FreeFEM, PETSc, SLEPc, and HPDDM

Pierre Jolivet — Sorbonne Université, CNRS, LIP6

December 12, 2024

`https://joliv.et/FF_16th.pdf`

FreeFEM days, 16th edition

Introduction

1. Introduction
2. Using PETSc
3. How to make my code go faster?
4. Advanced functionalities
5. Domain decomposition methods
6. Conclusion

Introduction

Basic sequential functionalities

Domain specific language for FE

- mesh structure
- associated finite element spaces
- vector and matrix assemblies
- algebraic operations

Interface to other libraries

- installed by PETSc (`make petsc-slepc`)
  - MPICH (if no other MPI implementation)
  - CMake (if unavailable)
  - Bison
  - BLAS (if unavailable)
  - MUMPS (if Fortran available)
  - SuperLU_DIST (if Fortran unavailable)
  - SuperLU
  - SuiteSparse
  - *hypre*
  - [Par]METIS
  - SCOTCH
  - TetGen
  - SLEPc
  - HPDDM (if shared libraries available)
  - ARPACK (if Fortran available)
  - [Par]Mmg
  - Htool

Interface to other libraries

- installed by PETSc (`make petsc-slepc`)
  - MPICH (if no other MPI implementation)
  - CMake (if unavailable)
  - Bison
  - BLAS (if unavailable)
  - MUMPS (if Fortran available)
  - SuperLU_DIST (if Fortran unavailable)
  - SuperLU
  - SuiteSparse
  - *hypre*
  - [Par]METIS
  - SCOTCH
  - TetGen
  - SLEPc
  - HPDDM (if shared libraries available)
  - ARPACK (if Fortran available)
  - [Par]Mmg
  - Htool
- tested on Windows, Linux, Apple Intel/ARM, A64FX...

Interface to other libraries

- installed by PETSc (`make petsc-slepc`)
  - MPICH (if no other MPI implementation)
  - CMake (if unavailable)
  - Bison
  - BLAS (if unavailable)
  - MUMPS (if Fortran available)
  - SuperLU_DIST (if Fortran unavailable)
  - SuperLU
  - SuiteSparse
  - *hypre*
  - [Par]METIS
  - SCOTCH
  - TetGen
  - SLEPc
  - HPDDM (if shared libraries available)
  - ARPACK (if Fortran available)
  - [Par]Mmg
  - Htool
- tested on Windows, Linux, Apple Intel/ARM, A64FX...
- could add other packages: HDF5, GSL, Boost, zlib?

Interface to other libraries

- installed by PETSc (`make petsc-slepc`)
  - MPICH (if no other MPI implementation)
  - CMake (if unavailable)
  - Bison
  - BLAS (if unavailable)
  - MUMPS (if Fortran available)
  - SuperLU_DIST (if Fortran unavailable)
  - SuperLU
  - SuiteSparse
  - *hypre*
  - [Par]METIS
  - SCOTCH
  - TetGen
  - SLEPc
  - HPDDM (if shared libraries available)
  - ARPACK (if Fortran available)
  - [Par]Mmg
  - Htool
- tested on Windows, Linux, Apple Intel/ARM, A64FX...
- could add other packages: HDF5, GSL, Boost, zlib?

$\Longrightarrow$ sequential or parallel (with FreeFem++-mpi)

Using PETSc

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 | . . .

Specific type

- `Mat` instead of `matrix`
- easy conversion via `matrix A; Mat B(A)`

Specific type

- `Mat` instead of `matrix`
- easy conversion via `matrix A; Mat B(A)`
- same algebraic operations (`*`, `^-1`...)
- `set(sparams = "...")` to fine-tune solvers
- `ObjectView` or `MatLoad`

Specific type

- `Mat` instead of `matrix`
- easy conversion via `matrix A; Mat B(A)`
- same algebraic operations (`*`, `^-1`...)
- `set(sparams = "...")` to fine-tune solvers
- `ObjectView` or `MatLoad`
- single exposed type but many implementations
  - `MatAIJ`
  - `MatSBAIJ`
  - `MatHtool`
  - `MatNest`
  - `MatShell`
  - `MatDense`
  - `MatComposite`
  - `MatIS`
  - `MatTranspose`

How to make my code go faster?

Going parallel

- no shared-memory parallelism
- distributed-memory paradigm (MPI)

Going parallel

- no shared-memory parallelism
- distributed-memory paradigm (MPI)
- not fully transparent to users
- lightweight macros for workload distribution

Finite elements in parallel

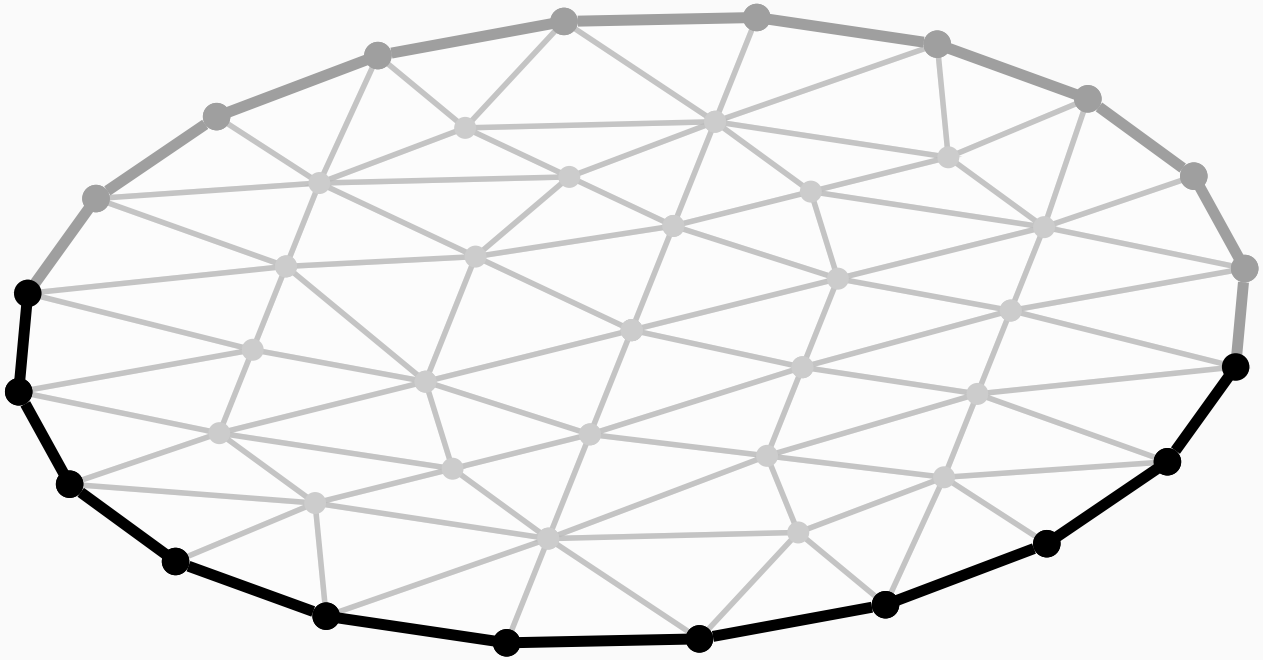
Distributed-memory (MPI)

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

Finite elements in parallel

Distributed-memory (MPI)

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

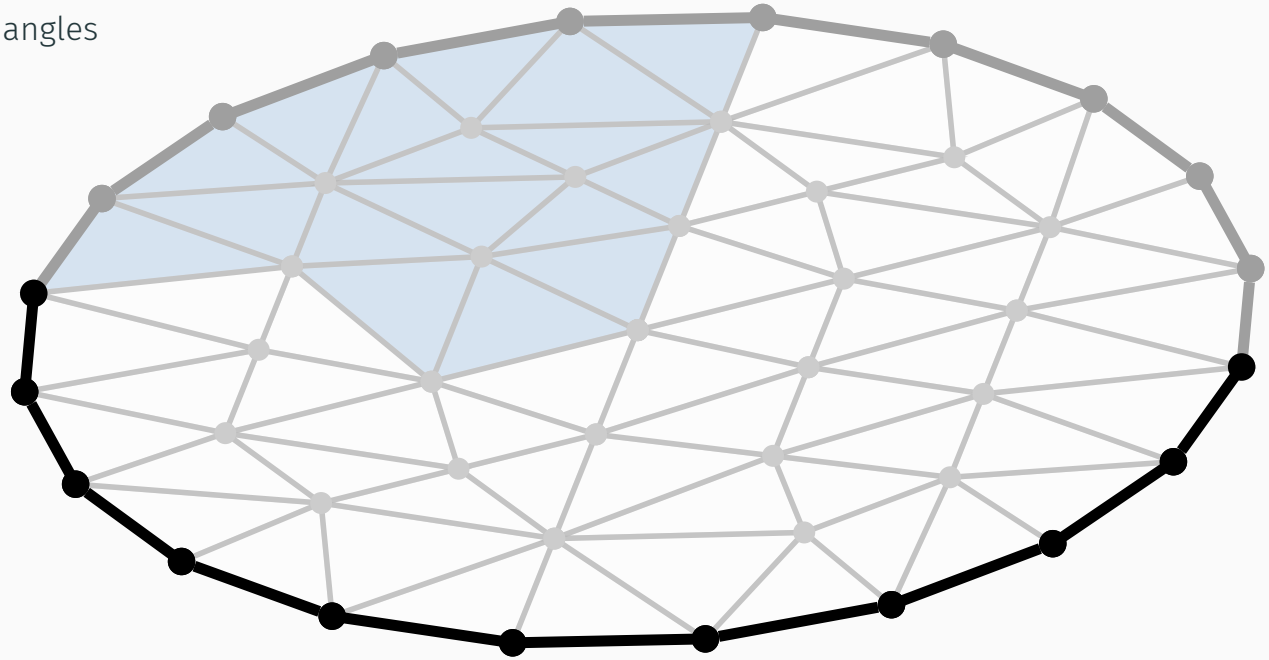


Finite elements in parallel

Distributed-memory (MPI)

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

Subdomain #1: 17 triangles

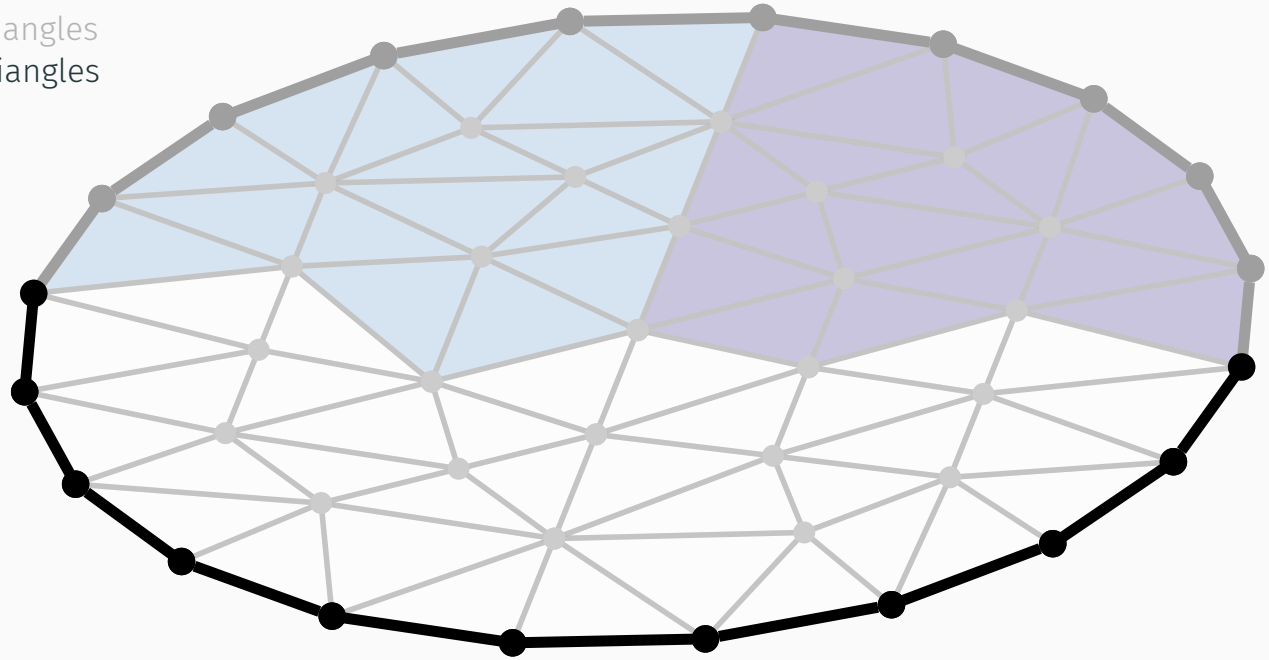


Finite elements in parallel

Distributed-memory (MPI)

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

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

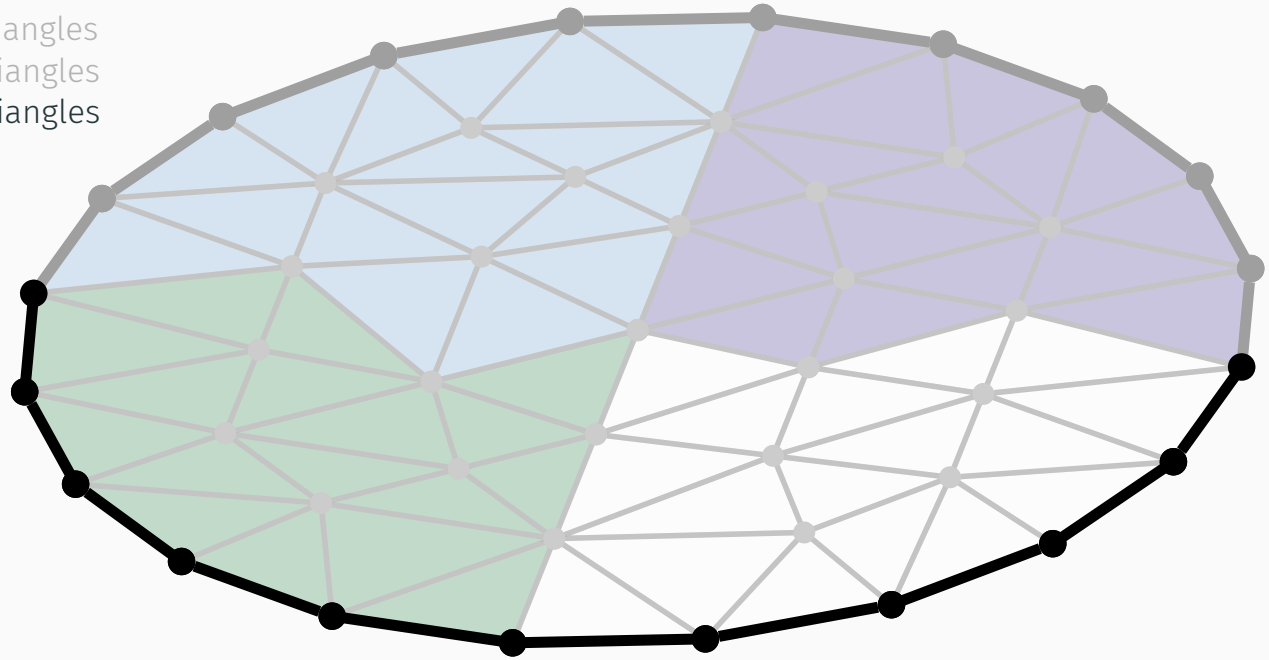


Finite elements in parallel

Distributed-memory (MPI)

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

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

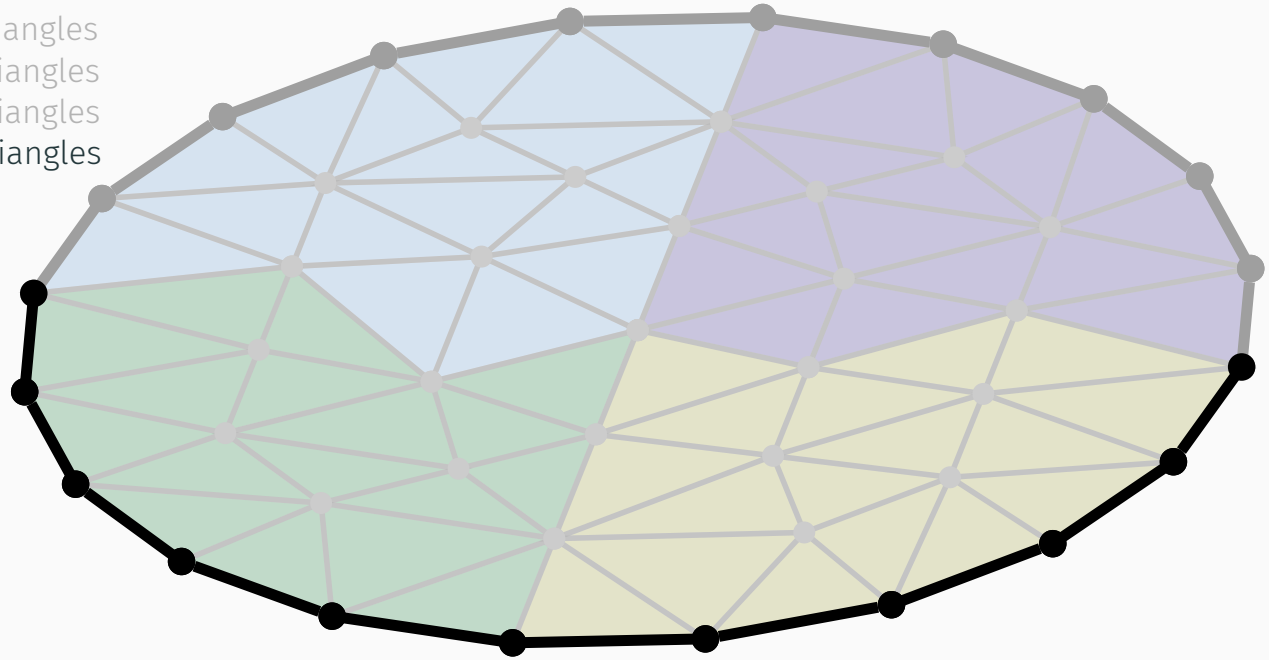


Finite elements in parallel

Distributed-memory (MPI)

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

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

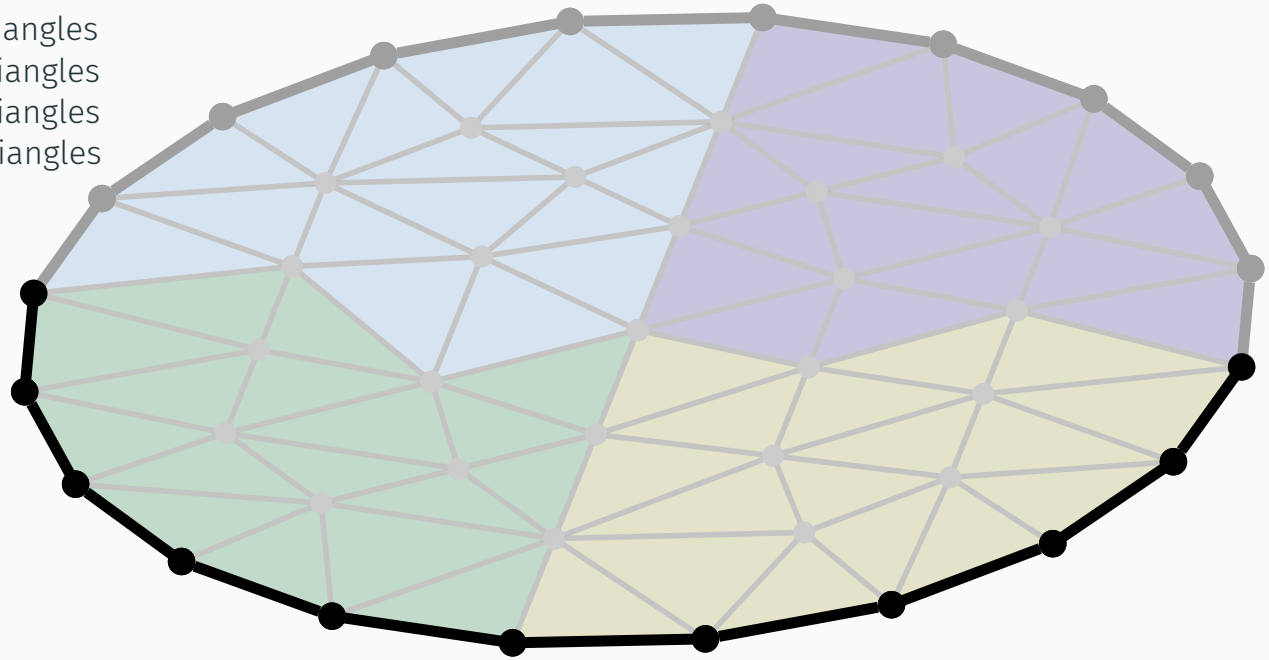


Finite elements in parallel

Distributed-memory (MPI)

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

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



The end of the story?

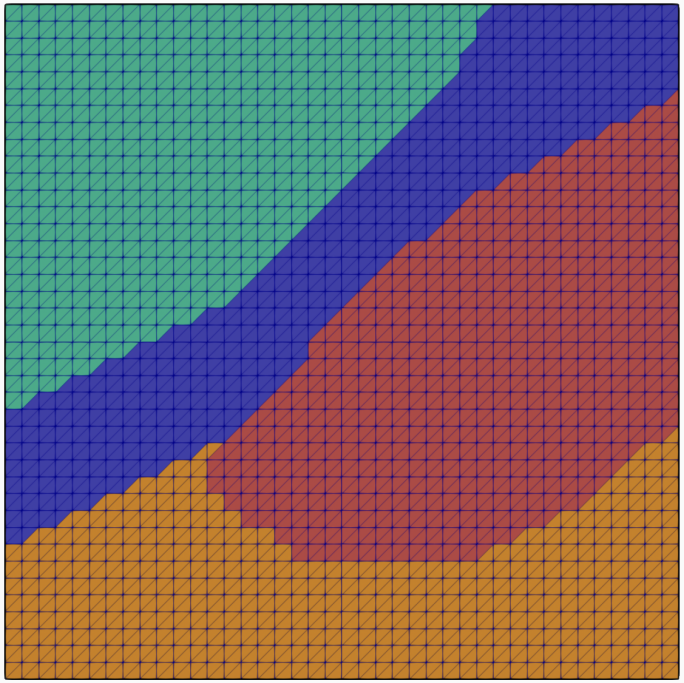
- assemblies are now easy to parallelize

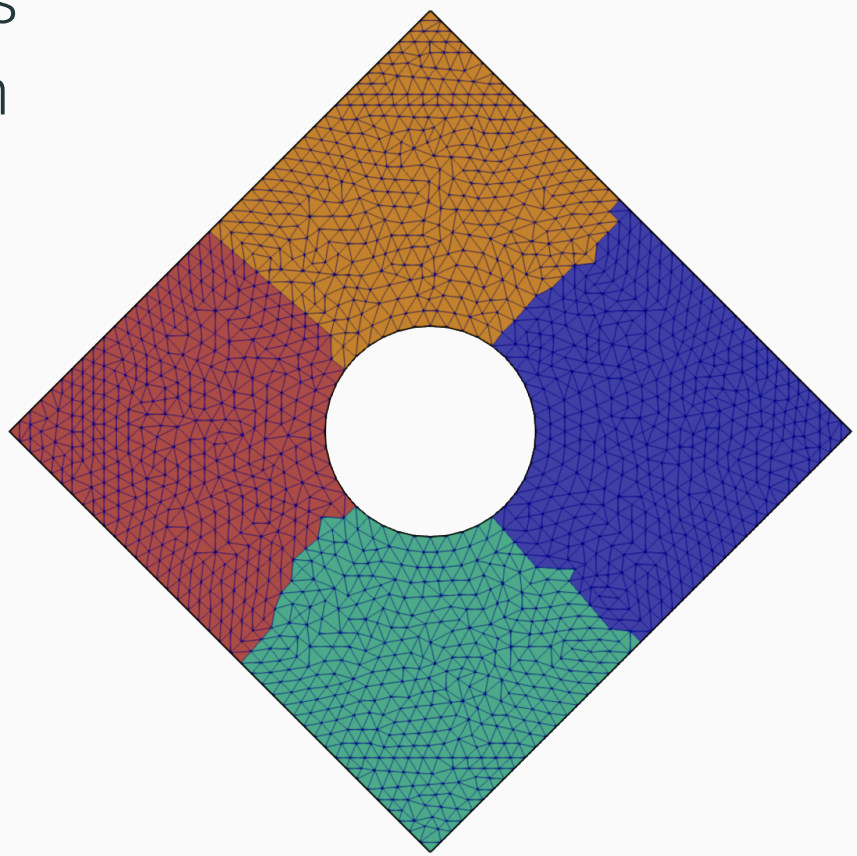
The end of the story?

- assemblies are now easy to parallelize
- many algorithms are still challenging
  - distance to a level-set
  - unstructured adaptive mesh refinement
  - nonlinear and linear solvers

Another example: interpolation

- different meshes
- different basis functions
- different decomposition





Small snippet of macro_ddm.idp

- DmeshSave
- DmeshLoad
- DmeshCreate
- MatCreate
- VecInterpolate
- MatInterpolate

Small snippet of macro_ddm.idp

- DmeshSave
- DmeshLoad
- DmeshCreate
- MatCreate
- VecInterpolate
- MatInterpolate

- examples, see examples/hpddm/README.md
- functionalities, see CHANGELOG.md
- https://joliv.et/FreeFem-tutorial

Advanced functionalities

Nonstandard eigensolvers

- nonlinear $T(\lambda)x = 0 \implies$ `NEPSolve`
- polynomial $\left(\sum_{i=0}^{N} \lambda^i A_i\right)x = 0, N \geqslant 2 \implies$ `PEPSolve`
- SVD $Av = \sigma u \implies$ `SVDSolve`
- more in [Roman et al., 2023]

Array of PETSc matrices

As easy as FreeFEM matrices: `Mat[int] A(N)`

Easier use of shells for preconditioners

Oseen equations: $A = \begin{bmatrix} F & B \\ B^T & C \end{bmatrix} \Longrightarrow$

```
Mat A = [[F , B],
         [B', C]]
```

Easier use of shells for preconditioners

Oseen equations: $A = \begin{bmatrix} F & B \\ B^T & C \end{bmatrix} \Longrightarrow$

```
Mat A = [[F , B],
         [B', C]]
```

```
Mat Object: 4 MPI processes
 type: nest
 rows=37507, cols=37507
  Matrix object:
   type=nest, rows=2, cols=2
   MatNest structure:
   (0,0) : prefix="fieldsplit_0_", type=mpiaij...
   (0,1) : type=mpiaij, rows=33282, cols=4225
   (1,0) : type=transpose, rows=4225, cols=33282
   (1,1) : prefix="fieldsplit_1_", type=mpiaij...
```

Easier use of shells for preconditioners

Oseen equations: $A = \begin{bmatrix} F & B \\ B^T & C \end{bmatrix} \Longrightarrow$

```
Mat A = [[F , B],
         [B', C]]
```

Triangular preconditioner: $\tilde{A} = \begin{bmatrix} \tilde{F} & B \\ 0 & \tilde{S} \end{bmatrix}$ with $\tilde{S}^{-1} \approx -M_p^{-1} F_p L_p^{-1}$

Easier use of shells for preconditioners

Oseen equations: $A = \begin{bmatrix} F & B \\ B^T & C \end{bmatrix} \Longrightarrow$ `Mat A = [[F , B], [B', C]]`

Triangular preconditioner: $\tilde{A} = \begin{bmatrix} \tilde{F} & B \\ 0 & \tilde{S} \end{bmatrix}$ with $\tilde{S}^{-1} \approx -M_p^{-1} F_p L_p^{-1}$

```
func real[int] PCD(real[int]& in) {
    real[int] out(in.n);
    KSPSolve(Lp,in,in); MatMult(Fp,in,out);
    KSPSolve(Mp,out,out); out *= -1.0;
    return out; // = -Mp \ Fp * Lp \ in
}
set(C, parent = A, precon = PCD,
       sparams = "-fieldsplit_1_pc_type shell");
```

FEM-BEM coupling

Solve the generalized eigenvalue problem for (u, λ):

$$\begin{bmatrix} A_k & D'_k - \frac{1}{2}M^{\mathrm{tr}} \\ M^{\mathrm{tr}} & -S_k \end{bmatrix} u = \lambda \begin{bmatrix} M & 0 \\ 0 & 0 \end{bmatrix} u$$

```
Mat<complex> C;
Sk *= -1.0;
C = [[Ak , R*TDkM]
     [F2B, Sk    ]];
```

FEM-BEM coupling

Solve the generalized eigenvalue problem for (u, λ):

$$\begin{bmatrix} A_k & D'_k - \frac{1}{2}M^{\mathrm{tr}} \\ M^{\mathrm{tr}} & -S_k \end{bmatrix} u = \lambda \begin{bmatrix} M & 0 \\ 0 & 0 \end{bmatrix} u$$

```
Mat<complex> C;
Sk *= -1.0;
C = [[Ak , R*TDkM]
     [F2B, Sk    ]];
ObjectView(C);
```

```
Mat Object: 4 MPI processes
  type=nest, rows=2, cols=2
  MatNest structure:
  (0,0) : type=mpiaij, rows=10410, cols=10410
  (0,1) : type=composite, rows=10410, cols=...
  (1,0) : type=mpiaij, rows=1156, cols=10410
  (1,1) : type=htool, rows=1156, cols=1156
```

More economical MatNest

- support for dense blocks `real[int, int] d`
- pay the (memory) price twice with

```
matrix A = [[a,  d],
            [d', 0]]
```

More economical MatNest

- support for dense blocks `real[int, int] d`
- pay the (memory) price twice with

```
matrix A = [[a,  d],
            [d', 0]]
```

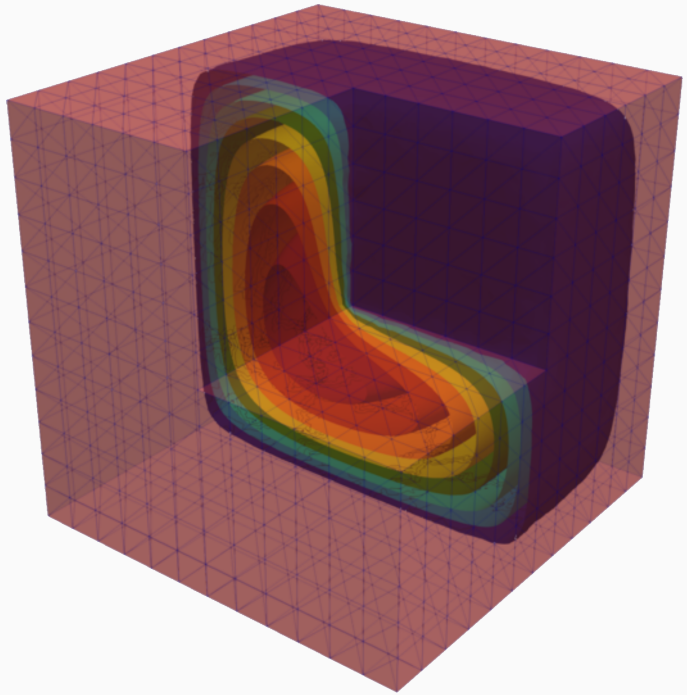
- no such overhead with

```
Mat B = [[b,  d],
         [d', 0]]
```

- the action of the transpose is computed implicitly
- `MATSOLVERMUMPS` handles `MatNest` since 3.20

Parallel mesh adaptation

- Poisson equation on the Fischera corner
- algebraic system solved with GAMG
- 128 MPI processes of Irène@TGCC



Parallel mesh adaptation

- Poisson equation on the Fischera corner
- algebraic system solved with GAMG
- 128 MPI processes of Irène@TGCC

Iteration	#2
# of tetrahedra	47k
ParMmg (sec)	2.6
transfer (sec)	2.8
GAMG (sec)	0.2

Parallel mesh adaptation

- Poisson equation on the Fischera corner
- algebraic system solved with GAMG
- 128 MPI processes of Irène@TGCC

Iteration	#2	#3
# of tetrahedra	47k	91k
ParMmg (sec)	2.6	4.9
transfer (sec)	2.8	1.9
GAMG (sec)	0.2	0.2

Parallel mesh adaptation

- Poisson equation on the Fischera corner
- algebraic system solved with GAMG
- 128 MPI processes of Irène@TGCC

Iteration	#2	#3	#4
# of tetrahedra	47k	91k	221k
ParMmg (sec)	2.6	4.9	16.2
transfer (sec)	2.8	1.9	1.4
GAMG (sec)	0.2	0.2	0.5

Parallel mesh adaptation

- Poisson equation on the Fischera corner
- algebraic system solved with GAMG
- 128 MPI processes of Irène@TGCC

Iteration	#2	#3	#4	#5
# of tetrahedra	47k	91k	221k	686k
ParMmg (sec)	2.6	4.9	16.2	22.0
transfer (sec)	2.8	1.9	1.4	1.5
GAMG (sec)	0.2	0.2	0.5	0.6

Parallel mesh adaptation

- Poisson equation on the Fischera corner
- algebraic system solved with GAMG
- 128 MPI processes of Irène@TGCC

Iteration	#2	#3	#4	#5	#6
# of tetrahedra	47k	91k	221k	686k	2.2M
ParMmg (sec)	2.6	4.9	16.2	22.0	49.9
transfer (sec)	2.8	1.9	1.4	1.5	2.7
GAMG (sec)	0.2	0.2	0.5	0.6	0.9

Parallel mesh adaptation

- Poisson equation on the Fischera corner
- algebraic system solved with GAMG
- 128 MPI processes of Irène@TGCC

Iteration	#2	#3	#4	#5	#6	#7
# of tetrahedra	47k	91k	221k	686k	2.2M	7M
ParMmg (sec)	2.6	4.9	16.2	22.0	49.9	211
transfer (sec)	2.8	1.9	1.4	1.5	2.7	11.5
GAMG (sec)	0.2	0.2	0.5	0.6	0.9	2.8

Parallel mesh adaptation

- Poisson equation on the Fischera corner
- algebraic system solved with GAMG
- 128 MPI processes of Irène@TGCC

Iteration	#2	#3	#4	#5	#6	#7	#8
# of tetrahedra	47k	91k	221k	686k	2.2M	7M	22M
ParMmg (sec)	2.6	4.9	16.2	22.0	49.9	211	180
transfer (sec)	2.8	1.9	1.4	1.5	2.7	11.5	66
GAMG (sec)	0.2	0.2	0.5	0.6	0.9	2.8	4.2

Parallel mesh adaptation

- Poisson equation on the Fischera corner
- algebraic system solved with GAMG
- 128 MPI processes of Irène@TGCC

Iteration	#2	#3	#4	#5	#6	#7	#8	#9
# of tetrahedra	47k	91k	221k	686k	2.2M	7M	22M	62M
ParMmg (sec)	2.6	4.9	16.2	22.0	49.9	211	180	299
transfer (sec)	2.8	1.9	1.4	1.5	2.7	11.5	66	87
GAMG (sec)	0.2	0.2	0.5	0.6	0.9	2.8	4.2	6.0

Parallel mesh adaptation

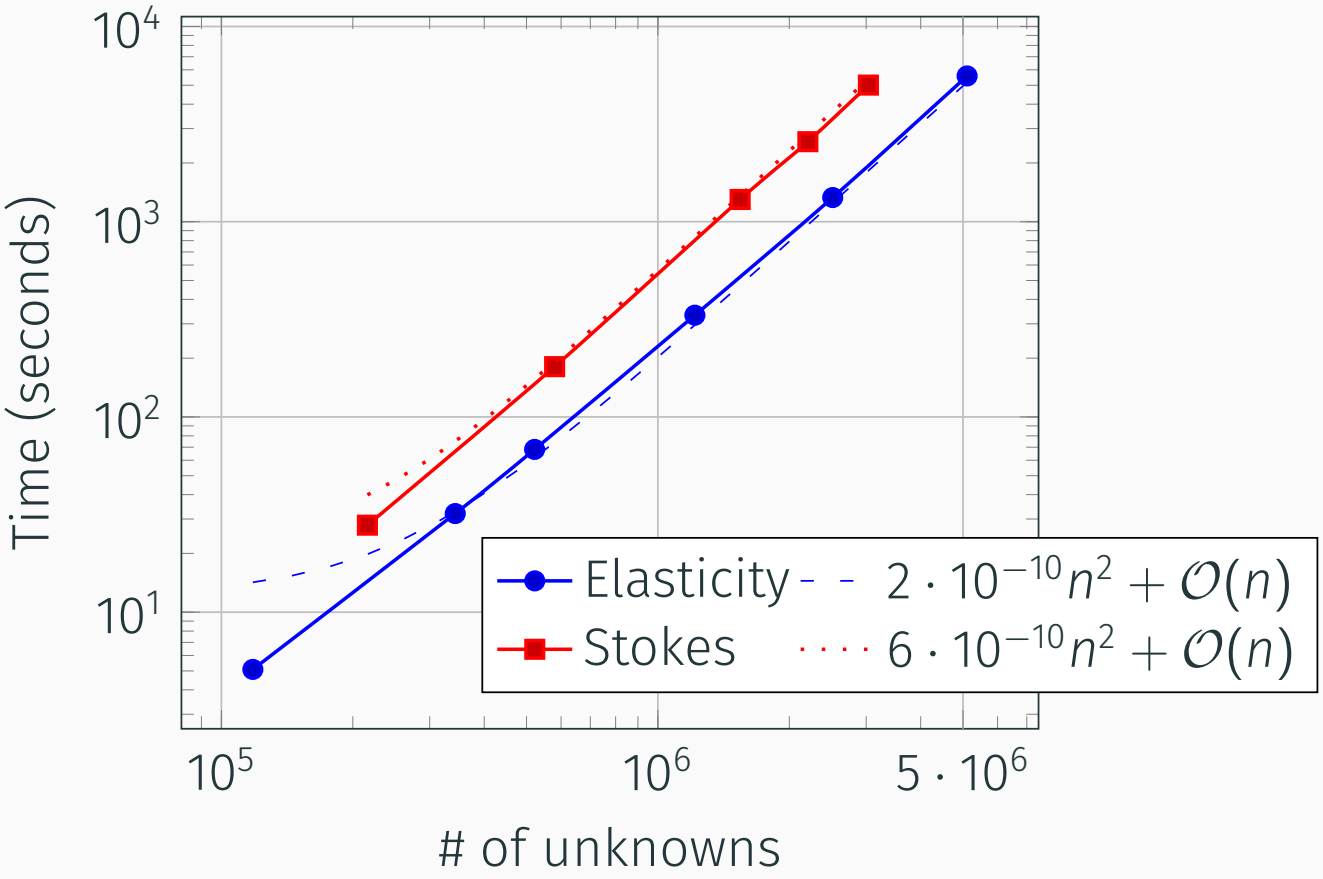
- Poisson equation on the Fischera corner
- algebraic system solved with GAMG
- 128 MPI processes of Irène@TGCC

Iteration	#2	#3	#4	#5	#6	#7	#8	#9	#10
# of tetrahedra	47k	91k	221k	686k	2.2M	7M	22M	62M	170M
ParMmg (sec)	2.6	4.9	16.2	22.0	49.9	211	180	299	697
transfer (sec)	2.8	1.9	1.4	1.5	2.7	11.5	66	87	297
GAMG (sec)	0.2	0.2	0.5	0.6	0.9	2.8	4.2	6.0	—

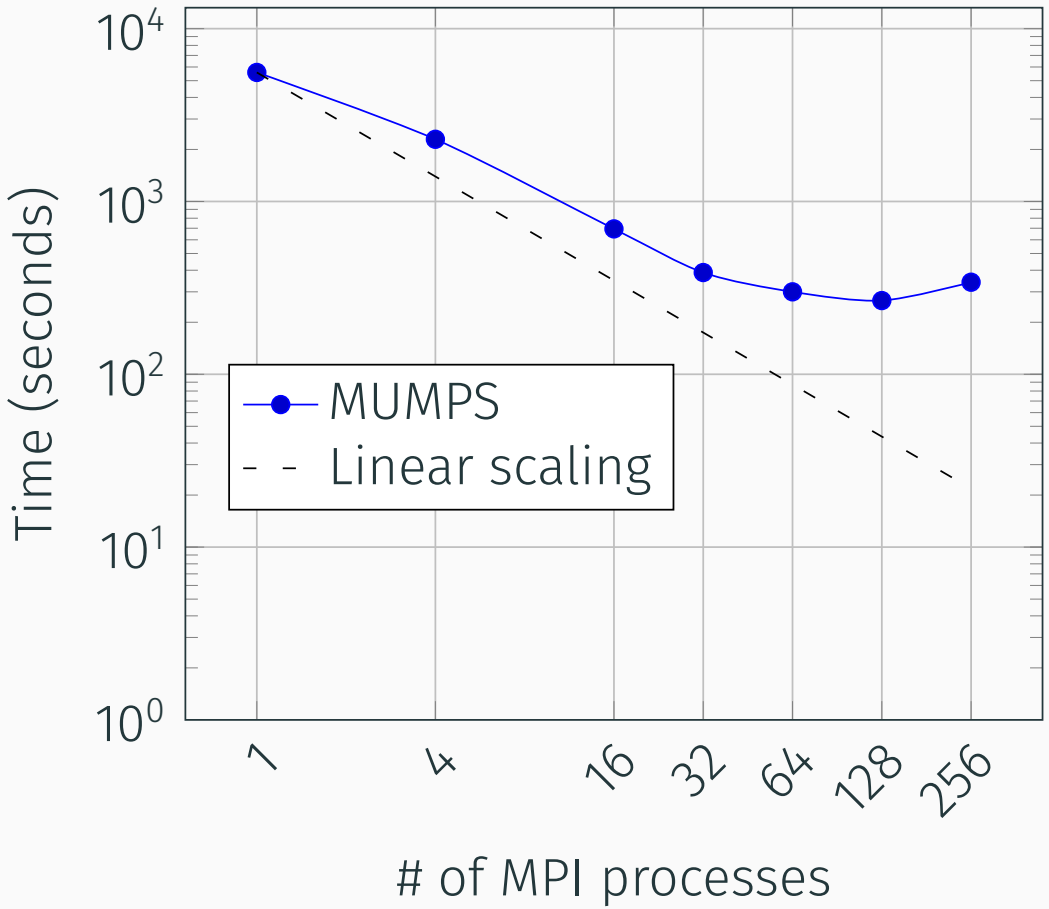
Domain decomposition methods

Complexity study of exact factorization

- 3D problems, lowest-order FE
- sequential, double-precision, exact LDL^T factorization
- `https://mumps-solver.org`

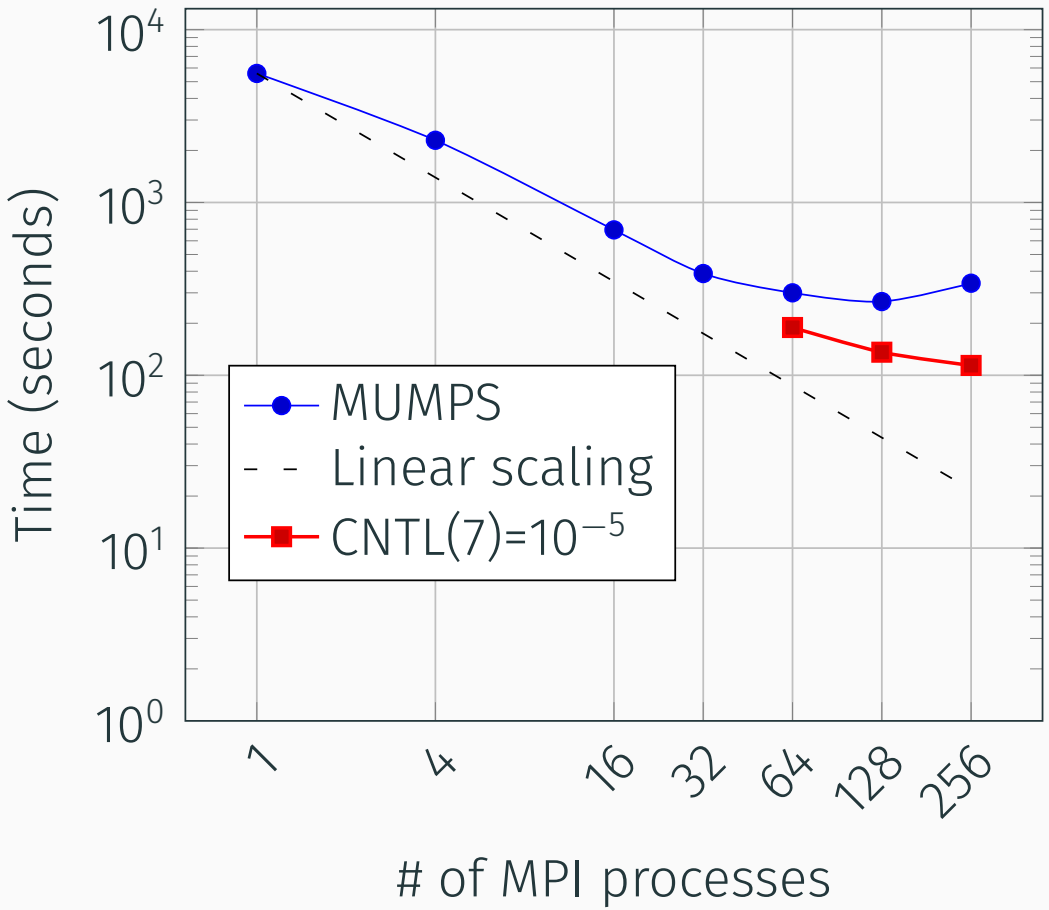


Scalability for the largest (elasticity) problem



- extremely robust solver
- difficult to reach ideal efficiency

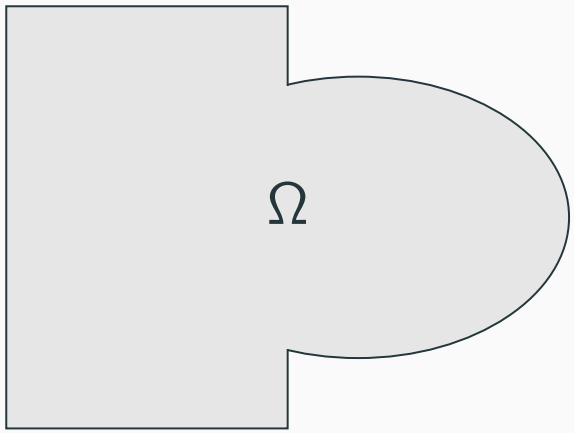
Scalability for the largest (elasticity) problem



- extremely robust solver
- difficult to reach ideal efficiency
- FGMRES with a 10^{-5} tolerance, 15 iterations

Overlapping Schwarz methods

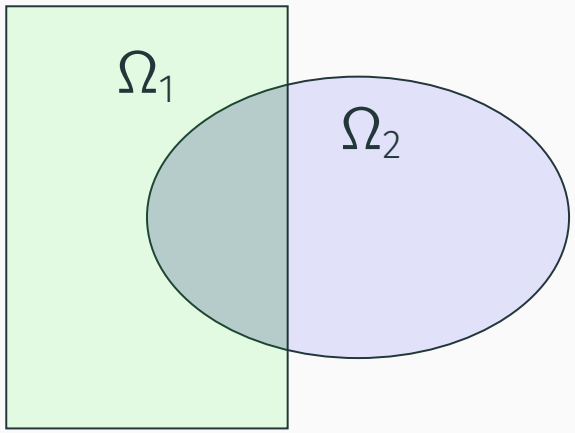
Consider the linear system: $Ax = b \in \mathbb{K}^n$



Overlapping Schwarz methods

Consider the linear system: $Ax = b \in \mathbb{K}^n$

Two-way decomposition of $[\![1; n]\!]$ and restriction operators

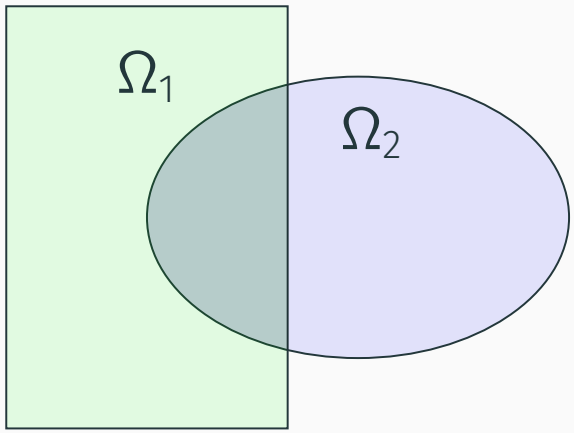


Overlapping Schwarz methods

Consider the linear system: $Ax = b \in \mathbb{K}^n$

Two-way decomposition of $[\![1; n]\!]$ and restriction operators

$$M_{\text{ASM}}^{-1} = \sum_{i=1}^{N=2} R_i^T (R_i A R_i^T)^{-1} R_i$$



[Schwarz, 1870]

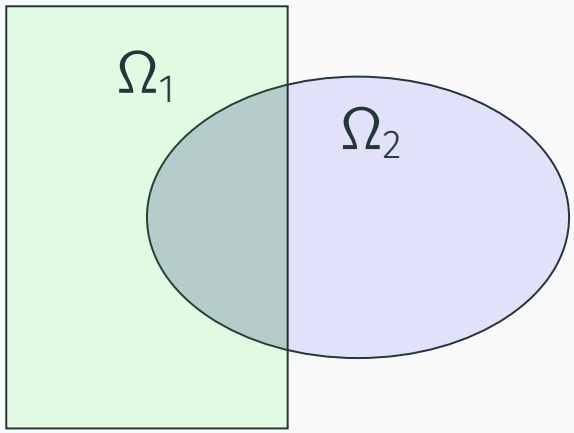
Overlapping Schwarz methods

Consider the linear system: $Ax = b \in \mathbb{K}^n$

Two-way decomposition of $[\![1; n]\!]$ and restriction operators

$$M_{\text{ASM}}^{-1} = \sum_{i=1}^{N=2} R_i^T (R_i A R_i^T)^{-1} R_i$$

$$M_{\text{RAS}}^{-1} = \sum_{i=1}^{N=2} \tilde{R}_i^T (R_i A R_i^T)^{-1} R_i$$



[Schwarz, 1870] [Cai and Sarkis, 1999]

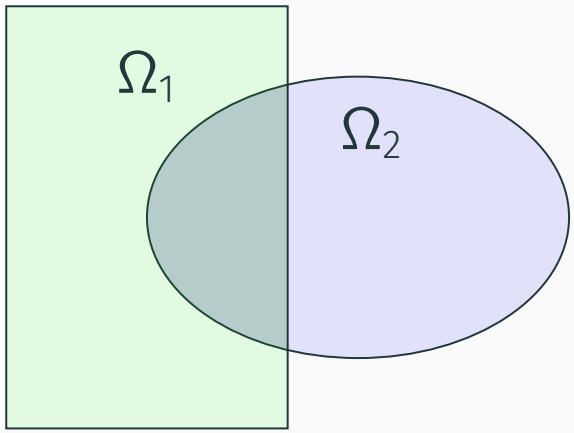
Overlapping Schwarz methods

Consider the linear system: $Ax = b \in \mathbb{K}^n$

Two-way decomposition of $[\![1; n]\!]$ and restriction operators

$$M_{\text{ASM}}^{-1} = \sum_{i=1}^{N=2} R_i^T (R_i A R_i^T)^{-1} R_i$$

$$M_{\text{RAS}}^{-1} = \sum_{i=1}^{N=2} \tilde{R}_i^T (R_i A R_i^T)^{-1} R_i$$



[Schwarz, 1870] [Cai and Sarkis, 1999]

$\rightarrow$ no guaranteed scalability for large N

Coarse correction for domain decomposition

Abstract setting

- define locally $W_i = \begin{bmatrix} \Lambda_1 & \cdots & \Lambda_{\gamma_i} \end{bmatrix}$

Coarse correction for domain decomposition

Abstract setting

- define locally $W_i = \begin{bmatrix} \Lambda_1 & \cdots & \Lambda_{\gamma_i} \end{bmatrix}$
- use $Z = \begin{bmatrix} \tilde{R}_1^T W_1 & \cdots & \tilde{R}_N^T W_N \end{bmatrix}$ to compute $A_C = Z^T A Z$

Coarse correction for domain decomposition

Abstract setting

- define locally $W_i = \begin{bmatrix} \Lambda_1 & \cdots & \Lambda_{\gamma_i} \end{bmatrix}$
- use $Z = \begin{bmatrix} \tilde{R}_1^T W_1 & \cdots & \tilde{R}_N^T W_N \end{bmatrix}$ to compute $A_C = Z^T A Z$
- coarse correction $M^{-1}_{\text{additive}} = Z A_C^{-1} Z^T + M^{-1}_{\text{ASM}}$

Coarse correction for domain decomposition

Abstract setting

- define locally $W_i = \begin{bmatrix} \Lambda_1 & \cdots & \Lambda_{\gamma_i} \end{bmatrix}$
- use $Z = \begin{bmatrix} \tilde{R}_1^T W_1 & \cdots & \tilde{R}_N^T W_N \end{bmatrix}$ to compute $A_C = Z^T A Z$
- coarse correction $M^{-1}_{\text{additive}} = Z A_C^{-1} Z^T + M^{-1}_{\text{ASM}}$

How to choose γ_i and $\{\Lambda_j\}_{j=1,\ldots,\gamma_i}$?

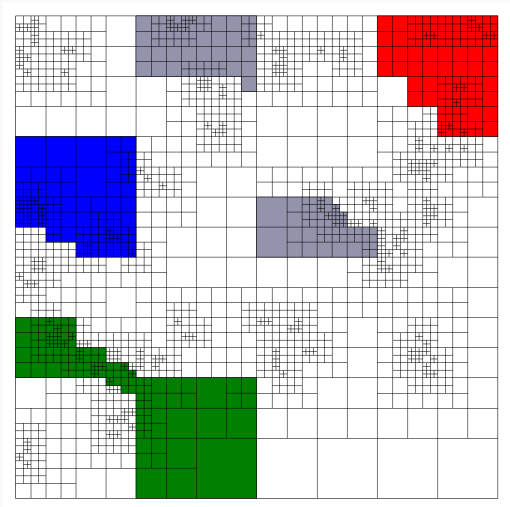
- aggressive coarsening
- capturing the “difficulties” of A
- at a moderate cost

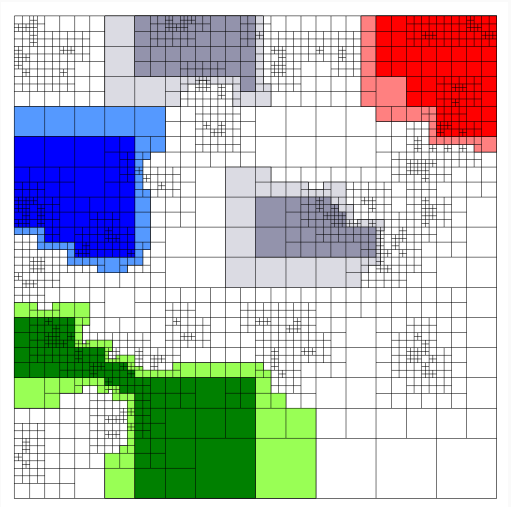
HPDDM

- specific implementation [Jolivet, Hecht, Nataf, et al., 2013]

HPDDM + PETSc

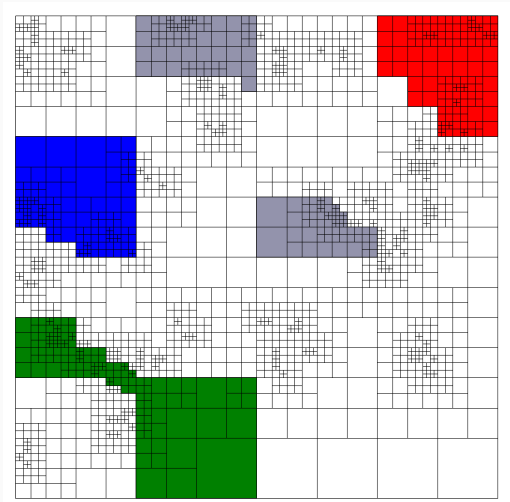
- specific implementation [Jolivet, Hecht, Nataf, et al., 2013]
- interfaced in PETSc [Jolivet, Roman, and Zampini, 2021]
  - user-provided N_i (`PCHPDDMSetAuxiliaryMat`)
  - automatically assembled on `DMPlex`

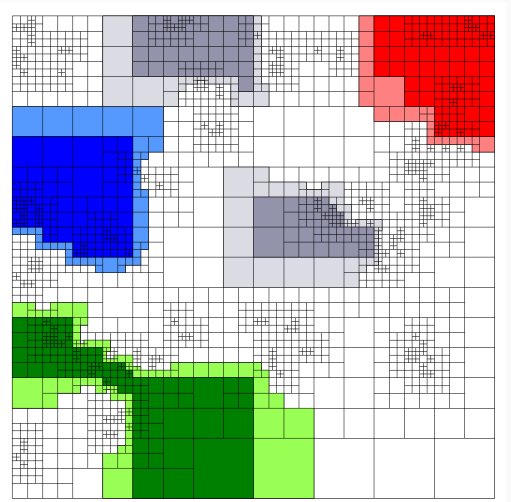




HPDDM + PETSc

- specific implementation [Jolivet, Hecht, Nataf, et al., 2013]
- interfaced in PETSc [Jolivet, Roman, and Zampini, 2021]
  - user-provided N_i (`PCHPDDMSetAuxiliaryMat`)
  - automatically assembled on `DMPlex`

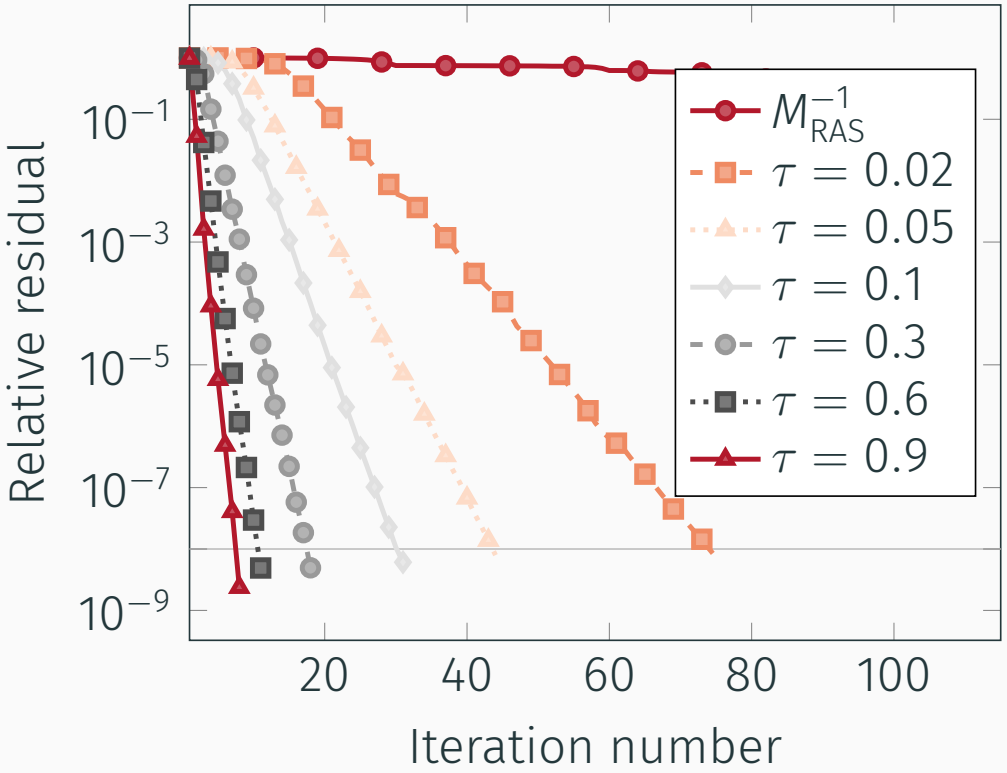




$\rightarrow$ leveraging the flexibility of PETSc for algebraic variants

Adaptive preconditioning for general systems

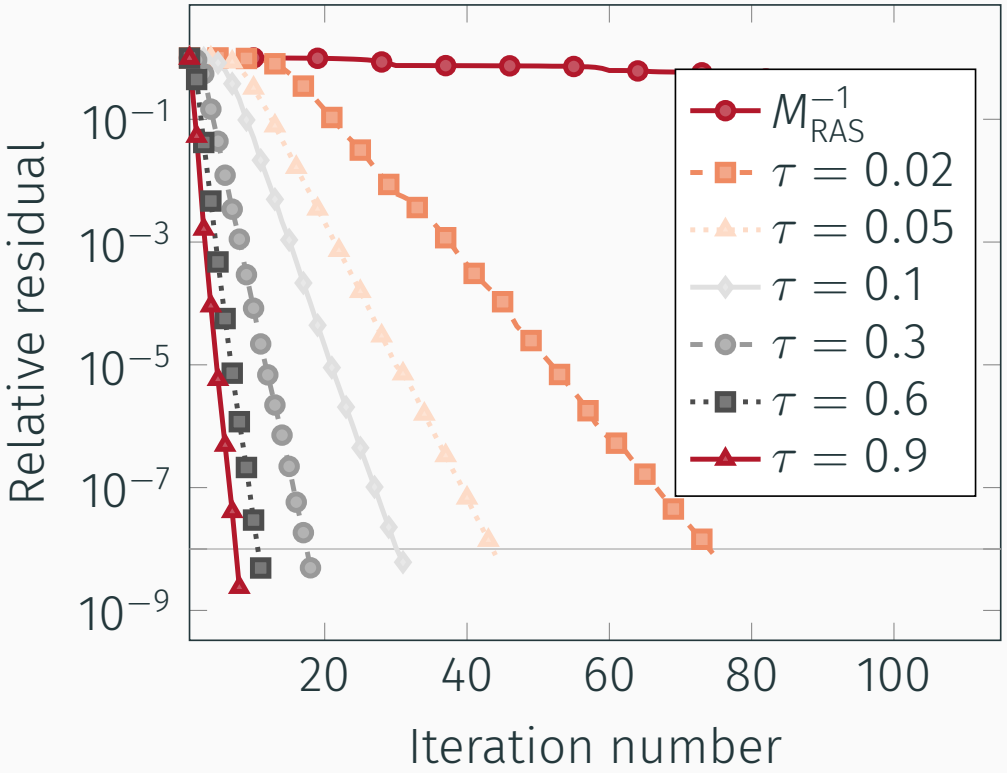
G3_circuit from SSMC on
256 processes [Al Daas, Jolivet, and Rees, 2023]



τ	n_C	Iterations
0.02	739	75
0.05	1,831	44
0.1	3,608	31
0.3	11,784	18
0.6	34,402	11
0.9	71,385	8

Adaptive preconditioning for general systems

G3_circuit from SSMC on
256 processes [Al Daas, Jolivet, and Rees, 2023]

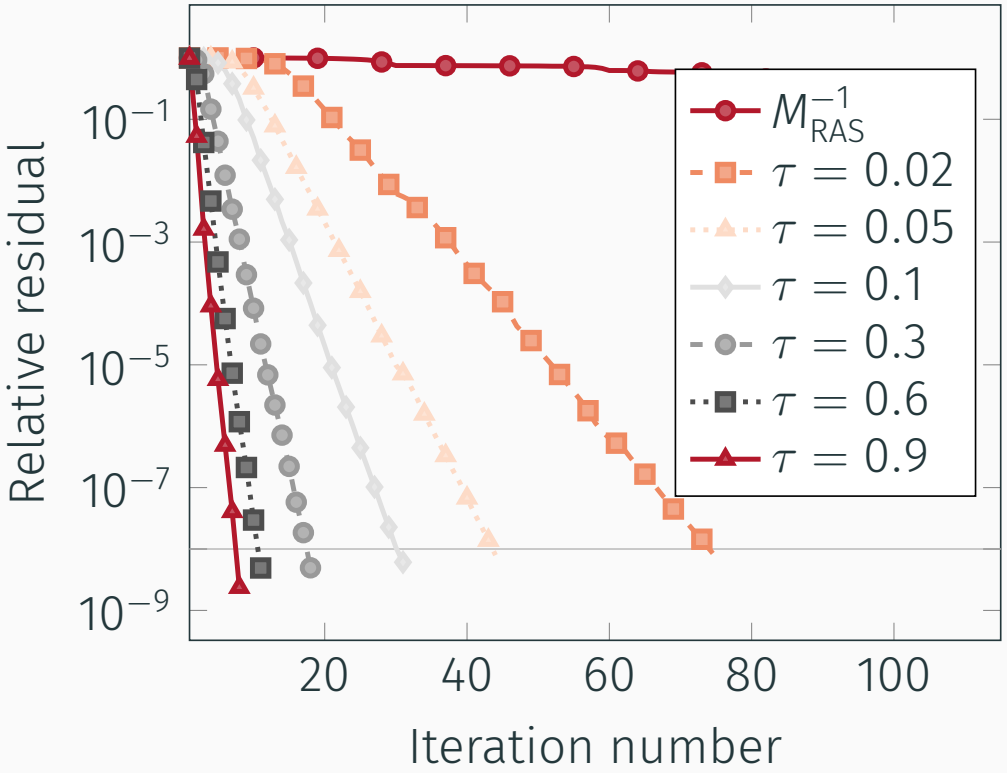


τ	n_C	Iterations
0.02	739	75
0.05	1,831	44
0.1	3,608	31
0.3	11,784	18
0.6	34,402	11
0.9	71,385	8

Identifier	M_{deflated}^{-1}	BoomerAMG	GAMG
thermal2	26	14	20
Transport	9	10	98
atmosmodj	7	8	17

Adaptive preconditioning for general systems

G3_circuit from SSMC on
256 processes [Al Daas, Jolivet, and Rees, 2023]



τ	n_C	Iterations
0.02	739	75
0.05	1,831	44
0.1	3,608	31
0.3	11,784	18
0.6	34,402	11
0.9	71,385	8

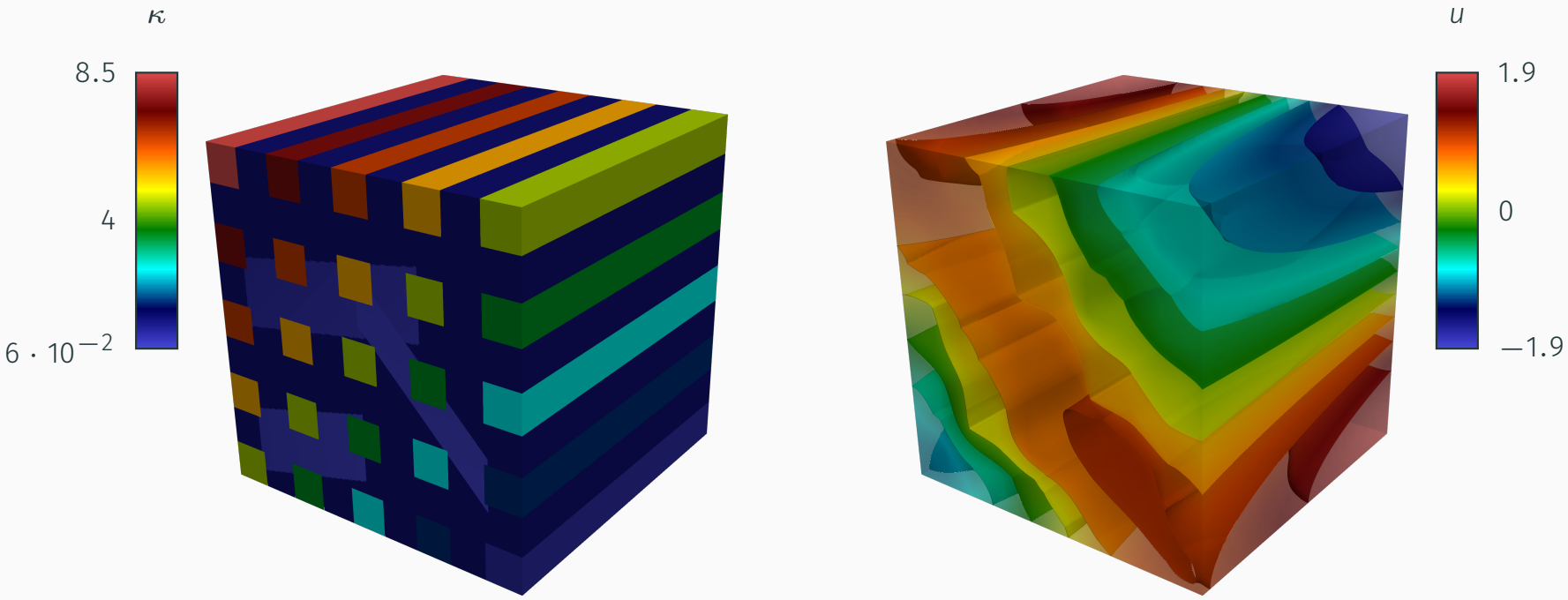
Identifier	M_{deflated}^{-1}	BoomerAMG	GAMG
thermal2	26	14	20
Transport	9	10	98
atmosmodj	7	8	17

Identifier	M_{deflated}^{-1}	BoomerAMG	GAMG
Chevron4	5	†	†
consph	93	†	†
nxp1	20	†	†

Nonlinear problem

- automatic assembly of Neumann matrices in FreeFEM
- Newton method using SNES [Brune et al., 2015]

$$-\nabla \cdot (\kappa \nabla u) - 6.2e^{u} = 0$$

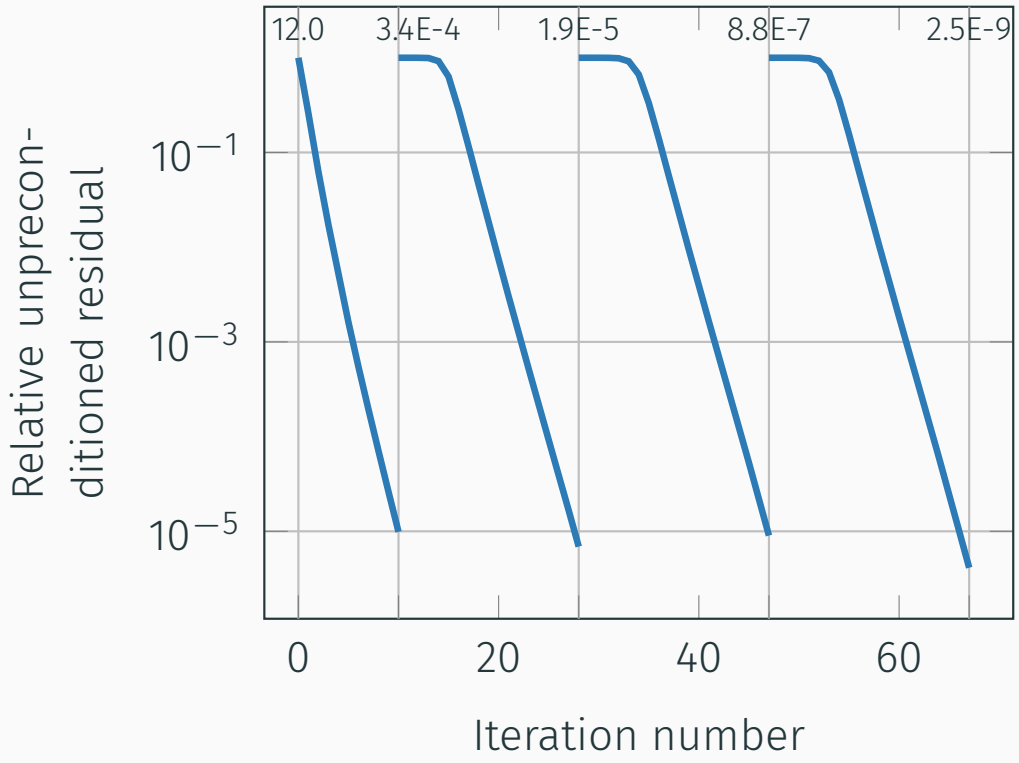


(a) Coefficient distribution (b) Isosurfaces of the solution

Nonlinear problem

- automatic assembly of Neumann matrices in FreeFEM
- Newton method using SNES [Brune et al., 2015]

$$-\nabla \cdot (\kappa \nabla u) - 6.2e^{u} = 0$$



Recent extensions to saddle-point problems

$$Ax = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$$

- theory from [Nataf and Tournier, 2022]
- apply HPDDM on the top-left block: $A_{00}^{-1} \approx M_{00,\text{deflated}}^{-1}$

Recent extensions to saddle-point problems

$$Ax = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$$

- theory from [Nataf and Tournier, 2022]
- apply HPDDM on the top-left block: $A_{00}^{-1} \approx M_{00,\text{deflated}}^{-1}$
- approximate $S \approx \tilde{S} = A_{11} - A_{10} M_{00,\text{deflated}}^{-1} A_{01}$

Recent extensions to saddle-point problems

$$Ax = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$$

- theory from [Nataf and Tournier, 2022]
- apply HPDDM on the top-left block: $A_{00}^{-1} \approx M_{00,\text{deflated}}^{-1}$
- approximate $S \approx \tilde{S} = A_{11} - A_{10} M_{00,\text{deflated}}^{-1} A_{01}$
- write $\tilde{S} = S_{11} - A_{10} Q_{00} A_{01}$ with Q_{00} the coarse operator

Recent extensions to saddle-point problems

$$Ax = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$$

- theory from [Nataf and Tournier, 2022]
- apply HPDDM on the top-left block: $A_{00}^{-1} \approx M_{00,\text{deflated}}^{-1}$
- approximate $S \approx \tilde{S} = A_{11} - A_{10} M_{00,\text{deflated}}^{-1} A_{01}$
- write $\tilde{S} = S_{11} - A_{10} Q_{00} A_{01}$ with Q_{00} the coarse operator
- two-level preconditioner $S_{11} \rightsquigarrow M_{11,\text{balanced}}^{-1}$
- preconditioner for S via $(I - M_{11,\text{balanced}}^{-1} A_{10} Q_{00} A_{01})$

Recent extensions to saddle-point problems

$$Ax = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$$

- theory from [Nataf and Tournier, 2022]
- apply HPDDM on the top-left block: $A_{00}^{-1} \approx M_{00,\text{deflated}}^{-1}$
- approximate $S \approx \tilde{S} = A_{11} - A_{10} M_{00,\text{deflated}}^{-1} A_{01}$
- write $\tilde{S} = S_{11} - A_{10} Q_{00} A_{01}$ with Q_{00} the coarse operator
- two-level preconditioner $S_{11} \rightsquigarrow M_{11,\text{balanced}}^{-1}$
- preconditioner for S via $(I - M_{11,\text{balanced}}^{-1} A_{10} Q_{00} A_{01})$

```
-prefix_push fieldsplit_1_
    -pc_type hpddm
    -pc_hpddm_schur_precondition geneo
-prefix_pop
```

- more at `stokes-block-hpddm-2d-PETSc.edp`

Recent coarse spaces

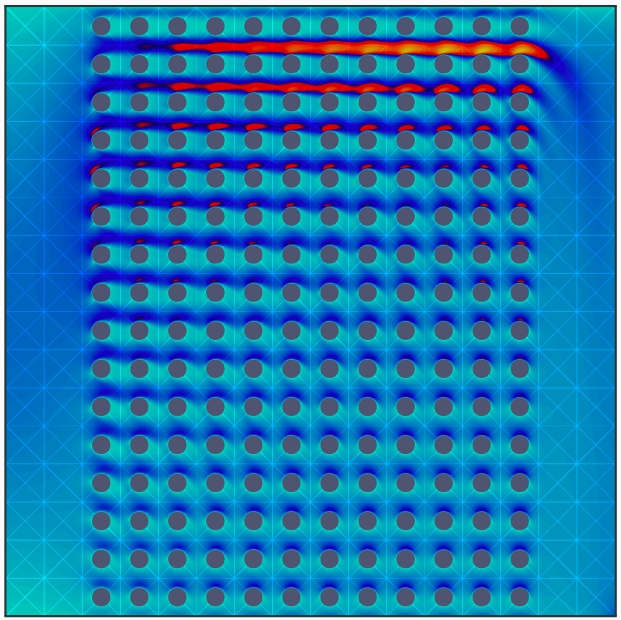
- for general systems [Al Daas, Jolivet, Nataf, et al., 2024]
- robust for nonsymmetric problems and algebraic

Recent coarse spaces

- for general systems [Al Daas, Jolivet, Nataf, et al., 2024]
- robust for nonsymmetric problems and algebraic
- Oseen equations: $-\rho\mathbf{V}\cdot\nabla\mathbf{u} = -\nabla p + \mu\Delta\mathbf{u}$

$$-\nabla\cdot\mathbf{u} = 0$$

- solution in a perforated domain from CEA
- 10^{-10} outer tolerance, 100M d.o.f., 1k MPI processes

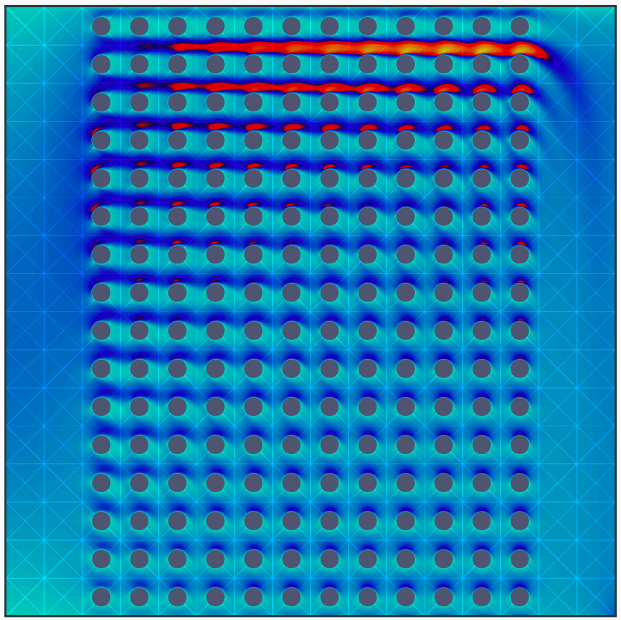


Recent coarse spaces

- for general systems [Al Daas, Jolivet, Nataf, et al., 2024]
- robust for nonsymmetric problems and algebraic
- Oseen equations: $-\rho\mathbf{V}\cdot\nabla\mathbf{u} = -\nabla p + \mu\Delta\mathbf{u}$

$$-\nabla\cdot\mathbf{u} = 0$$

- solution in a perforated domain from CEA
- 10^{-10} outer tolerance, 100M d.o.f., 1k MPI processes



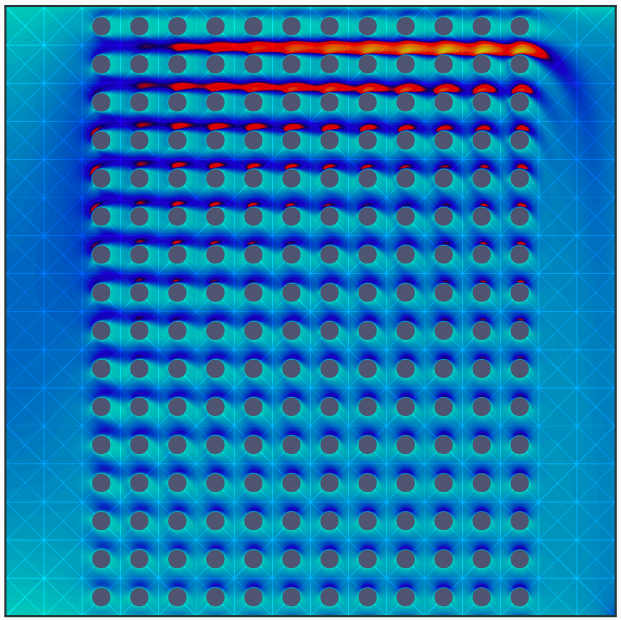
$\mu = 0.1$	outer solves	A_{00} solves	S solves
GenEO	5	50	8
harmonic	5	22	7

Recent coarse spaces

- for general systems [Al Daas, Jolivet, Nataf, et al., 2024]
- robust for nonsymmetric problems and algebraic
- Oseen equations: $-\rho\mathbf{V}\cdot\nabla\mathbf{u} = -\nabla p + \mu\Delta\mathbf{u}$

$$-\nabla\cdot\mathbf{u} = 0$$

- solution in a perforated domain from CEA
- 10^{-10} outer tolerance, 100M d.o.f., 1k MPI processes



$\mu = 0.1$	outer solves	A_{00} solves	S solves
GenEO	5	50	8
harmonic	5	22	7

5×10^{-4}	outer solves	A_{00} solves	S solves
GenEO	†	†	†
harmonic	6	14	6

Conclusion

Final words

- multiphysics solvers
  - high level of abstraction of FreeFEM
  - versatility of PETSc
- composability of solvers
- `https://joliv.et/FreeFem-tutorial`

Final words

- multiphysics solvers
  - high level of abstraction of FreeFEM
  - versatility of PETSc
- composability of solvers
- `https://joliv.et/FreeFem-tutorial`
- let us know if things break (or work)
  - `https://community.freefem.org`
  - `https://petsc.org` + `https://slepc.upv.es`
  - `https://github.com/hpddm/hpddm/issues`

Final words

- multiphysics solvers
  - high level of abstraction of FreeFEM
  - versatility of PETSc
- composability of solvers
- `https://joliv.et/FreeFem-tutorial`
- let us know if things break (or work)
  - `https://community.freefem.org`
  - `https://petsc.org` + `https://slepc.upv.es`
  - `https://github.com/hpddm/hpddm/issues`

Thank you!

References 1

- Al Daas, Hussam, Pierre Jolivet, Frédéric Nataf, and Pierre-Henri Tournier (2024). **"A robust two-level Schwarz preconditioner for sparse matrices".** In: *SIAM Journal on Scientific Computing*, submitted for publication.
- Al Daas, Hussam, Pierre Jolivet, and Tyrone Rees (2023). **"Efficient algebraic two-level Schwarz preconditioner for sparse matrices".** In: *SIAM Journal on Scientific Computing* 45.3, A1199–A1213.
- Brune, Peter R., Matthew G. Knepley, Barry F. Smith, and Xuemin Tu (2015). **"Composing scalable nonlinear algebraic solvers".** In: *SIAM Review* 57.4, pp. 535–565.
- Cai, Xiao-Chuan and Marcus Sarkis (1999). **"A Restricted Additive Schwarz Preconditioner for General Sparse Linear Systems".** In: *SIAM Journal on Scientific Computing* 21.2, pp. 792–797.

References II

- 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, Jose E. Roman, and Stefano Zampini (2021). **"KSPHPDDM and PCHPDDM: extending PETSc with advanced Krylov methods and robust multilevel overlapping Schwarz preconditioners".** In: *Computers & Mathematics with Applications* 84, pp. 277–295. URL: `https://github.com/prj-/jolivet2020petsc`.

References III

- Nataf, Frédéric and Pierre-Henri Tournier (2022). **"Recent advances in domain decomposition methods for large-scale saddle point problems".** In: *Comptes Rendus. Mécanique*.
- Roman, Jose E., Fernando Alvarruiz, Carmen Campos, Lisandro Dalcin, Pierre Jolivet, and Alejandro Lamas Daviña (2023). **"Improvements to SLEPc in releases 3.14–3.18".** In: *ACM Transactions on Mathematical Software* 49.3, 29:1–29:11.
- Schwarz, Hermann (1870). **"Über einen Grenzübergang durch alternierendes Verfahren".** In: *Vierteljahrsschrift der Naturforschenden Gesellschaft in Zürich* 15, pp. 272–286.