}$$

-ksp_type fgmres -pc_fieldsplit_type multiplicative
-fieldsplit_velocity_pc_type gamg
-fieldsplit_pressure_pc_type jacobi

Fieldsplit preconditioners [example10.edp]

- separate preconditioners for each field
- unknowns are interleaved
- underlying `IS`
- `fespace Vh(Pk,Pk,Pq)` ⇒ `Vh [u,v,p] = [1,1,2]`
- new option prefixes `-fieldsplit_%d_`

Optional parameters

- prefixes customizable with a `string[int]`
- approximate Schur complement

Examples for Stokes equations [Knepley 2013]

$$A = \begin{bmatrix} C & B \\ B^T & 0 \end{bmatrix} \approx \mathrm{pc}_A = \begin{bmatrix} \mathrm{ksp}_C & 0 \\ 0 & -\mathrm{ksp}_S \end{bmatrix}$$

-ksp_type fgmres -pc_fieldsplit_type schur
-fieldsplit_velocity_pc_type gamg
-fieldsplit_pressure_pc_type jacobi
-pc_fieldsplit_schur_factorization_type diag

Fieldsplit preconditioners [example10.edp]

- separate preconditioners for each field
- unknowns are interleaved
- underlying `IS`
- `fespace Vh(Pk,Pk,Pq)` ⇒ `Vh [u,v,p] = [1,1,2]`
- new option prefixes `-fieldsplit_%d_`

Optional parameters

- prefixes customizable with a `string[int]`
- approximate Schur complement

Examples for Stokes equations [Knepley 2013]

$$A = \begin{bmatrix} C & B \\ B^T & 0 \end{bmatrix} \approx \mathrm{pc}_A = \begin{bmatrix} \mathrm{ksp}_C & 0 \\ B^T & \mathrm{ksp}_S \end{bmatrix}$$

-ksp_type fgmres -pc_fieldsplit_type schur
-fieldsplit_velocity_pc_type gamg
-fieldsplit_pressure_pc_type jacobi
-pc_fieldsplit_schur_factorization_type lower

Fieldsplit preconditioners [example10.edp]

- separate preconditioners for each field
- unknowns are interleaved
- underlying `IS`
- `fespace Vh(Pk,Pk,Pq)` ⇒ `Vh [u,v,p] = [1,1,2]`
- new option prefixes `-fieldsplit_%d_`

Optional parameters

- prefixes customizable with a `string[int]`
- approximate Schur complement

Examples for Stokes equations [Knepley 2013]

$$A = \begin{bmatrix} C & B \\ B^T & 0 \end{bmatrix} \approx \mathrm{pc}_A = \begin{bmatrix} C^{-1} & B \\ 0 & \mathrm{ksp}_S \end{bmatrix}$$

-ksp_type gmres -fieldsplit_pressure_ksp_max_its 1
-pc_fieldsplit_type schur -fieldsplit_velocity_pc_type
lu -fieldsplit_pressure_ksp_type richardson
-pc_fieldsplit_schur_factorization_type upper

Matrix-free operators

`MatShell`

- user-provided routines (`MatMult`, `MatMultTranspose`)
- (user-provided) preconditioning less intuitive
- `func` passed to PETSc

Matrix-free operators

`MatShell`

- user-provided routines (`MatMult`, `MatMultTranspose`)
- (user-provided) preconditioning less intuitive
- `func` passed to PETSc

- must use PETSc numbering
- KSP is oblivious to a `Mat` type
- underlying PCSHELL

Example 11: 1D finite differences [example11.edp]

Aim

Second-order centered scheme

$$n^2 \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & & -1 & 2 \end{bmatrix} x = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

Example 11: 1D finite differences [example11.edp]

Aim

Second-order centered scheme

$$n^2 \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & & -1 & 2 \end{bmatrix} x = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

- useful monitoring, e.g., `-ksp_view_singularvalues`
- variable `precon` to supply a PCSHELL
- 1D `meshL`

Nonlinear solvers SNES

- easy-to-use interface to (quasi-)Newton methods
- solve $\mathcal{F}(u) = b$
- handle variational inequalities on u
- must provide two `func`s
  - to evaluate residuals stored as `K[int]`
  - to update a Jacobian stored as `Mat`
- `ChangeNumbering` to go from PETSc to FreeFEM vectors

Nonlinear solvers SNES

- easy-to-use interface to (quasi-)Newton methods
- solve $\mathcal{F}(u) = b$
- handle variational inequalities on u
- must provide two `func`s
  - to evaluate residuals stored as `K[int]`
  - to update a Jacobian stored as `Mat`
- `ChangeNumbering` to go from PETSc to FreeFEM vectors

$\Rightarrow$ solver for the linearized systems configured via `set`

Example 12: Newton method [example12.edp]

Aim

Solve

$$\nabla J(u) = 0 \quad \text{in } \Omega$$

$$\text{with } J(u) = \int_\Omega \frac{1}{2} f\left(||\nabla u||^2\right) - cu$$

$$\text{subject to } u_{\text{lower}} \leqslant u \leqslant u_{\text{upper}} \text{ in } \mathcal{C}$$

and $f : x \mapsto (1+a)x - \log(1+x)$

Example 12: Newton method [example12.edp]

Aim

Solve

$$\nabla J(u) = 0 \quad \text{in } \Omega$$

$$\text{with } J(u) = \int_\Omega \frac{1}{2} f\left(||\nabla u||^2\right) - cu$$

$$\text{subject to } u_{\text{lower}} \leqslant u \leqslant u_{\text{upper}} \text{ in } \mathcal{C}$$

and $f : x \mapsto (1+a)x - \log(1+x)$

- compute constant term from $\mathcal{F}(u) = b$
- assemble residuals and update the Jacobian
- additional parameters in `SNESSolve` for the bounds
- no automatic differentiation

Example 13: solution reconstruction [example13.edp]

Aim

Distributed local functions to a global function

- mesh adaptation
- centralized postprocessing

Example 13: solution reconstruction [example13.edp]

Aim

Distributed local functions to a global function

- mesh adaptation
- centralized postprocessing

- `DmeshCreate` additional macro `N2O`
- + `restrict` to go from local to global
- global reduction needed to sum contributions

Example 14: systems with multiple RHS [example14.edp]

Aim

Solve the complex-valued system

$$\Delta u + k^2 u = f_i \quad \text{in } \Omega$$

$$u \cdot n = 0 \quad \text{on } \Gamma$$

with multiple point sources $\{f_i\}_{i=1,2,\ldots}$

Example 14: systems with multiple RHS [example14.edp]

Aim

Solve the complex-valued system

$$\Delta u + k^2 u = f_i \quad \text{in } \Omega$$
$$u \cdot n = 0 \quad \text{on } \Gamma$$

with multiple point sources $\{f_i\}_{i=1,2,\ldots}$

- `KSPSolve` + `complex[int,int]`
- use PETSc numbering
- single solve with an exact factorization
- block GMRES from HPDDM

End-user perspectives

- timesteppers TS manual pages
- optimizers Tao manual pages
- easy composability between PETSc objects

SLEPc

Introduction

- Scalable Library for Eigenvalue Problem Computations
- built on top of PETSc
- suite of additional data structures
  - EPS
  - ST
- useful for compiling other libraries (ARPACK)
- `http://slepc.upv.es`
- `https://gitlab.com/slepc/slepc`

Interface

Parameters

- one `Mat` or two for generalized eigenproblems
- arrays for eigenvalues/eigenvectors
  - FreeFEM numbering
  - PETSc numbering for `MatNest`
- careful with --with-scalar-type

Interface

Parameters

- one `Mat` or two for generalized eigenproblems
- arrays for eigenvalues/eigenvectors
  - FreeFEM numbering
  - PETSc numbering for `MatNest`
- careful with --with-scalar-type

Interface

Parameters

- one `Mat` or two for generalized eigenproblems
- arrays for eigenvalues/eigenvectors
  - FreeFEM numbering
  - PETSc numbering for `MatNest`
- careful with --with-scalar-type

Spectral transformations

- improve convergence for some eigenproblems
- underlying KSP configured via `set`

Miscellaneous

Matrix-free operators

- just as with `KSPSolve` with a `MatShell`
- PETSc numbering
- limited number of `ST`

Miscellaneous

Matrix-free operators

- just as with `KSPSolve` with a `MatShell`
- PETSc numbering
- limited number of `ST`

Periodic boundary conditions

Custom partitioning that may be imbalanced

Miscellaneous

Matrix-free operators

- just as with `KSPSolve` with a `MatShell`
- PETSc numbering
- limited number of `ST`

Periodic boundary conditions

Custom partitioning that may be imbalanced

HPDDM Krylov methods

- `-ksp_type` hpddm
- `-st_ksp_type` hpddm

Example 1: Steklov–Poincaré operator [example1.edp]

Aim

Find the eigenvectors of the operator

$$\mathrm{DtN}: \Gamma_N \to \mathbb{R}$$

$$g \mapsto \partial_n v$$

where v satisfies

$$-\Delta v = 0 \text{ in } \Omega$$

$$v = 0 \text{ on } \Gamma_D$$

$$v = g \text{ on } \Gamma_N$$

Example 1: Steklov–Poincaré operator [example1.edp]

Aim

Find the eigenvectors of the operator

$$\begin{aligned} \mathrm{DtN} : \Gamma_N &\to \mathbb{R} \\ g &\mapsto \partial_n v \end{aligned}$$

where v satisfies

$$\begin{aligned} -\Delta v &= 0 \text{ in } \Omega \\ v &= 0 \text{ on } \Gamma_D \\ v &= g \text{ on } \Gamma_N \end{aligned}$$

- equivalent to finding (λ, u) s.t. $Au = \lambda Bu$
- shift-and-invert spectral transformation
- same `Mat` distribution for A and B using `Mat B(A,Loc)`

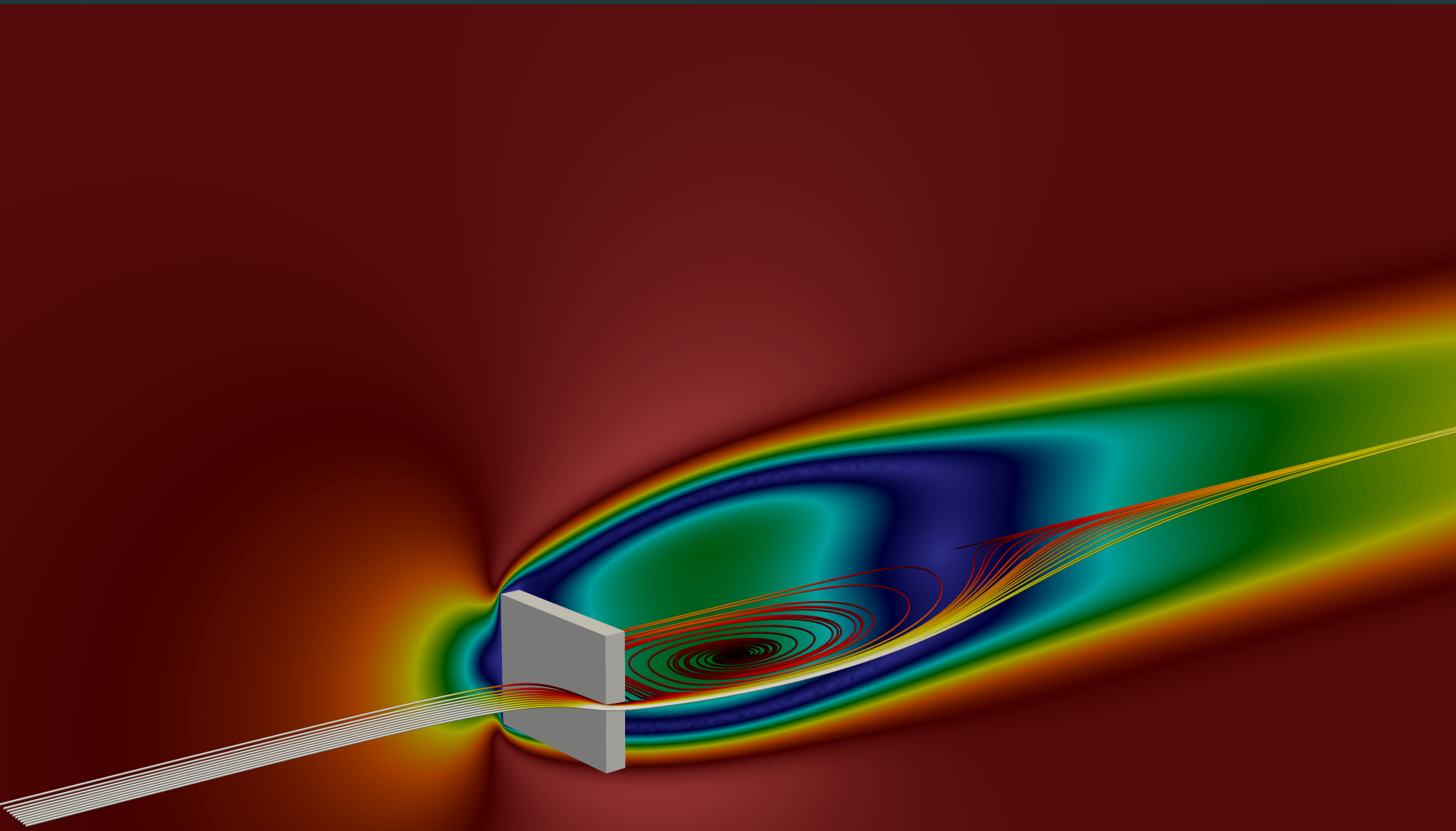
Applications

Augmented Lagrangian for stability analysis

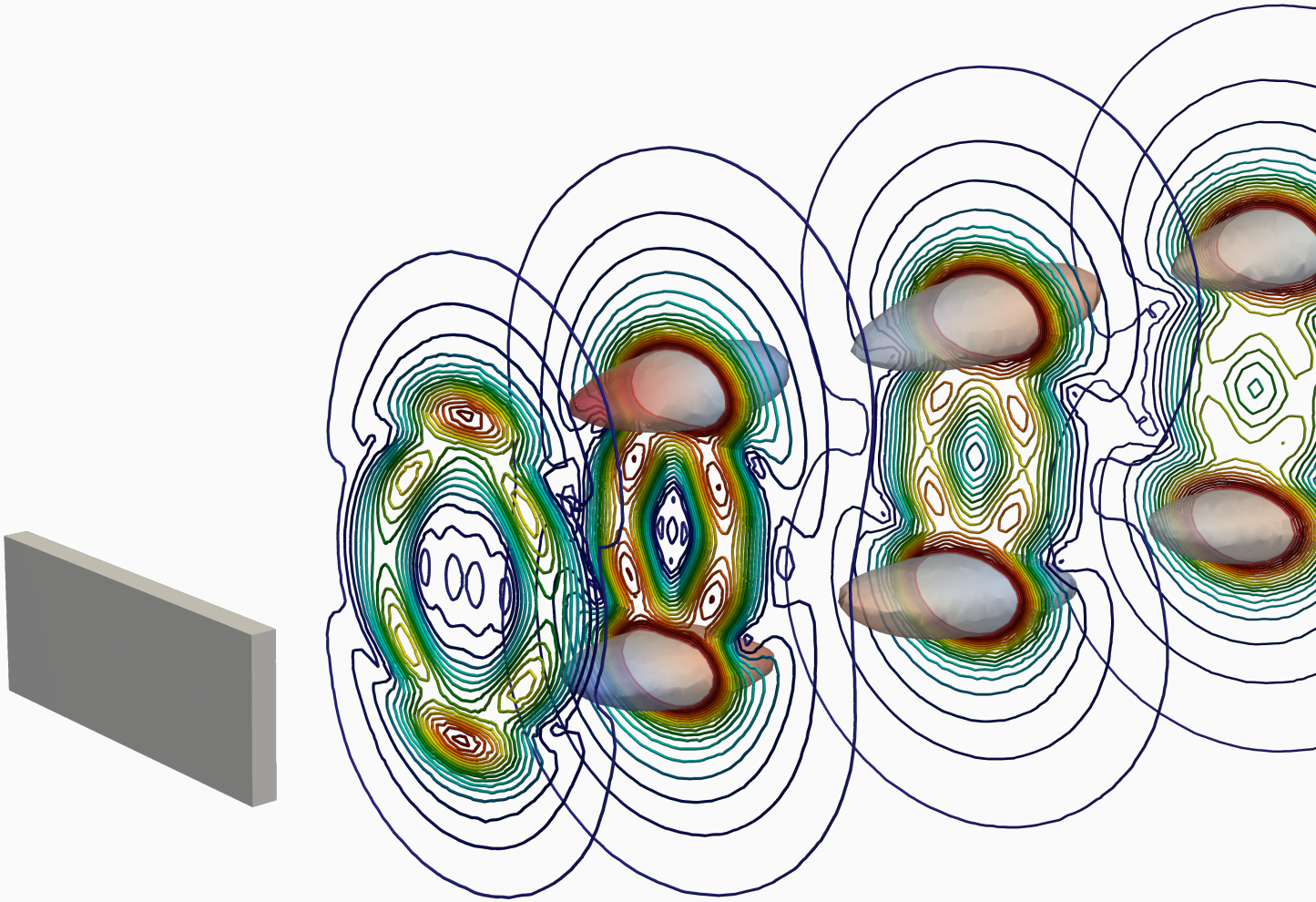
[Moulin et al. 2019]

- `https://github.com/prj-/moulin2019al`
- Nonlinear-solver.edp
  - `-gamma` (0.1)
  - `-mesh` (FlatPlate3D.mesh)
  - `-Re` (50)
- Eigensolver.edp
  - `-shift_real` (10^{-6})
  - `-shift_imag` (0.6)
  - `-nev` (5)
  - `-recycle` (0)

Baseflow



Leading unstable eigenvector



Thank you!

Questions?

Optimal compilation process I

- remove any previous instance of FreeFEM
- install gcc/clang and gfortran
- make sure you have a working MPI implementation

```
> export FF_DIR=${PWD}/FreeFem-sources
> export PETSC_DIR=${PWD}/petsc
> export PETSC_ARCH=arch-FreeFem
> export PETSC_VAR=${PETSC_DIR}/${PETSC_ARCH}
```

Optimal compilation process II


```
> git clone https://github.com/FreeFem/FreeFem-sources
> git clone https://gitlab.com/petsc/petsc
```

Optimal compilation process III


```
> cd ${PETSC_DIR} && ./configure --download-mumps
  --download-parmetis --download-metis
  --download-hypre --download-superlu
  --download-slepc --download-hpddm
  --download-ptscotch --download-suitesparse
  --download-scalapack --download-tetgen
  --download-mmg --download-parmmg
  --with-fortran-bindings=no --with-scalar-type=real
  --with-debugging=no
> make
```

Optimal compilation process IV

compilation for complex scalars, optional

```
> export PETSC_ARCH=arch-FreeFem-complex
> ./configure --with-mumps-dir=arch-FreeFem
  --with-parmetis-dir=arch-FreeFem
  --with-metis-dir=arch-FreeFem
  --with-superlu-dir=arch-FreeFem --download-slepc
  --download-hpddm --download-htool
  --with-ptscotch-dir=arch-FreeFem
  --with-suitesparse-dir=arch-FreeFem
  --with-scalapack-dir=arch-FreeFem
  --with-tetgen-dir=arch-FreeFem
  --with-fortran-bindings=no
  --with-scalar-type=complex --with-debugging=no
> make
```

Optimal compilation process V

- > cd ${FF_DIR}
- > git checkout develop
- > autoreconf -i
- > ./configure --without-hdf5
  --with-petsc=${PETSC_VAR}/lib
  --with-petsc_complex=${PETSC_VAR}-complex/lib
  - ◦ if PETSc is not detected, overwrite MPIRUN
- > make -j4
- > export PATH=${PATH}:${FF_DIR}/src/mpi
- > export PATH=${PATH}:${FF_DIR}/src/nw
  - ◦ setup ~/.freefem++.pref or define FF_INCLUDEPATH and FF_LOADPATH

References I


Adams, Mark, Harun H. Bayraktar, Tony M. Keaveny, and Panayiotis Papadopoulos (2004). **"Ultrascalable Implicit Finite Element Analyses in Solid Mechanics with over a Half a Billion Degrees of Freedom".** In: *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*. SC04. IEEE Computer Society, 34:1–34:15.


Al Daas, Hussam, Laura Grigori, Pierre Jolivet, and Pierre-Henri Tournier (2021). **"A multilevel Schwarz preconditioner based on a hierarchy of robust coarse spaces".** In: *SIAM Journal on Scientific Computing* 43.3, A1907–A1928. URL: `https://github.com/prj-/aldaas2019multi`.


Amestoy, Patrick, Iain Duff, Jean-Yves L'Excellent, and Jacko Koster (2001). **"A fully asynchronous multifrontal solver using distributed dynamic scheduling".** In: *SIAM Journal on Matrix Analysis and Applications* 23.1, pp. 15–41. URL: `https://mumps-solver.org`.


Balay, Satish, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith (1997). **"Efficient management of parallelism in object-oriented numerical software libraries".** In: *Modern Software Tools in Scientific Computing*, pp. 163–202.

References II


Cai, Xiao-Chuan, Maksymilian Dryja, and Marcus Sarkis (2003). **"Restricted additive Schwarz preconditioners with harmonic overlap for symmetric positive definite linear systems".** In: *SIAM Journal on Numerical Analysis* 41.4, pp. 1209–1231.


St-Cyr, Amik, Martin J. Gander, and Stephen Thomas (2007). **"Optimized multiplicative, additive, and restricted additive Schwarz preconditioning".** In: *SIAM Journal on Scientific Computing* 29.6, pp. 2402–2425.


Falgout, Robert D. and Ulrike Meier Yang (2002). **"*hypre*: A library of high performance preconditioners".** In: *Computational Science—ICCS 2002*, pp. 632–641.


Farhat, Charbel and François-Xavier Roux (1991). **"A method of finite element tearing and interconnecting and its parallel solution algorithm".** In: *International Journal for Numerical Methods in Engineering* 32.6, pp. 1205–1227.


Gee, Michael W., Christopher M. Siefert, Jonathan J. Hu, Ray S. Tuminaro, and Marzio G. Sala (2006). ***ML 5.0 Smoothed Aggregation User's Guide.*** Tech. rep. SAND2006-2649. Sandia National Laboratories. URL: `https://trilinos.github.io/ml.html`.

References III


Geuzaine, Christophe and Jean-François Remacle (2009). **"Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities".** In: *International Journal for Numerical Methods in Engineering* 79.11, pp. 1309–1331. URL: `http://geuz.org/gmsh`.


Gosselet, Pierre and Christian Rey (2006). **"Non-overlapping domain decomposition methods in structural mechanics".** In: *Archives of Computational Methods in Engineering* 13.4, pp. 515–572.


Haferssas, Ryadh, Pierre Jolivet, and Frédéric Nataf (2017). **"An additive Schwarz method type theory for Lions's algorithm and a symmetrized optimized restricted additive Schwarz method".** In: *SIAM Journal on Scientific Computing* 39.4, A1345–A1365.


Hecht, Frédéric (2012). **"New development in FreeFem++".** In: *Journal of Numerical Mathematics* 20.3-4, pp. 251–266.


Hénon, Pascal, Pierre Ramet, and Jean Roman (2002). **"PaStiX: a high-performance parallel direct solver for sparse symmetric positive definite systems".** In: *Parallel Computing* 28.2, pp. 301–321. URL: `http://pastix.gforge.inria.fr`.

References iv


Hernandez, Vicente, Jose E. Roman, and Vicente Vidal (2005). **"SLEPc: a scalable and flexible toolkit for the solution of eigenvalue problems".** In: *ACM Transactions on Mathematical Software* 31.3, pp. 351–362. URL: https://slepc.upv.es.


Hiptmair, Ralf and Jinchao Xu (2007). **"Nodal auxiliary space preconditioning in *H*(curl) and *H*(div) spaces".** In: *SIAM Journal on Numerical Analysis* 45.6, pp. 2483–2509.


Jolivet, Pierre, Frédéric Hecht, Frédéric Nataf, and Christophe Prud'homme (2013). **"Scalable Domain Decomposition Preconditioners for Heterogeneous Elliptic Problems".** In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC13*. ACM.


Jolivet, Pierre and Pierre-Henri Tournier (2016). **"Block iterative methods and recycling for improved scalability of linear solvers".** In: *Proceedings of the 2016 International Conference for High Performance Computing, Networking, Storage and Analysis*. SC16. IEEE.


Knepley, Matthew (2013). ***Nested and Hierarchical Solvers in PETSc.*** URL: https://www.caam.rice.edu/~mk51/presentations/SIAMCSE13.pdf.

References v


Li, Xiaoye (2005). **"An Overview of SuperLU: Algorithms, Implementation, and User Interface".** In: *ACM Transactions on Mathematical Software* 31.3, pp. 302–325. URL: http://crd-legacy.lbl.gov/~xiaoye/SuperLU.


Mandel, Jan (1993). **"Balancing domain decomposition".** In: *Communications in Numerical Methods in Engineering* 9.3, pp. 233–241.


Moulin, Johann, Pierre Jolivet, and Olivier Marquet (2019). **"Augmented Lagrangian preconditioner for large-scale hydrodynamic stability analysis".** In: *Computer Methods in Applied Mechanics and Engineering* 351, pp. 718–743. URL: https://github.com/prj-/moulin2019al.


Schwarz, Hermann (1870). **"Über einen Grenzübergang durch alternierendes Verfahren".** In: *Vierteljahrsschrift der Naturforschenden Gesellschaft in Zürich* 15, pp. 272–286.


Si, Hang (2013). ***TetGen: A Quality Tetrahedral Mesh Generator and 3D Delaunay Triangulator.*** Tech. rep. 13. URL: http://wias-berlin.de/software/tetgen.

References vi


Spillane, Nicole, Victorita Dolean, Patrice Hauret, Frédéric Nataf, Clemens Pechstein, and Robert Scheichl (2013). **“Abstract robust coarse spaces for systems of PDEs via generalized eigenproblems in the overlaps”.** In: *Numerische Mathematik* 126.4, pp. 741–770.


Suzuki, Atsushi and François-Xavier Roux (2014). **“A dissection solver with kernel detection for symmetric finite element matrices on shared memory computers”.** In: *International Journal for Numerical Methods in Engineering* 100.2, pp. 136–164.

Index i

Index II

Index iii

Index iv

Index v

Index vi