Scalable parallel implementation of a coupling scheme using libMesh

Thiago MILANETTO SCHLITTLER, Régis COTTEREAU

Laboratoire MSSMat, CNRS, CentraleSupélec, Université Paris-Saclay

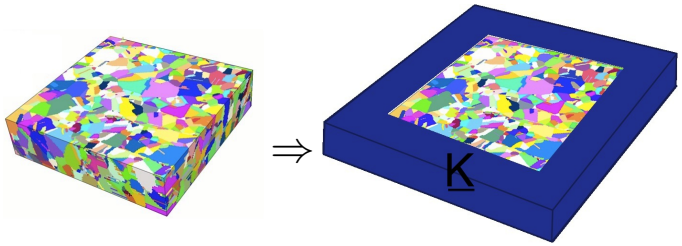
SIAM - CSE17

Funded by the French National Research Agency (projet number ANR-14-CE07-0007 CouESt)

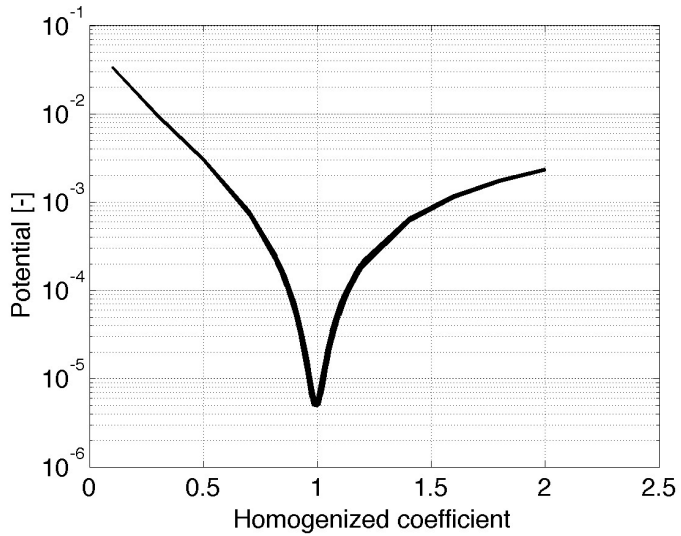
Motivation

Homogenization

Exchange known heterogeneous model by unknown homogeneous model



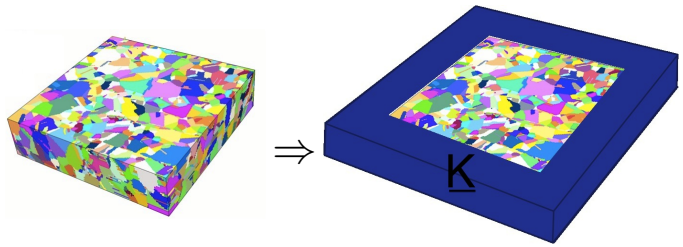
- Example:
  - **Micro:** heterogeneous, stochastic, non-linear model
  - **Macro:** homogeneous, deterministic model
    $\rightarrow$ homogenization of the micro model
- Models are linked through a coupling method



R. Cottereau, Int. J. Numer. Meth. Engng, 95, 1, 2013

Motivation

- Coupling method: take into account ...
  - different scales of the coupled models
  - non-conformal / incompatible meshes
- Latter is very important for stochastic models (random grain geometry realizations)



In this talk ...

- Present a scalable framework (**CArl**) to parallelize the coupling construction and to solve the coupled system

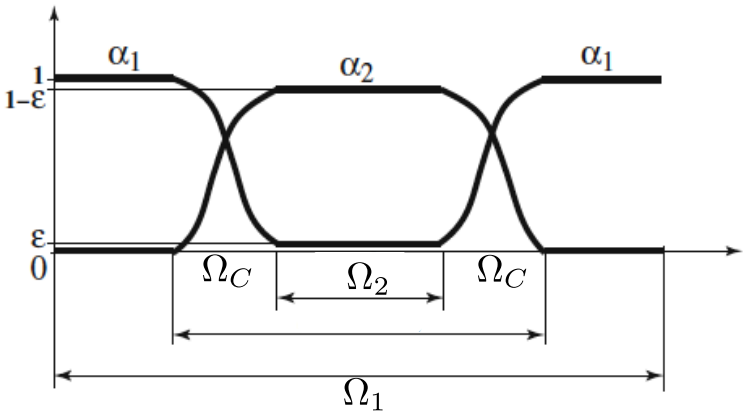
Outline

- Arlequin method: Description and formulation
- Intersection search: Parallelization and scaling
- Coupled solver: Implementation and scaling
- Example: linear heterogeneous anisotropic material
  - Same coupling formulation as non-linear model
  - Corresponds to one realization of a stochastic model

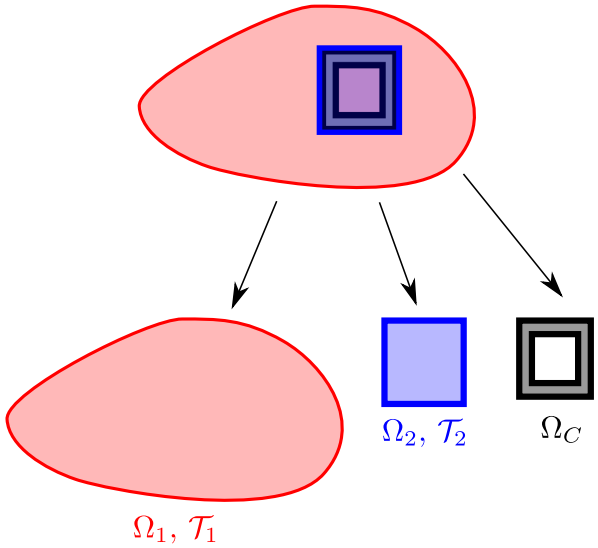
Other coupling methods have similar challenges (i.e. surface coupling)

Arlequin method

- Two overlapping models, defined on regions Ω_1, Ω_2
  - Meshes $\mathcal{T}_1$ and $\mathcal{T}_2$
- Coupling defined over region Ω_C
  - Volumetric coupling
  - Energy is partitioned between the models at the coupling region



D. Néron *et al.*, Comput Mech, 2016



Arlequin method

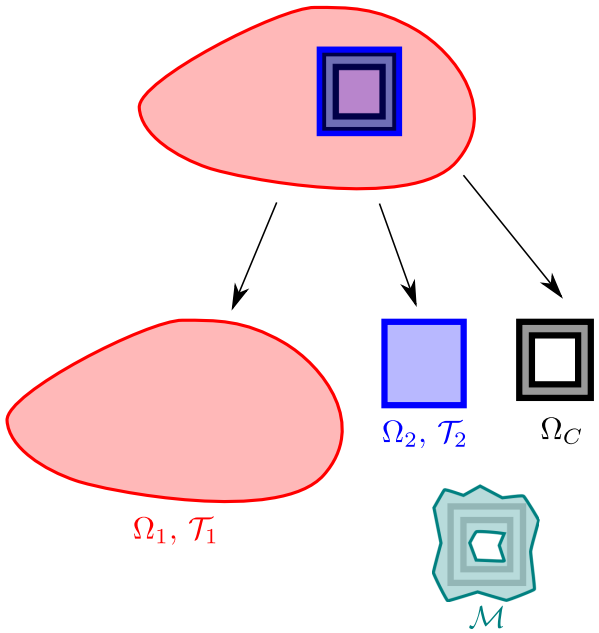
- Formulation:

Find $(\boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{\Phi}) \in \boldsymbol{V_1} \times \boldsymbol{V_2} \times \boldsymbol{M}$:

$$
\begin{aligned}
a_1(\boldsymbol{u}_1, \boldsymbol{v}) + c(\boldsymbol{\Phi}, \boldsymbol{v}) &= f_1(\boldsymbol{v}) \quad && \forall \boldsymbol{v} \in \boldsymbol{V_1} \\
a_2(\boldsymbol{u}_2, \boldsymbol{v}) - c(\boldsymbol{\Phi}, \boldsymbol{v}) &= f_2(\boldsymbol{v}) \quad && \forall \boldsymbol{v} \in \boldsymbol{V_2} \\
c(\boldsymbol{\psi}, \boldsymbol{u}_1 - \boldsymbol{u}_2) &= 0 \quad && \forall \boldsymbol{\psi} \in \boldsymbol{M}
\end{aligned}
$$

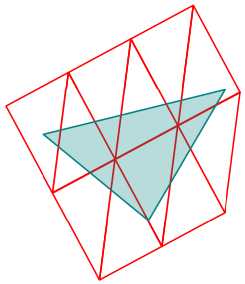
- Coupling uses a *mediator space* $\boldsymbol{M}$
  - mesh $\mathcal{M}$
  - $\boldsymbol{u}_2$, $\boldsymbol{\psi}$: stochastic models
  - Coupling operator

$$c(\boldsymbol{\psi}, \boldsymbol{v}) = \int_{\Omega^C} \boldsymbol{\kappa} \left(\boldsymbol{\varepsilon}(\boldsymbol{\psi}) : \boldsymbol{\varepsilon}(\boldsymbol{v}) + \frac{1}{e^2} \boldsymbol{\psi} \cdot \boldsymbol{v} \right) d\Omega$$

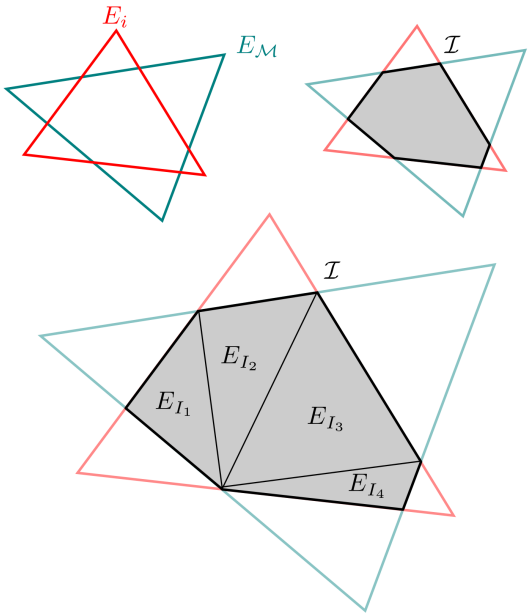


Arlequin method

$$c(\psi, \boldsymbol{v}) = \int_{\Omega^C} \kappa \left(\varepsilon(\psi) : \varepsilon(\boldsymbol{v}) + \frac{1}{e^2} \psi \cdot \boldsymbol{v} \right) d\Omega$$



- Spatial parts of ψ and $\boldsymbol{v}$ are from different meshes! Cannot use quadrature from either!
- Solution: intersection mesh between $\mathcal{T}_i$ and $\mathcal{M}$, **inside** Ω^C
- Only used to compute the integral



How to search and parallelize the intersections efficiently over hundreds of processors?

Intersection search: parallelization

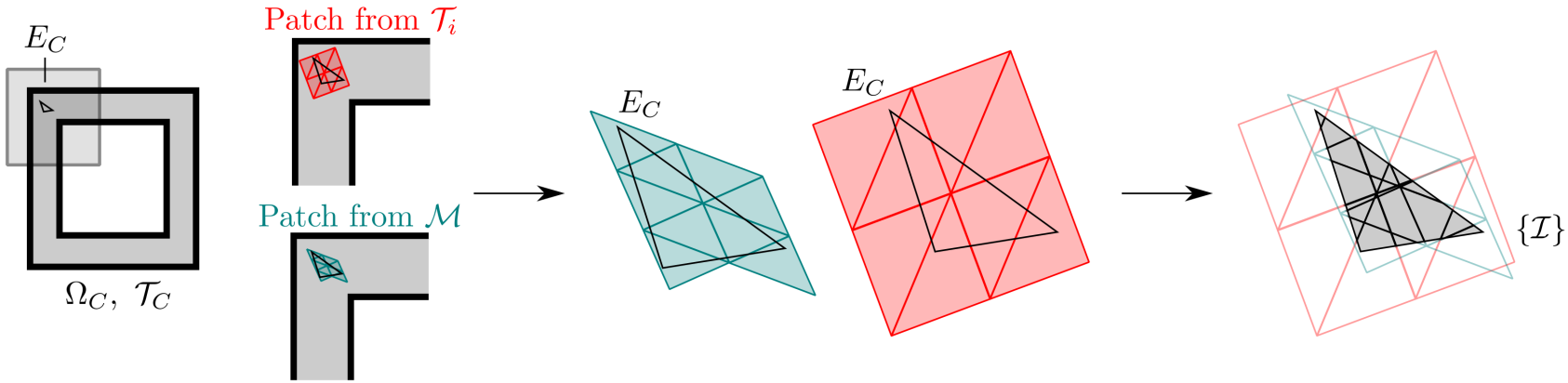
Serial search:

- Test all pairs: $O(n^2)$
- Advancing front: at best $O(n)$

M. Gander *et al.*, Domain Decomposition Methods in Science and Engineering XVIII, 2009

Parallel search: Distribute the work using Ω^C

1. Mesh the coupling region Ω^C: $\mathcal{T}_C$, n_C elems. $\rightarrow$ **No physical system associated!**
2. Build "patches" of $\mathcal{T}_i$ and $\mathcal{M}$ overlapping $E_C \in \mathcal{T}_C \rightarrow$ **Use advancing front!**
3. n_C separate problems: search all intersections between the patches, *inside* E_C

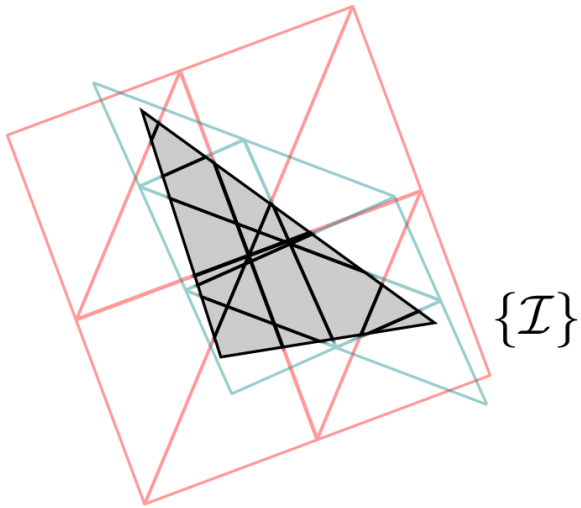


Intersection search: balancing over a computer cluster

- Library used to build intersections: CGAL

  http://www.cgal.org

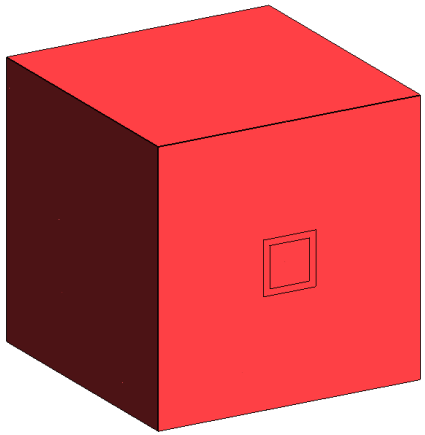
  - Exact geometry, no edge cases
  - Expensive to construct intersection polyhedrons between elements
- Building the set $\{\mathcal{I}\}$: **Expensive**
  Find nb. of intersections: **Cheap!**

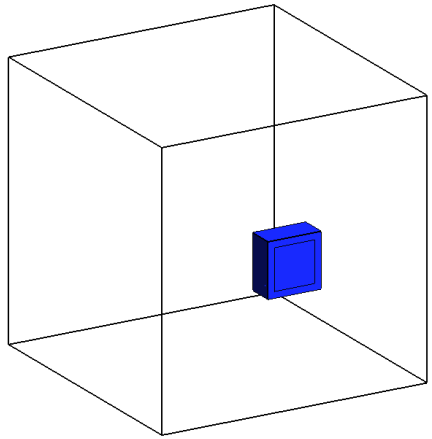


Balancing:

1. Do preliminary, non-balanced intersection *search*
2. Partition $\mathcal{T}_C$ using number of intersections as weight

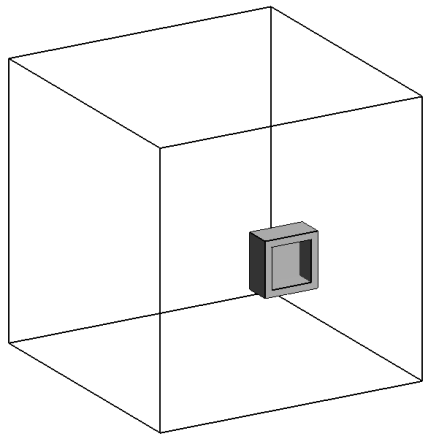
Intersection search: strong scaling





Elem.

$\mathcal{T}_1 : 1.75 \cdot 10^6$
$\mathcal{T}_2 : 4.38 \cdot 10^5$
$\mathcal{T}_C : 2.73 \cdot 10^3$
$\mathcal{T}_I : 6.66 \cdot 10^6$



Intersection search: strong scaling

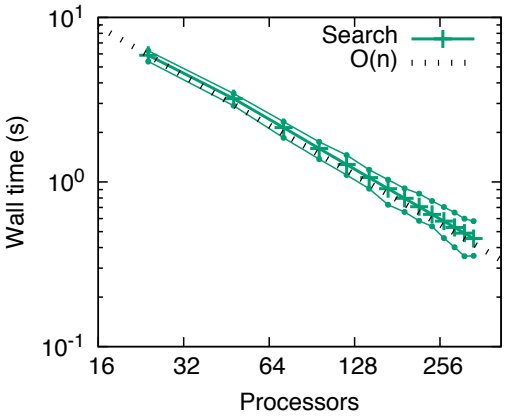


Figure: Search: test all pairs

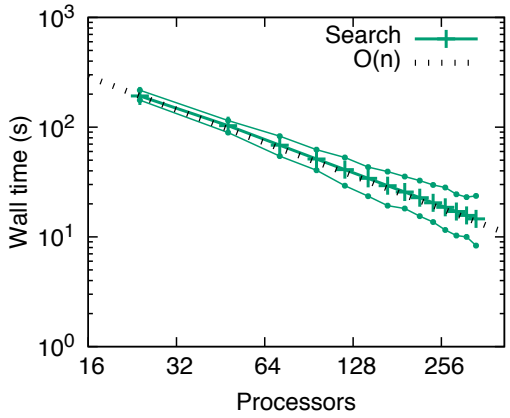


Figure: Search: advancing front

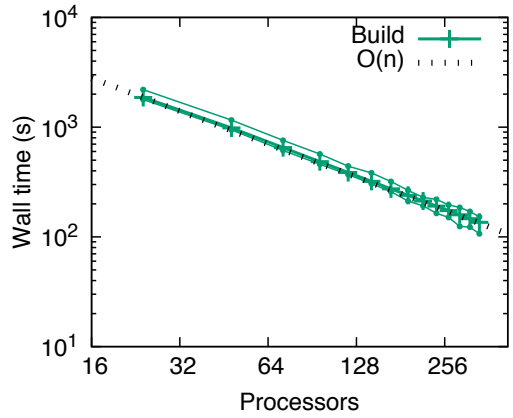


Figure: Build

Xeon E5-2670v3

Elem.
$\mathcal{T}_1$: $1.75 \cdot 10^6$
$\mathcal{T}_2$: $4.38 \cdot 10^5$
$\mathcal{T}_C$: $2.73 \cdot 10^3$
$\mathcal{T}_I$: $6.66 \cdot 10^6$

- Search time: $\sim 200x$ faster than construction (test all pairs)
- Nb. of elements of $\mathcal{T}_C$: $\sim 10\times$ nb. of processors
- Advancing front search
  - Needs a construction step $\rightarrow$ Slower

Intersection search: effects of repartitioning

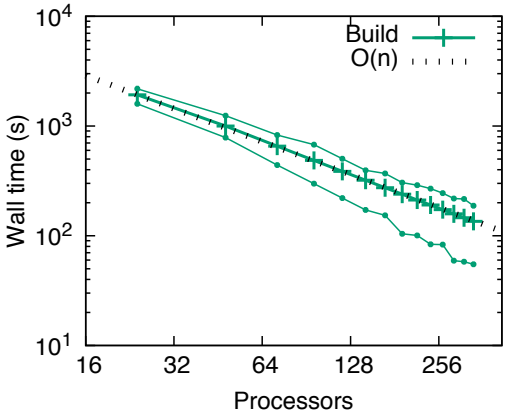


Figure: Build: no repartition

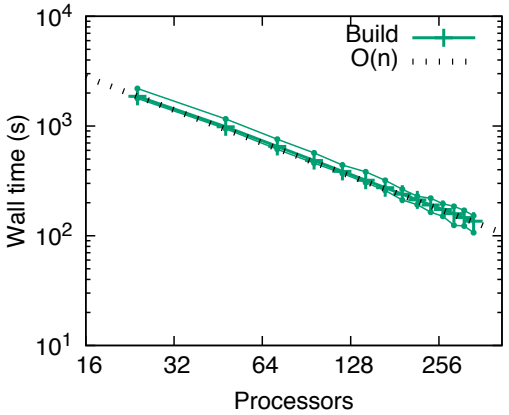


Figure: Build: with repartition

Xeon E5-2670v3

Elem.
$\mathcal{T}_1 : 1.75 \cdot 10^6$
$\mathcal{T}_2 : 4.38 \cdot 10^5$
$\mathcal{T}_C : 2.73 \cdot 10^3$
$\mathcal{T}_I : 6.66 \cdot 10^6$

- Repartitioning allows better distribution of intersection construction
- Worth the cost of an extra search step

Arlequin coupled solver, CArl

- Main component: `carl::coupled_system` class
- Adds a layer to libMesh's `EquationSystems` $\rightarrow$ `System` structure

```
std::pair<std::string, libMesh::EquationSystems*> m_macro_EqSys;
std::map<std::string, libMesh::EquationSystems*> m_micro_EqSys;
```

Similar mappings for other systems (intersections, mediator ...) and coupling matrices

- Solvers

`carl::coupled_solver` (LATIN, **FETI**)

`carl::generic_solver_interface` (system solvers, *can use external programs*)

- Coupling assemble functions, system parameters ...

CArl: FETI and Arlequin method

- Discretized coupled system: $\begin{bmatrix} \boldsymbol{K}_1 & \boldsymbol{0} & \boldsymbol{C}_1^t \\ \boldsymbol{0} & \boldsymbol{K}_2 & -\boldsymbol{C}_2^t \\ \boldsymbol{C}_1 & -\boldsymbol{C}_2 & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{u}_1 \\ \boldsymbol{u}_2 \\ \boldsymbol{\Phi} \end{bmatrix} = \begin{bmatrix} \boldsymbol{F}_1 \\ \boldsymbol{F}_2 \\ \boldsymbol{0} \end{bmatrix}.$

FETI solver, applied to the Arlequin method

1. Find the uncoupled solutions: $\boldsymbol{K}_i \cdot \boldsymbol{u}_i^0 = \boldsymbol{F}_i,\ i = 1, 2$
2. Solve the coupling correction equation:
   $\left(\boldsymbol{C}_1 \boldsymbol{K}_1^{-1} \boldsymbol{C}_1^t + \boldsymbol{C}_2 \boldsymbol{K}_2^{-1} \boldsymbol{C}_2^t\right) \cdot \boldsymbol{\Phi} = \boldsymbol{C}_1 \boldsymbol{u}_1^0 - \boldsymbol{C}_2 \boldsymbol{u}_2^0$
3. Add coupling correction to the solutions:
   $\boldsymbol{u}_1 = \boldsymbol{u}_1^0 - \boldsymbol{K}_1^{-1} \boldsymbol{C}_1^t \boldsymbol{\Phi} \qquad \boldsymbol{u}_2 = \boldsymbol{u}_2^0 + \boldsymbol{K}_2^{-1} \boldsymbol{C}_2^t \boldsymbol{\Phi}$

H. B. Dhia *et al.*, Multimodeling of multi-alterated structures in the Arlequin framework, European Journal of Computational Mechanics, 2008
C. Farhat *et al.*, A method of finite element tearing and interconnecting and its parallel solution algorithm, International Journal for Numerical Methods in Engineering, 1991

CArl: FETI / PCG solver

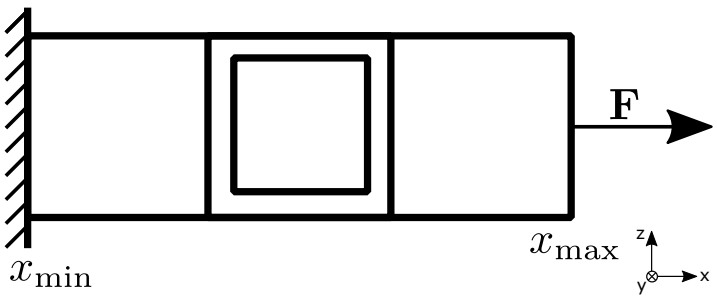
$$\underbrace{\left(\boldsymbol{C}_1\boldsymbol{K}_1^{-1}\boldsymbol{C}_1^t + \boldsymbol{C}_2\boldsymbol{K}_2^{-1}\boldsymbol{C}_2^t\right)}_{\boldsymbol{A}} \cdot \boldsymbol{\Phi} = \underbrace{\boldsymbol{C}_1\boldsymbol{u}_1^0 - \boldsymbol{C}_2\boldsymbol{u}_2^0}_{\boldsymbol{b}}$$

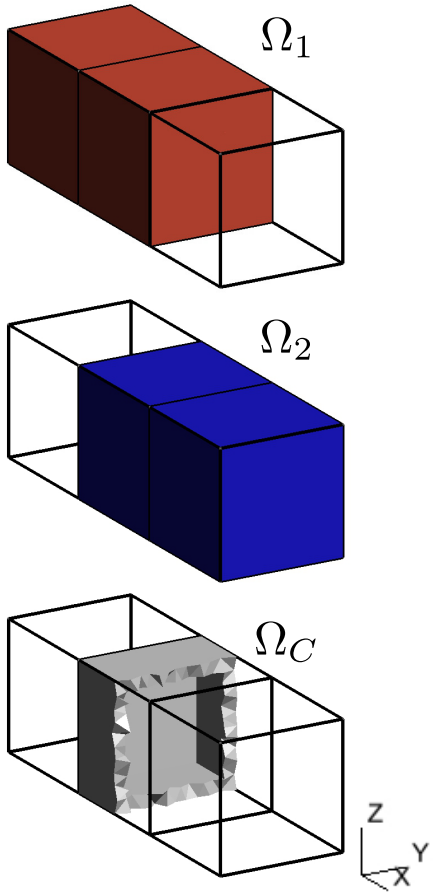
- Avoid explicit construction of $\boldsymbol{A}$: Use an iterative solver (CG), and apply $\boldsymbol{A}$ "by parts", calling the uncoupled system solvers
- Caveat: one of the models might not have Dirichlet boundary conditions $\rightarrow$ **ill-conditioned!**
  1. Use a **projected** CG algorithm (PCG)
  2. Rigid body modes correction: $\boldsymbol{u}_2 = \boldsymbol{u}_2^0 + \boldsymbol{K}_2^+\boldsymbol{C}_2^t\boldsymbol{\Phi} + \boldsymbol{R}_2 \cdot \boldsymbol{\alpha}$
- Possibly time-consuming components of the FETI / PCG solver:
  - Calls to model solvers
  - Preconditioner
  - Rigid body modes projectors (if a model has no boundary conditions)

C. Farhat *et al.*, A method of finite element tearing and interconnecting and its parallel solution algorithm, International Journal for Numerical Methods in Engineering, 1991

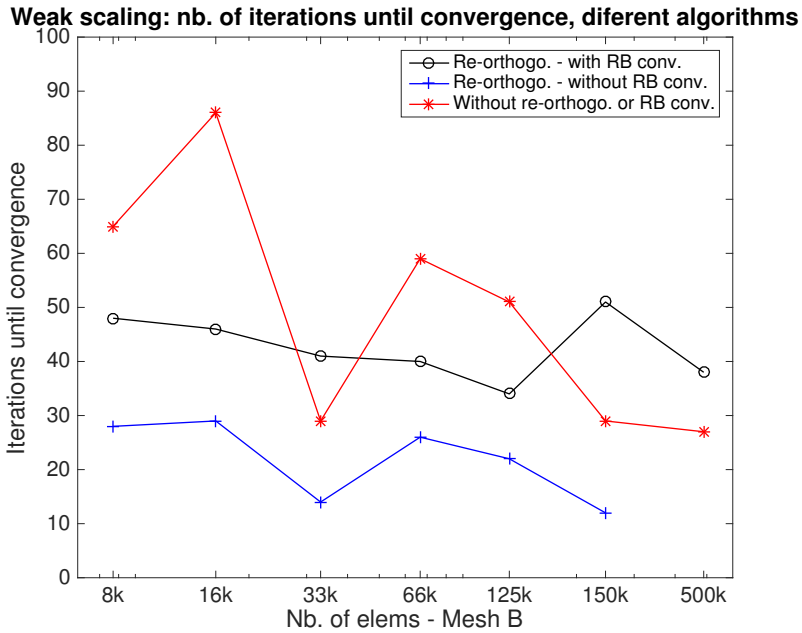
CArl: FETI / PCG solver

- Weak scaling of the coupled solver?
- Cost of each part?
- Tests done with traction test
  - **Macro system:** domain Ω_1
    - Clamped x_{min} face
    - Mediator mesh extracted from this system
  - **Micro system:** domain Ω_2
    - Force applied on x_{max} face
  - Homogeneous, linear elasticity





CArI / FETI: weak scaling



Elem.

$\mathcal{T}_1$: $5.0 \cdot 10^5$ $\quad$ $\mathcal{M}$: $1.4 \cdot 10^5$

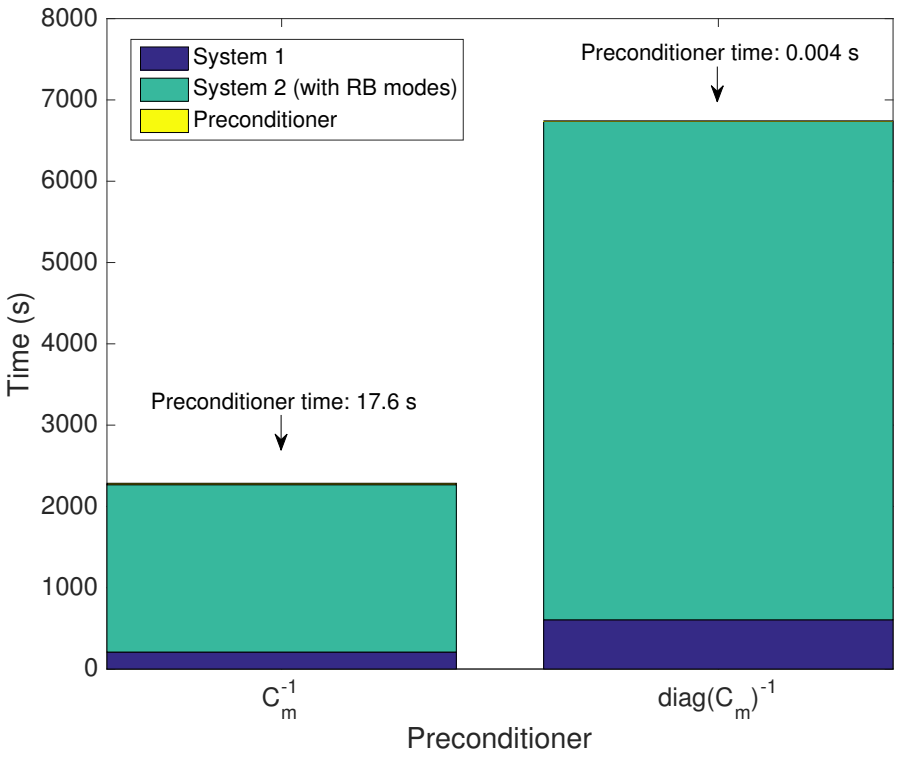
$\mathcal{T}_2$: $8.0 \cdot 10^4 \sim 5.0 \cdot 10^5$

($6 \sim 384$ procs.)

- Simulation time (and scaling) depends on choice of model solvers
- Weak scaling of only the FETI solver: **iterations until convergence**
- Meshes $\mathcal{T}_1$ and $\mathcal{M}$ kept unchanged
- Mesh $\mathcal{T}_2$: increse number of elems.
- Initial algorithm: erratic number of iterations
- Good scaling when checking convergence of $\boldsymbol{R}_2 \cdot \boldsymbol{\alpha}$

$$\boldsymbol{u}_2 = \boldsymbol{u}_2^0 + \boldsymbol{K}_2^+ \boldsymbol{C}_2^t \boldsymbol{\Phi} + \boldsymbol{R}_2 \cdot \boldsymbol{\alpha}$$

CArI / FETI: computational costs



384 procs. (Xeon E5-2670v3 @ 2.30 GHz)

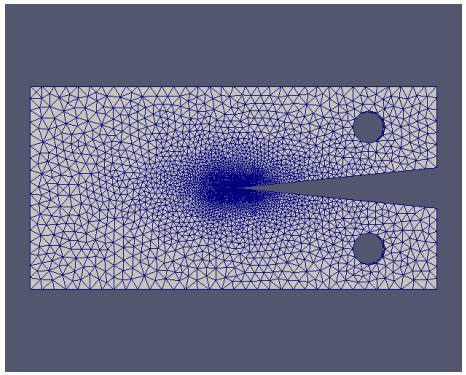
Elem.

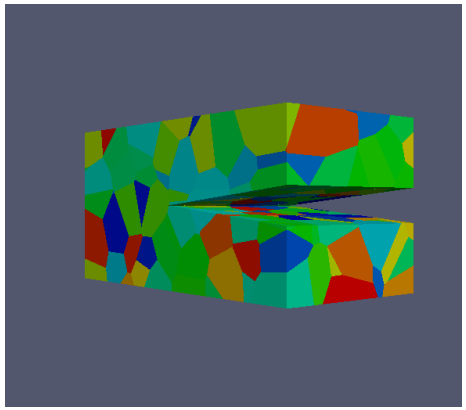
$\mathcal{T}_1, \mathcal{T}_2 : 1 \cdot 10^6 \quad \mathcal{M} : 2.7 \cdot 10^5$

- Solve time dominated by calls to model solvers
  - Sys. 1: $\sim 9\%$
  - Sys. 2: $\sim 90\%$
  - Projection: $\sim 0.006\%$
- Preconditioner based on coupling matrix $\boldsymbol{C}_m$
- Restricted to mediator space
- $\boldsymbol{C}_m^{-1}$: 46 iter. , $\sim 0.7\%$
- $\mathrm{diag}(\boldsymbol{C}_m)^{-1}$: 221 iter. , $\sim 0\%$

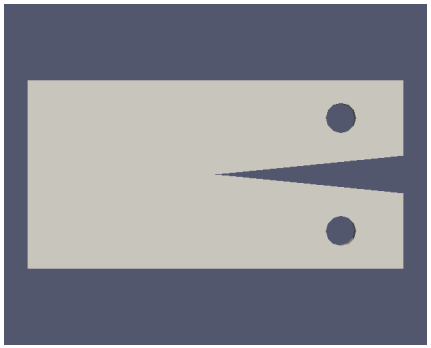
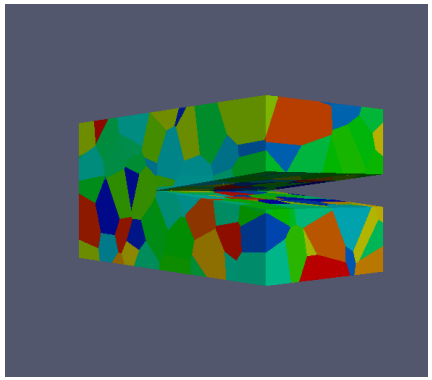
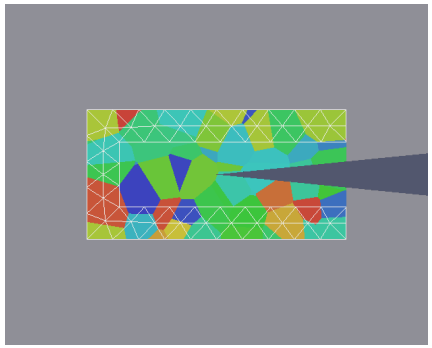
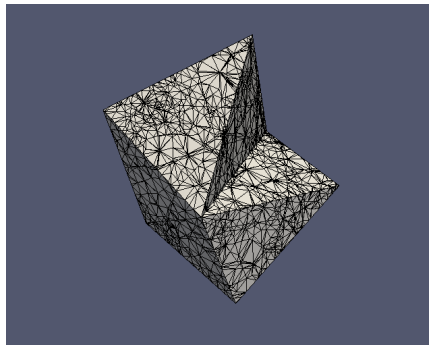
Example: linear heterogeneous anisotropic material

- **Macro system:**
  - Homogeneous, linear elasticity
  - $E = 200\,GPa$, $\mu = 80\,GPa$
  - Fissure on the xz plan
  - z displacements enforced on the holes
  - Clamped left surface
- **Micro system:**
  - Heterogeneous, anisotropic elasticity
  - $c_{11} = 198\,GPa$, $c_{12} = 125\,GPa$, $c_{44} = 122\,GPa$
  - Elasticity tensor: cubic symmetry, random grain orientations (isotropic distribution)
  - No direct boundary conditions / forces
  - Mesh generated from Voronoi diagram
  - 250 grains





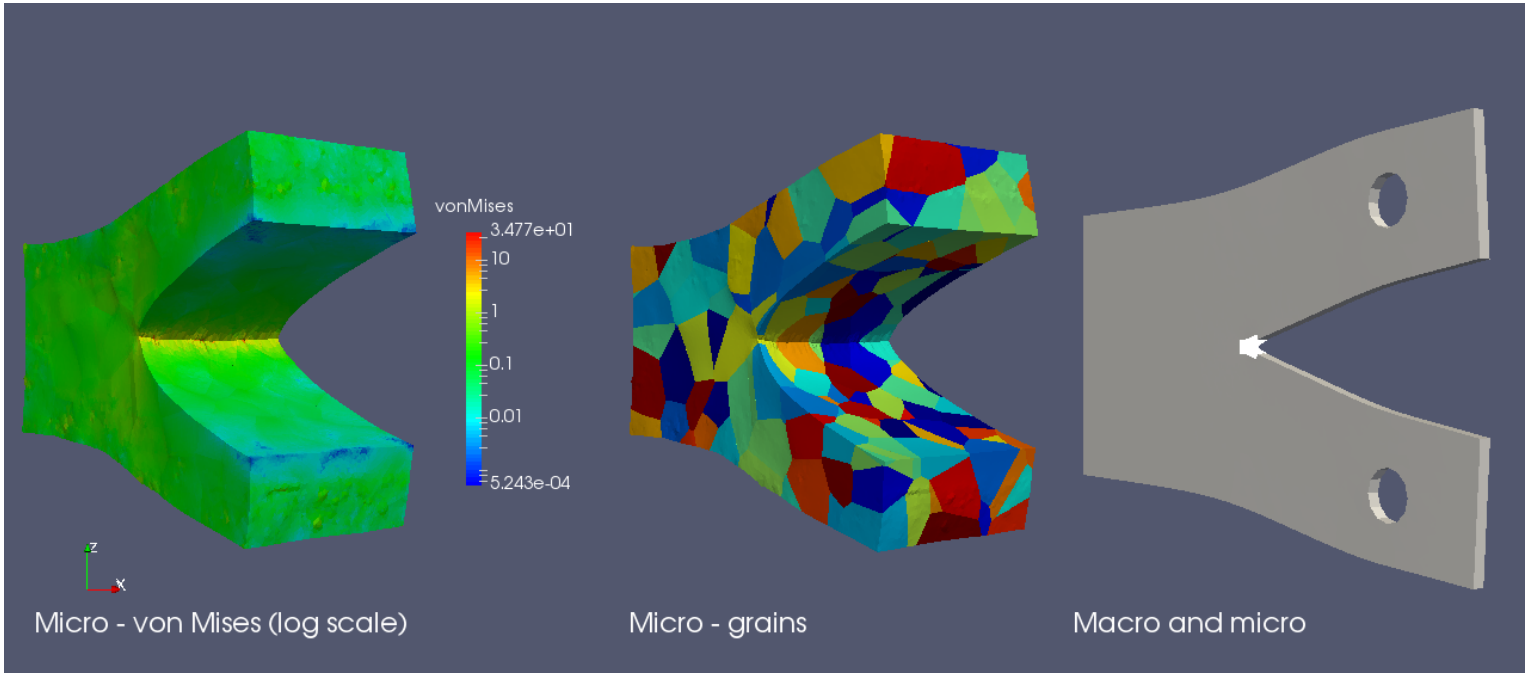
Example: linear heterogeneous anisotropic material

Meshes	Elem.
Macro (homogeneous):	$3.4 \cdot 10^4$
Micro (250 grains, anisotropic):	$1.1 \cdot 10^6$
Coupling region:	$1.1 \cdot 10^3$
Intersection mesh:	$7.2 \cdot 10^6$

- Macro mesh element length: $\sim 10x$ smaller near fissure
- Micro mesh element length: $\sim 10x$ smaller than the smallest macro system element

Example: linear heterogeneous anisotropic material



Coupling test

- 96 processors (Xeon E5-2670v3 @ 2.30 GHz), wall time
  - Intersections: $\sim$ **11 minutes** Solve: $\sim$**1 hour**

Conclusion

- Construction of a scalable algorithm to find the intersections between overlapping, non-conformal meshes
  - No need to adapt the models' meshes to build the coupling
  - No need to add construction constraints to the random meshes
- Development of a scalable, parallel coupled system solver

Current work

- Applying the current code to the homogenization problem
  - **Macro:** homogeneous, deterministic
  - **Micro:** stochastic, non-linear, anisotropic polycrystal

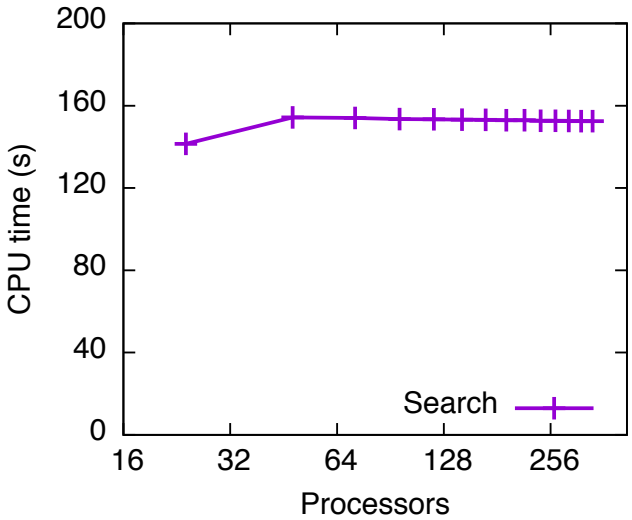
T. M. Schlittler, R. Cottereau, Fully scalable implementation of a volume coupling scheme for the modeling of polycrystalline materials, Computational Mechanics (article being finalized)

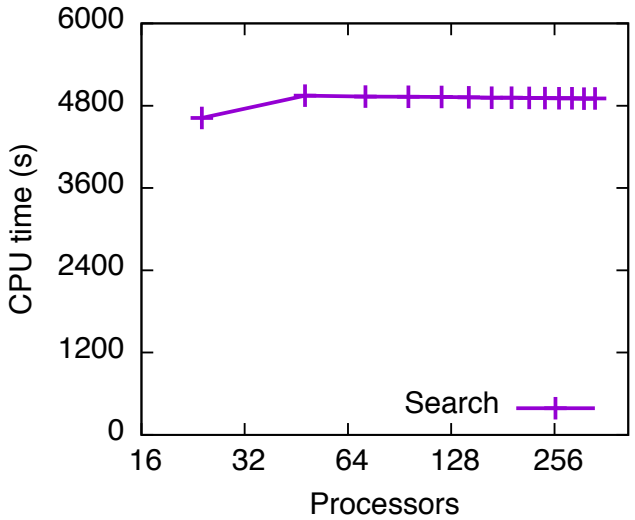
Github

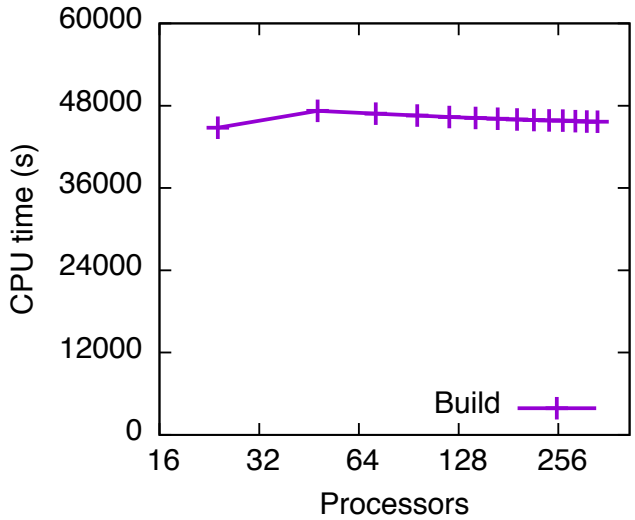
https://github.com/cottereau/CArl

Thanks for your attention!

Intersection search: scaling







Xeon E5-2695V2

@ 2.4GHz

Elem.
$\mathcal{T}_1$: $1.75 \cdot 10^6$
$\mathcal{T}_2$: $4.38 \cdot 10^5$
$\mathcal{T}_C$: $2.73 \cdot 10^3$
$\mathcal{T}_I$: $6.66 \cdot 10^6$

- Search time: $\sim 200x$ faster than construction (test all pairs)
- Nb. of elements of $\mathcal{T}_C$: $\sim 10\times$ nb. of processors
- Advancing front search
  - Needs a construction step $\rightarrow$ Slower