Fully scalable implementation of a volume coupling scheme for the modeling of multiscale materials

Thiago Milanetto Schlittler, Régis Cottereau

Noname manuscript No.
(will be inserted by the editor)

Fully scalable implementation of a volume coupling scheme for the modeling of multiscale materials

Thiago Milanetto Schlittler · Régis Cottereau

the date of receipt and acceptance should be inserted later

Abstract We present in this paper a new implementation of a multi-scale, multi-model coupling algorithm, with a proposed parallelization scheme for the construction of the coupling terms between the models. This allows one to study such problems with a fully scalable algorithm on large computer clusters, even when the models and/or the coupling have a high number of degrees of freedom. As an application example, we will consider a system composed by an homogeneous, macroscopic elasto-plastic model and an anisotropic polycrystalline material model, with a volume coupling based on the Arlequin framework.

Keywords Multimodel, Multiscale, Arlequin method, FETI method, Mesh intersection search

1 Introduction

Coupling methods, in general, allow the study of multi-scale systems, taking into account the different physical processes of each scale - for example, the coupling between continuous and granular mediums [31], and of heterogeneous wave propagation solvers [25,19]. These methods can be classified in several different types, depending on the desired application. Methods like the VMS [22] and the HMM [13], and similar others, are used when the parameters of the macro-scale model are not known, but depend on the micro-scale over the whole domain. They can be classified as embedding methods, with the VMS enriching the macro-scale with the micro-scale model over element patches, and the HMM altering the quadrature evaluation of the weak formulation. If, on the other hand, the macro-scale quantity of interest depends on the micro-scale only over a part of the former's domain, more local methods are used. Examples include the non-overlapping domain decomposition methods [2]. They are derived from domain decomposition techniques developed to solve numerically PDEs over large-scale computer clusters [12, 16], and they are used when the coupling is done over an interface between the models. These models use the following formulation: find $(\mathbf{u}_1, \mathbf{u}_2, \mathbf{\Phi}) \in \mathbf{W}_1 \times \mathbf{W}_2 \times \mathbf{M}$ such that

$$a_1(\mathbf{u}_1, \mathbf{v}_1) + c(\mathbf{\Phi}, \mathbf{v}_1) = \ell_1(\mathbf{v}_1), \qquad \forall \mathbf{v}_1 \in \mathbf{W}_1; \tag{1a}$$

$$a_2(\mathbf{u}_2, \mathbf{v}_2) - c(\mathbf{\Phi}, \mathbf{v}_2) = \ell_2(\mathbf{v}_2), \qquad \forall \mathbf{v}_2 \in \mathbf{W}_2; \tag{1b}$$

$$c(\boldsymbol{\psi}, \mathit{\Pi}_1 \mathbf{u}_1 - \mathit{\Pi}_2 \mathbf{u}_2) = 0, \qquad \forall \boldsymbol{\psi} \in \mathbf{M}, \tag{1c}$$

where the operators $\mathit{\Pi}_l : \mathbf{W}_l \to \mathbf{M}$ project functions from $\mathbf{W}_l$ into their restriction on the interface. The weak formulations of the coupled models (first and second equations) are altered by adding a coupling operator $c(\cdot, \cdot)$ and a Lagrange multiplier $\mathbf{\Phi}$, defined over a mediator space $\mathbf{M}$ and associated to the interface between the models. Let us note as $\mathbf{W}_l$ the spaces associated to each model l. The last equation guarantees that the solutions of the systems, when projected into $\mathbf{M}$, are continuous under the coupling operation. The formulation Eq. (1) can be solved iteratively by, for example, enforcing strongly the continuity condition, or by computing the Lagrange multiplier as an intermediary step [16].

For overlapping domains, still in the context of a limited domain coupling, methods based on volume couplings can be used, such as the Arlequin framework [3, 5, 28] and the bridging domain method [32]. They keep the same formulation as Eq. (1) but differ from the interface methods by defining the term $c(\cdot, \cdot)$ as a volume

MSSMat (CentraleSupélec/CNRS/Université Paris-Saclay), Grande Voie des Vignes, 92295 Châtenay-Malabry Cedex, France
E-mail: regis.cottereau@centralesupelec.fr

coupling over the overlapping domains. These methods differ mainly on the choice of this coupling term. In both cases, $c(\cdot,\cdot)$ is built using an intermediary mesh, and a specific effort must be set on integrating functions borne by incompatible meshes: that related to each of the models, and that of the coupling operator $c(\cdot,\cdot)$.

The problem of meshing geometrical intersections is also present in methods where a volume is cut by an interface, such as with Nitsche's method [17,23,15] or the eXtended Finite Element Method [27,7,11]. In the former method, two overlapping meshes are considered: one representing an object, and another representing its surroundings - which is altered to exclude the overlapping between both meshes. To do this operation, one needs to identify the geometric intersections between the interface of the object mesh and the background mesh, and re-mesh the intersecting elements of the latter. In the latter method, functions with discontinuities (or with discontinuous derivatives) along a given interface have to be integrated over a volume mesh. Both these methods and the overlapping coupling techniques therefore share similar characteristic features in terms of integration, which are discussed in more detail in [26].

Generally speaking, the coupling step is not parallelized for methods following the formulation Eq. (1), since the mediator space is, in many cases, small when compared to the models' spaces associated to it. By consequence, while the steps associated with each model scale well when parallelized, the serial coupling step breaks this scalability of the algorithm as a whole. Previous works [4,14] using the coupling algorithms and parallelism focused on simulating several couplings between a single global model and many local models, with each coupling being associated to one processor - hence each coupling is serial. Here, we will use the same theoretical framework as presented in ref. [4,14], but we will focus on an implementation with parallelized and scalable coupling step, possible due to a detailed work on all steps of the implementation.

As an example of this implementation, we will study a coupled system formed by a macroscopic and homogeneous elastic model and a microscopic anisotropic elastic model, representing a polycrystalline material. While we will focus on this example here, this framework - and thus this implementation - can be applied to other cases involving multi-scale physics, such as the ones cited above. Similarly, we insist that, while we focus here in the Arlequin framework, the parallelization scheme presented here is applicable to any coupling methods using the formulation Eq. (1).

This paper is structured on the following manner. We will start by defining our physical problem, and by recalling the Arlequin framework and the FETI domain decomposition method [16], which was adapted to solve the Arlequin problem in refs. [4,14]. We will follow this with a discussion on the coupling operators - including our implementation of a parallel intersection search algorithm, needed to assemble them. This part should be seen as the main contribution of this article, ensuring the actual scalability (in practice) of a known numerical coupling method (the FETI / Arlequin solver). Finally, we will present in the last section our numerical results, including the scalability tests of the algorithms and an application case with a crack test involving homogeneous and heterogeneous elasticity models.

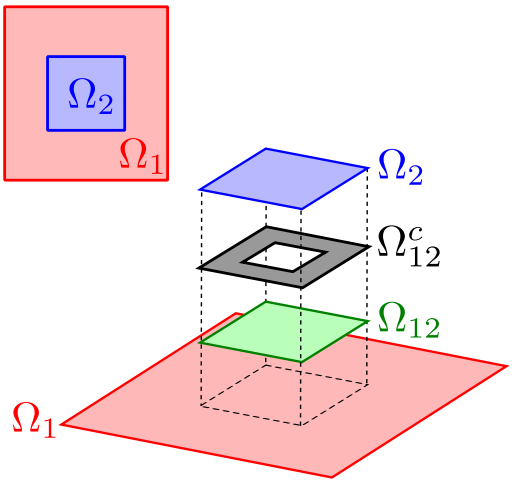


Figure 1: (color online) Domains used in the Arlequin framework. The model domains are Ω_1 and Ω_2, the overlapping domain is Ω_{12}, and the coupling domain (marked in gray) is Ω_{12}^c.

2 Arlequin framework

2.1 Formulation

We will consider here two linear elastic models, each associated to bounded regular domains Ω_1 and Ω_2, and which overlap over a region Ω_{12}. The latter is decomposed into two non-overlapping regions, $\Omega_{12} = \Omega_{12}^c \cup \Omega_{12}^f$, with Ω_{12}^c defining the coupling domain between the two models (see Fig. 1). Furthermore, let us note as Γ_u and Γ_F, respectively, the clamped region of Ω_1, and its region where a field of surface density forces $\mathbf{F}$ is applied.

The functional spaces associated to each model are

$$\mathbf{W}_1 = \left\{\mathbf{v} \in \mathcal{H}^1(\Omega_1); \mathbf{v} = 0 \text{ on } \Gamma_u\right\},$$
$$\mathbf{W}_2 = \left\{\mathbf{v} \in \mathcal{H}^1(\Omega_2)\right\}.$$

The internal and external works $a_l(\mathbf{u}_l, \mathbf{v}_l)$ and $\ell_l(\mathbf{v}_l)$ are weighted by weight parameter functions (α_1, α_2) de-

fined in Ω_1 and Ω_2, respectively. These weights guarantee the partitioning of the energy between the models on the coupling region, and they follow the relations

$$\begin{aligned} &\alpha_l \geq 0 \text{ in } \Omega_l,\ l = 1,2, \\ &\alpha_l = 1 \text{ in } \Omega_l \setminus \Omega_{12}, \\ \alpha_1 + \alpha_2 = 1 &\text{ in } \Omega_{12}. \end{aligned} \tag{2}$$

The weighted works are defined as

$$\forall (\mathbf{u}_1, \mathbf{v}_1) \in \mathbf{W}_1 \times \mathbf{W}_1, \tag{3}$$

$$a_1(\mathbf{u}_1, \mathbf{v}_1) = \int_{\Omega_1} \alpha_1 \boldsymbol{\sigma}(\mathbf{u}_1) : \boldsymbol{\varepsilon}(\mathbf{v}_1)\, d\Omega,$$

$$\ell_1(\mathbf{v}_1) = \int_{\Omega_1} \alpha_1 \mathbf{f} \cdot \mathbf{v}_1 d\Omega + \int_{\Gamma_F} \alpha_1 \mathbf{F} \cdot \mathbf{v}_1 dS,$$

and

$$\forall (\mathbf{u}_2, \mathbf{v}_2) \in \mathbf{W}_2 \times \mathbf{W}_2,$$

$$a_2(\mathbf{u}_2, \mathbf{v}_2) = \int_{\Omega_2} \alpha_2 \boldsymbol{\sigma}(\mathbf{u}_2) : \boldsymbol{\varepsilon}(\mathbf{v}_2)\, d\Omega,$$

$$\ell_2(\mathbf{v}_2) = \int_{\Omega_2} \alpha_2 \mathbf{f} \cdot \mathbf{v}_2 d\Omega.$$

In these equations, $\mathbf{f}$ denotes the field of volume density forces applied on both model domains. $\boldsymbol{\sigma}(\mathbf{v}_l)$ and $\boldsymbol{\varepsilon}(\mathbf{v}_l)$ are the linearized strain and stress tensors, associated to the displacement field $\mathbf{v}_l$, and linked through Hooke's law.

The coupled problem involving the two systems follows the formulation presented in Eq. (1), with the coupling operator being defined over the coupling domain Ω_{12}^c. The mediator space, noted by $\mathbf{M}$, is defined as $\mathbf{M} = \left\{\boldsymbol{\Phi} \in \mathcal{H}^1(\Omega_{12}^c)\right\}$. The coupling terms $c(\cdot,\cdot)$ link this mediator space to either of the models spaces, or to itself (in the case of the third equation of Eq. (1)) and it follows

$$\forall (\boldsymbol{\Psi}, \boldsymbol{\Phi}) \in \mathbf{M} \times \mathbf{V}, \mathbf{V} \in \{\mathbf{W}_1, \mathbf{W}_2, \mathbf{M}\}$$

$$c(\boldsymbol{\Psi}, \boldsymbol{\Phi}) = \int_{\Omega_{12}^c} \boldsymbol{\kappa} \left(\boldsymbol{\varepsilon}(\boldsymbol{\Psi}) : \boldsymbol{\varepsilon}(\boldsymbol{\Phi}) + \frac{1}{e^2} \boldsymbol{\Psi} \cdot \boldsymbol{\Phi} \right) d\Omega. \tag{4}$$

The parameters $\boldsymbol{\kappa}$ and e are of the order of magnitude of the material's rigidity and the width of the coupling domain. The coupling term is, then, analogous to a stiffness.

The domains Ω_1 and Ω_2 can be discretized by associating to them, respectively, the meshes $\mathcal{T}_1$ and $\mathcal{T}_2$, each one with n_1 and n_2 degrees of freedom (DoF's). The formulation Eq. (1) can then be written as

$$\mathbf{K}_1 \mathbf{u}_1 + \mathbf{C}_1^T \boldsymbol{\Phi} = \mathbf{F}_1, \tag{5a}$$

$$\mathbf{K}_2 \mathbf{u}_2 - \mathbf{C}_2^T \boldsymbol{\Phi} = \mathbf{F}_2, \tag{5b}$$

$$\mathbf{C}_1 \mathbf{u}_1 - \mathbf{C}_2 \mathbf{u}_2 = 0. \tag{5c}$$

Following standard notation, $\mathbf{K}_l$ and $\mathbf{F}_l$, $l = 1, 2$, are the matrices and vectors representing the internal and external virtual works of the model l, with dimensions $n_l \times n_l$ and n_l, respectively. The coupling terms $\mathbf{C}_l$ are $n_m \times n_l$ matrices, where n_m is the number of DoF's of $\mathcal{M}$. Each element of this matrix is given by

$$\mathbf{C}_{l,ij} = \int_{\Omega_{12}^c} \boldsymbol{\kappa} \left(\boldsymbol{\varepsilon}(\nu_i^m) : \boldsymbol{\varepsilon}(\nu_j^l) + \frac{1}{e^2} \nu_i^m \cdot \nu_j^l \right) d\Omega, \tag{6}$$

where ν_i^m and ν_j^l are form functions associated respectively to the coupling region and the model l.

If the models' and the mediator form functions can be interpolated exactly by a common function space (as we will do here, in Sec. 4), the coupling matrices $\mathbf{C}_l$ can be written as the products of a $n_m \times n_m$ coupling matrix, $\mathbf{C}_m$ and $n_m \times n_l$ interpolation matrices $\mathbf{P}_l$:

$$\mathbf{C}_l = \mathbf{C}_m \mathbf{P}_l. \tag{7}$$

This new coupling matrix is defined only in the mediator space $\mathbf{M}$, and is given by

$$\mathbf{C}_{m,ij} = \int_{\Omega_{12}^c} \boldsymbol{\kappa} \left(\boldsymbol{\varepsilon}(\nu_i^m) : \boldsymbol{\varepsilon}(\nu_j^m) + \frac{1}{e^2} \nu_i^m \cdot \nu_j^m \right) d\Omega, \tag{8}$$

where now all the form functions are associated to the coupling region.

3 Solving the Arlequin problem: FETI solver

The system (5) can be solved as a monolithic problem, but this approach has the limitation of not allowing the usage of the proper solvers of the super-imposed models. One can think, for example, of the coupling between models such as linear / nonlinear, deterministic / stochastic, continuum / atomistic ... . Some works have been proposed to solve this problem, adapting domain decomposition methods such as the FETI [16,4,14] or the LATIN [28] methods. In the context of these adaptations, each domain of the decomposition corresponds to a different model domain, Ω_l, and the domain interfaces correspond to the coupling region Ω_{12}^c.

We focus in this article on a Arlequin solver based on the FETI method. This domain decomposition method, essentially, solves a given model inside each of the domains of the decomposition, and then calculates corrections due to the continuity between these domains. It has a formulation similar to the one presented in Eq. (1), with the differences that the binary matrices connecting the domains (noted as $\mathbf{B}_i$, in ref. [16]) are exchanged by the coupling matrices $\mathbf{C}_i$.

An application of this method to the Arlequin problem is presented in refs. [4,14], where the different models and the model couplings correspond, respectively, to the domains of the decomposition and their interfaces. We will present in this section a summarized version of this application, restricted to two coupled models. This will be followed by more detailed description of a solver for the coupling corrections, with a strong focus on which of its components affect the solver's scalability.

3.1 FETI solver for well conditioned matrices

If both matrices $\mathbf{K}_1$ and $\mathbf{K}_2$ are well conditioned, the system (5) can be rewritten in the following manner. By isolating the displacements in the first and second equations, we have

$$\mathbf{u}_1 = \mathbf{u}_1^0 - \mathbf{K}_1^{-1}\mathbf{C}_1^T\mathbf{\Phi}, \tag{9}$$
$$\mathbf{u}_2 = \mathbf{u}_2^0 + \mathbf{K}_2^{-1}\mathbf{C}_2^T\mathbf{\Phi}, \tag{10}$$

where $\mathbf{u}_1^0$ and $\mathbf{u}_2^0$ are the solutions of the decoupled models:

$$\mathbf{K}_l\mathbf{u}_l^0 = \mathbf{F}_l, \qquad l = 1, 2. \tag{11}$$

Substituting Eqs. (9) and (10) into Eq. (5c), we obtain a system depending only on the Lagrange multiplier $\mathbf{\Phi}$:

$$\left(\mathbf{C}_1\mathbf{K}_1^{-1}\mathbf{C}_1^T + \mathbf{C}_2\mathbf{K}_2^{-1}\mathbf{C}_2^T\right)\cdot\mathbf{\Phi} = \left(\mathbf{C}_1\mathbf{u}_1^0 - \mathbf{C}_2\mathbf{u}_2^0\right) \tag{12}$$
$$\mathbf{A}\mathbf{\Phi} = \mathbf{b}. \tag{13}$$

The solution $(\mathbf{u}_1, \mathbf{u}_2, \mathbf{\Phi})$ can then be found by first solving the decoupled systems (11), then solving the coupled system (12), and finally adding the coupling corrections to the models through Eqs. (9) and (10).

3.2 FETI solver with a singular matrix, $\mathbf{K}_2$

The method above must be altered if one of the models has a singular matrix $\mathbf{K}_l$ (for example, in the context of linear elasticity, if the model has no Dirichlet boundary conditions). Let us suppose here that the second model falls in this case and that $\mathbf{K}_2$ is singular. Eq. (5b) still admits solutions if $\mathbf{K}_2$ is consistent. In this situation, Eq. (10) is substituted by

$$\begin{aligned}\mathbf{u}_2 &= \mathbf{K}_2^+\left(\mathbf{F}_2 + \mathbf{C}_2^T\mathbf{\Phi}\right) + \mathbf{R}_2\boldsymbol{\alpha} \\ &= \mathbf{u}_2^{0,+} + \mathbf{K}_2^+\mathbf{C}_2^T\mathbf{\Phi} + \mathbf{R}_2\boldsymbol{\alpha}\end{aligned} \tag{14}$$

where $\mathbf{K}_2^+$ is the pseudo-inverse of $\mathbf{K}_2$ (a $n_2 \times n_2$ matrix such that $\mathbf{K}_2\mathbf{K}_2^+\mathbf{K}_2 = \mathbf{K}_2$). The matrix $\mathbf{R}_2$ is a $n_2 \times n_2^{RB}$ matrix, whose columns form a basis of $\mathbf{K}_2$'s null space, with n_{RB} vectors. $\boldsymbol{\alpha}$ is a n_2^{RB} vector defining a linear combination of this basis. In the context of the elastic models that we are considering here, $\mathbf{K}_2$'s null space is formed by the rigid body modes that are left "free" due to the lack of Dirichlet boundary conditions on the model.

The system (5b) admits at least one solution of the form (14) if and only if the term $\left(\mathbf{F}_2 + \mathbf{C}_2^T\mathbf{\Phi}\right)$ is orthogonal to $\mathbf{K}_2$'s null space. This condition is represented by the relation

$$\mathbf{R}_2^T\left(\mathbf{F}_2 + \mathbf{C}_2^T\mathbf{\Phi}\right) = \mathbf{0}. \tag{15}$$

Substituting Eq. (9) and (14) into Eq. (5c), we obtain a system depending on the Lagrange multiplier and the null space coefficients $\boldsymbol{\alpha}$:

$$\mathbf{A}^+\mathbf{\Phi} + \mathbf{R}_2^I\boldsymbol{\alpha} = \mathbf{b}^+, \tag{16}$$

where

$$\mathbf{A}^+ = \left(\mathbf{C}_1\mathbf{K}_1^{-1}\mathbf{C}_1^T + \mathbf{C}_2\mathbf{K}_2^+\mathbf{C}_2^T\right), \tag{17}$$
$$\mathbf{R}_2^I = \mathbf{C}_2\mathbf{R}_2, \text{ and} \tag{18}$$
$$\mathbf{b}^+ = \left(\mathbf{C}_1\mathbf{u}_1^0 - \mathbf{C}_2\mathbf{u}_2^{0,+}\right). \tag{19}$$

Together with (15), this equation can be written in matrix form as

$$\begin{bmatrix}\mathbf{A}^+ & \mathbf{R}_2^I \\ {\mathbf{R}_2^I}^T & \mathbf{0}\end{bmatrix}\cdot\begin{bmatrix}\mathbf{\Phi} \\ \boldsymbol{\alpha}\end{bmatrix} = \begin{bmatrix}\mathbf{b}^+ \\ -\mathbf{R}_2^T\mathbf{F}_2\end{bmatrix}. \tag{20}$$

3.3 Projected conjugate gradient algorithm

The system (20) is symmetric and non-singular, and hence it admits an unique solution $(\mathbf{\Phi}, \boldsymbol{\alpha})$. Still, this system is also indefinite, so an iterative method such as the conjugate gradient algorithm (CG) cannot be directly applied. Instead, a *projected* CG algorithm can be used [16,20]. We will present here a preconditioned and re-orthogonalized implementation of this algorithm, used originally in [4,14].

The algorithm itself is presented in Alg. 1. From a numerical point of view, it consists on applying the CG method to the equation $\mathbf{A}^+\cdot\mathbf{\Phi} = \mathbf{b}^+$, but modified in such a way that the constraint (15) is satisfied at each iteration. This can be done by choosing an initial solution guess, $\mathbf{\Phi}^{(0)}$, that follows this constraint, and imposing that the descent directions $\mathbf{p}^{(k)}$ are inside the null space of $\mathbf{R}_2^I$. This guarantees that $\mathbf{R}_2^I\mathbf{\Phi}^{(k)} = \mathbf{R}_2^I\mathbf{\Phi}^{(0)}$, and thus Eq. (15) is satisfied. A suitable choice of $\mathbf{\Phi}^{(0)}$ is

$$\mathbf{\Phi}^{(0)} = -\mathbf{R}_2^I\left({\mathbf{R}_2^I}^T\mathbf{R}_2^I\right)^{-1}{\mathbf{R}_2}^T\cdot\mathbf{F}_2, \tag{21}$$

and the descent direction condition can be achieved by applying an orthogonal projector operator,

$$\mathbf{\Pi}_R = \left[\mathbf{I}_{n_2^{RB}} - \mathbf{R}_2^I \left(\mathbf{R}_2^{I\,T}\mathbf{R}_2^I\right)^{-1}\mathbf{R}_2^{I\,T}\right] \tag{22}$$

on the vectors used to calculate the search directions. After the constrained value of $\mathbf{\Phi}$ is found, $\boldsymbol{\alpha}$ can be calculated from the system's residual $\mathbf{b}^+ - \mathbf{A}^+\mathbf{\Phi}$, using Eq. (16):

$$\boldsymbol{\alpha} = \left(\mathbf{R}_2^{I\,T}\mathbf{R}_2^I\right)^{-1}\mathbf{R}_2^{I\,T}\left(\mathbf{b}^+ - \mathbf{A}^+\mathbf{\Phi}\right). \tag{23}$$

3.4 Implementation of the projected CG algorithm

Some parts of the projected CG algorithm deserve a more detailed description. These include the numerical optimizations needed to avoid the explicit construction of the matrix operators $\mathbf{A}^+$ and $\mathbf{\Pi}_R$, the effects of numerical fluctuations, a proper choice of a preconditioner, and the choice of the convergence criteria. We will focus now on these points.

Indirect operator application: Two steps of the Alg. 1, involving the operators $\mathbf{A}^+$ and $\mathbf{\Pi}_R$, at lines 7 and 11, are potential performance bottlenecks. These operators involve the evaluation of inverse (dense) matrices, and thus we want to avoid their explicit construction.

In the case of the system matrix $\mathbf{A}^+$, each application of a term $\mathbf{C}_l\mathbf{K}_l^{-1}\mathbf{C}_l^T$ can be exchanged by a multiplication by $\mathbf{C}_l^T$, a call to the model l's solver, and a multiplication by $\mathbf{C}_l$ - avoiding the explicit construction of the inverse operator. The same is true for the terms with the pseudo-inverse, but in this case one must use a version of the corresponding model's solver that takes the null space in consideration. For a linear elasticity model, the PETSc Krylov solvers can do so using the `MatNullSpaceCreateRigidBody` and the `MatSetNullSpace` functions, which create and attach the rigid body modes to a given matrix, respectively. Another approach in the same context would consist for the user to indicate a set of additional Dirichlet boundary conditions blocking the rigid body modes while keeping the same amount of work on the rest of the displacement field.

For the projection operation, a similar decomposition onto a series of operations can be used. In this case, the $n_2^{RB} \times n_2^{RB}$ matrix $\mathbf{R}_2^{I\,T}\mathbf{R}_2^I$ is small enough to be inverted explicitly: in the worst case for a 3D linear elasticity model, it is a (6×6) matrix. Again due to this small size, the inverted matrix can be stored and multiplied locally, avoiding a communication bottleneck when Alg. 1 is used in parallel. A similar procedure can be followed for Eqs. (21) and (23), where this inverse matrix also appears.

Algorithm 1: `ProjectedPCG`: preconditioned and re-orthogonalized Conjugated Gradient algorithm, modified to project the search direction into the null space of the matrix $\mathbf{R}_2^I$. The inputs are the linear system's matrix and right-hand side, $\mathbf{A}^+$ and $\mathbf{b}^+$ (given here by Eqs. (17) and (19)). The preconditioner operator is represented by the matrix $\mathbf{M}_{PC}$, and the projection operator $\mathbf{\Pi}_R$ is given by Eq. (22).

Input:
System matrix: $\mathbf{A}^+ = \left(\mathbf{C}_1\mathbf{K}_1^{-1}\mathbf{C}_1^T + \mathbf{C}_2\mathbf{K}_2^+\mathbf{C}_2^T\right)$
System right-hand side: $\mathbf{b} = (\mathbf{C}_1 u_1^0 - \mathbf{C}_2 u_2^0)$

Output:
Converged solution $\mathbf{\Phi}^{k_f}$

`/* Initialization */`
1 Initial solution: $\mathbf{\Phi}^{(0)} = -\mathbf{R}_2^I\left(\mathbf{R}_2^{I\,T}\mathbf{R}_2^I\right)^{-1}\mathbf{R}_2{}^T \cdot \mathbf{F}_2$;
2 $\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}^+ \cdot \mathbf{\Phi}^{(0)}$;
3 $\mathbf{z}^{(0)} = \mathbf{\Pi}_R \cdot \mathbf{M}_{PC} \cdot \mathbf{\Pi}_R \cdot \mathbf{r}^{(0)}$;
4 $\rho^{(0)} = \left(\mathbf{r}^{(0)}, \mathbf{z}^{(0)}\right)_2$;
5 $\mathbf{p}^{(0)} = \mathbf{z}^{(0)}$;
6 **for** $k = 0,\ 1,\ 2\ldots$ *until convergence* **do**
7 $\quad\mathbf{q}^{(k)} = \mathbf{A}^+ \cdot \mathbf{p}^{(k)}$;
8 $\quad\gamma^{(k)} = \rho^{(k)} / \left(\mathbf{p}^{(k)}, \mathbf{q}^{(k)}\right)_2$;
9 $\quad\mathbf{\Phi}^{(k+1)} = \mathbf{\Phi}^{(k)} + \gamma^{(k)}\mathbf{p}^{(k)}$;
10 $\quad\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \gamma^{(k)}\mathbf{q}^{(k)}$;
$\quad$`/* Preconditioning and projection */`
11 $\quad\mathbf{z}^{(k+1)} = \mathbf{\Pi}_R \cdot \mathbf{M}_{PC} \cdot \mathbf{\Pi}_R \cdot \mathbf{r}^{(k+1)}$;
12 $\quad\rho^{(k+1)} = \left(\mathbf{r}^{(k+1)}, \mathbf{z}^{(k+1)}\right)_2$;
$\quad$`/* reorthogonalization of the descent */`
13 $\quad$**for** $i = 0,\ 1,\ 2\ldots k$ **do**
14 $\quad\quad\beta_i = \left(\mathbf{z}^{(k+1)}, \mathbf{q}^{(i)}\right)_2 / \left(\mathbf{p}^{(i)}, \mathbf{q}^{(i)}\right)_2$;
15 $\quad\mathbf{p}^{(k+1)} = \mathbf{z}^{(k+1)} - \sum_{i=0}^{k}\beta_i\mathbf{p}^{(i)}$;
16 $\quad$`CheckConvergence();`

Effects of numerical fluctuations: A classical CG algorithm relies on the fact that the descent directions $\left\{\mathbf{p}^{(i)}\right\}$ are orthogonal to each other, under the dot product associated to the matrix $\mathbf{A}^+$. In the case of the projected CG algorithm presented in Alg. 1, another condition is imposed on these vectors: that they are inside the null space of $\mathbf{R}_2^I$. This guarantees that the condition (15) is followed, and thus that the solution (14) exists.

Due to numerical fluctuations, though, these properties are not followed exactly. In the case of the first condition, this results only into a slower algorithm convergence. More importantly, in the case of the second con-

dition, these numerical errors result in incorrect rigid body mode corrections (the term $\mathbf{R}_2\boldsymbol{\alpha}$ in Eq. (14)), posing a serious problem to the robustness of the projected CG algorithm.

One way to avoid the first effect is to re-orthogonalize the descent directions (line 12 of Alg. 1). This change increases the memory cost of the algorithm, since we must save all the vectors $\{\mathbf{q}^{(i)}\}$. But, since system associated to the coupling space is the smallest one of the coupled problem, this should not be an issue. In the case of the second condition, the application of the projection operator directly on the residual (line 11 of Alg. 1), together with the reorthogonalization, guarantee by construction that the new descent direction $\mathbf{p}^{(k+1)}$ is inside $\mathbf{R}_2^I$'s null space.

Convergence condition: Usually, the convergence condition followed by a CG algorithm takes into account the relative or absolute convergence of the residual $\mathbf{r}^{(k)} = \mathbf{b}^+ - \mathbf{A}^+\boldsymbol{\Phi}^{(k+1)}$, by calculating its norm under the preconditioner dot product

$$\rho^{(k)} = \left(\mathbf{r}^{(k)}, \mathbf{M}_{PC}\mathbf{r}^{(k)}\right), \tag{24}$$

which tends to zero as we approach the exact solution. Due to the projection operations of the projected CG algorithm, though, this norm does not tend to zero. A more suitable choice of norm is the projected and preconditioned dot product, used in line 11 of Alg. 1:

$$\rho^{(k)} = \left(\boldsymbol{\Pi}_R\mathbf{r}^{(k)}, \cdot\mathbf{M}_{PC}\cdot\boldsymbol{\Pi}_R\mathbf{r}^{(k)}\right). \tag{25}$$

Furthermore, while we might reach a convergence of the solution $\boldsymbol{\Phi}^{(k)}$, this does not guarantee that the rigid body modes correction, $\mathbf{R}_2\boldsymbol{\alpha}$, has converged. The function `CheckConvergence()`, at the line 16 of Alg. 1, must then be altered to check the convergence of this correction term. Here, we chose a relative convergence test, comparing two consecutive values of the the corrections:

$$\text{abs}(|\mathbf{R}_2\boldsymbol{\alpha}^{(k+1)}|_2 - |\mathbf{R}_2\boldsymbol{\alpha}^{(k)}|_2) < \epsilon_{RB}|\mathbf{R}_2\boldsymbol{\alpha}^{(k+1)}|_2. \tag{26}$$

The effects of this choice of convergence check will be discussed in the section 6.1, together with the effects of the reorthogonalization.

Preconditioner: Following the ref. [4,14], we used a preconditioner $\mathbf{M}_{PC}$ based on the coupling matrix $\mathbf{C}_m$. Let us justify, from a mechanical point of view, this choice. Using Eq. (7), the matrix $\mathbf{A}^+$ can be rewritten in the following manner:

$$\mathbf{A}^+ = \left[\mathbf{C}_m\left(\mathbf{P}_1\mathbf{K}_1^{-1}\mathbf{P}_1^T\right) + \mathbf{C}_m\left(\mathbf{P}_2\mathbf{K}_2^+\mathbf{P}_2^T\right)\right]\mathbf{C}_m. \tag{27}$$

Recall that the coupling matrix $\mathbf{C}_m$ is analogous to a stiffness matrix with rigidity similar to those of the materials studied, and note that the terms $\left(\mathbf{P}_1\mathbf{K}_1^{-1}\mathbf{P}_1^T\right)$ and $\left(\mathbf{P}_2\mathbf{K}_2^+\mathbf{P}_2^T\right)$ correspond to interpolations of the inverses of stiffness matrices into the coupling region. The matrix $\mathbf{A}^+$ can then, in the context of choosing a preconditioner, be roughly approximated by $\mathbf{C}_m$.

The most straightforward preconditioner choice is the full inverse of the coupling matrix, $\mathbf{M}_{PC} = (\mathbf{C}_m)^{-1}$, but its efficient usage depends on solving yet another equation system inside the main `for` loop of the projected CG algorithm. This system is defined in the mediator mesh $\mathcal{M}$, though, and in most cases it should be relatively cheap to solve it, when compared to the full model's systems. If this is not the case, another option is the usage of a Jacobi-like preconditioner, with $\mathbf{M}_{PC} = [\text{diag}\,(\mathbf{C}_m)]^{-1}$. This choice avoids solving a new equation system, but is a rougher approximation. We tested and compared both choices (together with simulations without the preconditioner), and the results are presented in section 6.3.

3.5 Other domain decomposition methods

As we said previously, other domain decomposition methods can be used to solve the Arlequin problem, and the choice depends on the application and the method's strengths. One of such is the LATIN method, which is applied to the Arlequin problem in ref. [28]. This method has the advantage of not being susceptible to the rigid body modes problem seen for the FETI method, since its "decoupled" step changes the rigidity matrices (Eq. (33) in the reference). Due to this, it is compatible with external solvers that do not project out the rigid body modes. Furthermore, it is directly compatible with the proper generalized decomposition method [28]. On the other hand, due to these same changes to the rigidity matrices, the LATIN / Arlequin solver is arguably more intrusive: either the external solver must accept the (possibly dense) modifications of the rigidity matrices, either the external system must be solved internally using an algorithm similar to Alg. 1, using a $\mathbf{A}\cdot\mathbf{x}$ operation from the external solver.

4 Coupling matrix assembly and parallelization

Let us discuss now the parallelization of the discretized formulation construction, Eq. (5). The matrices $\mathbf{K}_1$, $\mathbf{K}_2$ and the vectors $\mathbf{F}_1$, $\mathbf{F}_2$ can be constructed in parallel using FEM libraries such as libMesh [24]. We will present in this section the parallelized construction of the coupling matrices $\mathbf{C}^l$. From Eq. (6), we have that these matrices involve the integration of form functions associated to different, incompatible meshes, $\mathcal{M}$ and $\mathcal{T}_l$.

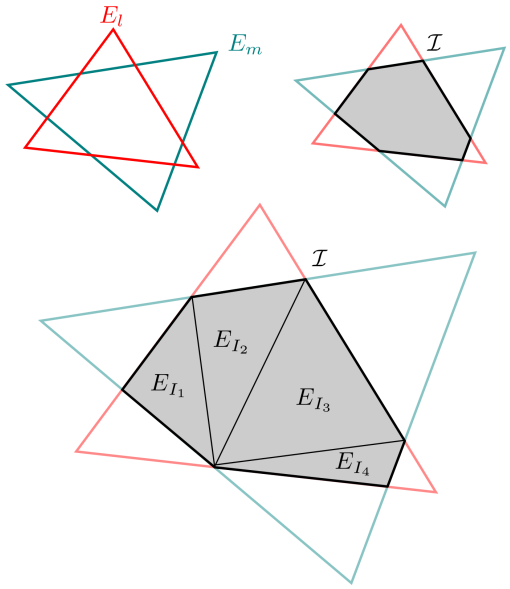


Figure 2: (color online) 2D example of an intersection between two elements $E_m \in \mathcal{M}$ (blue) and $E_l \in \mathcal{T}_l$ (red), corresponding to a non-zero coupling term $\mathbf{C}^l_{i,j}$. The intersection $\mathcal{I}$ (gray) does not follow the geometry of either of these two meshes, and thus it must be triangulated to allow the calculation of the coupling term.

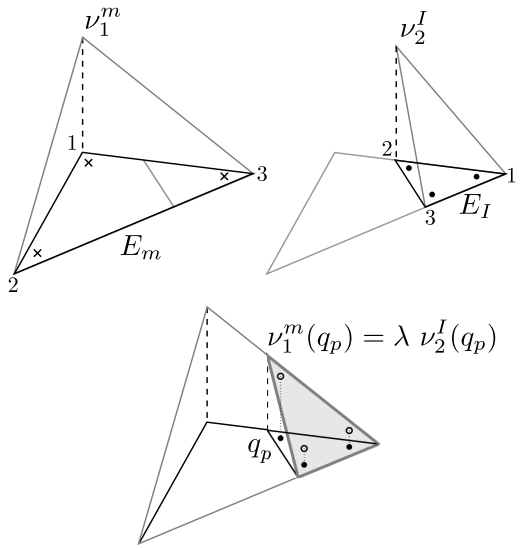


Figure 3: 2D example of the projection of the evaluation of a form function ν_1^m of the element E_m at a quadrature point q_p of the intersection element E_I. In the case represented, $\nu_1^m(q_p)$ is equal to the form function ν_2^I of the intersection elements, evaluated at q_p and rescaled by a constant λ.

We must, then, define a common mesh, over which we can project these form functions, to calculate the integrals using a Gaussian quadrature. This common mesh, which we will note $\mathcal{I}_m$, must be constructed before the proper assembly of the coupling matrices.

The form functions ν_i^m and ν_j^l used in Eq. (6) are defined on the elements $E_m \in \mathcal{M}$ and $E_l \in \mathcal{T}_l$, respectively. For two meshes formed by three dimensional elements, the coupling terms $\mathbf{C}^l_{i,j}$ will be non-zero only if their intersection defines a polyhedral region $\mathcal{I}$ (i.e., it will be zero if the intersection is empty or if it defines a lower dimensionality entity, such as a point or a surface). The integral Eq. (6) must be calculated over this intersection, but in the general case it does not follow the element geometry of either $\mathcal{M}$ or $\mathcal{T}_l$ (see Fig. 2 for a 2D example), and we cannot calculate the integrals using directly the original form functions.

We need, then, to decompose each region $\mathcal{I}$ into a form that can be treated using a Gaussian quadrature. This can be done by triangulating each intersection region $\mathcal{I}$, defining an element set $\{E_I\}$, Fig. 2. The union of all these intersection triangulations defines an intersection mesh, $\mathcal{I}_m$. Each element E_I is associated to a set of quadrature points $\{q_p\}$ and a form function basis $\{\nu_k^I\}$. The latter can be used to rewrite the form functions ν_i^m and ν_j^l. For linear form functions, this results in linear combinations,

$$\nu_i^m(q_p) = \sum_k \lambda_k^{i,m} \nu_k^I(q_p), \quad \nu_j^l(q_p) = \sum_{k'} \lambda_{k'}^{j,l} \nu_{k'}^I(q_p), \tag{28}$$

where the weights λ depend on the positions of the intersection elements' quadrature points inside the elements E_m and E_l. See Fig. 3 for a visualization of Eq. (28) for 2D elements.

The coupling terms $\mathbf{C}^l_{i,j}$ can be rewritten by inserting the linear decompositions of ν_i^m and ν_j^l into Eq. (6). The result is a sum of integrals:

$$\mathbf{C}^l_{i,j} = \sum_I \sum_{k,k'} \lambda_k^{i,m} \lambda_{k'}^{j,l} \mathbf{C}^I_{k,k'},$$
$$\mathbf{C}^I_{k,k'} = \int_{\Omega^c_{12}} \boldsymbol{\kappa} \left(\boldsymbol{\varepsilon}\left(\nu_k^I\right) : \boldsymbol{\varepsilon}\left(\nu_{k'}^I\right) + \frac{1}{e^2} \nu_k^I \cdot \nu_{k'}^I \right) d\Omega. \tag{29}$$

Each $\mathbf{C}^I_{k,k'}$ corresponds to an integral involving the form functions associated to an intersection element E_I, and is defined on its region. It can, then, be calculated using its quadrature points. Since these integrals are associated to the intersection mesh $\mathcal{I}_m$, it is reasonable to use its partitioning over all the processors to distribute the calculations. Another possibility would be to use the partitioning of either $\mathcal{M}$ and $\mathcal{T}_l$, which determines the distribution of $\mathbf{C}^l$'s degrees of freedom, but doing so

would leave idle processors, since not all of their parts are involved in the coupling.

Alg. 2 shows a possible implementation of the parallelized coupling matrix assembly. It takes as input three meshes, the system mesh $\mathcal{T}_l$, the mediator mesh $\mathcal{M}$ and the intersection mesh $\mathcal{I}_m$, and an intersection table $iTbl$. This table encodes the relations between these three meshes, associating to each element $E_I \in \mathcal{I}_m$ the pair of parent intersecting elements $\{E_m, E_l\}$, $E_m \in \mathcal{M}$ and $E_l \in \mathcal{T}_l$. The parallelized loop on line 5 of Alg. 2 uses this information to calculate the contribution of E_I using Eq. (29). Finally, the output is the parallel coupling matrix $\mathbf{C}^l$, with its row and column degrees of freedom following, respectively, the partitioning of $\mathcal{M}$ and $\mathcal{T}_l$. For simplicity sake, we suppose in Alg. 2 that all the processors have the data of all the elements E_m and E_l. Some FEM libraries do this by default to reduce the number of communications. If this becomes too memory intensive, reduced versions of $\mathcal{M}$ and $\mathcal{T}_l$, containing only elements that intersect the Ω_{12}^c coupling region, can be used.

Algorithm 2: `BuildCoupling` : Parallelized construction of the coupling matrix $\mathbf{C}^l$, following Eq. (29). The inputs are the system mesh $\mathcal{T}_l$, the mediator mesh $\mathcal{M}$, the intersection mesh $\mathcal{I}_m$, and the intersection table $iTbl$. The latter contains, for each element $E_I \in \mathcal{I}_m$ the intersecting pair of elements $\{E_m, E_l\}$, with $E_m \in \mathcal{M}$ and $E_l \in \mathcal{T}_l$. The output is the coupling matrix $\mathbf{C}^l$.

Input:
System mesh $\mathcal{T}_l$, mediator mesh $\mathcal{M}$
Intersection mesh $\mathcal{I}_m$, intersection table $iTbl$

Output:
Coupling matrix $\mathbf{C}^l$

1 Partition the meshes $\mathcal{T}_l$, $\mathcal{M}$;
2 Partition the mesh $\mathcal{I}_m$ and the intersection table $iTbl$;
3 Associate $\mathbf{C}^l$ rows to $\mathcal{M}$'s DoF's;
4 Associate $\mathbf{C}^l$ columns to $\mathcal{T}_l$'s DoF's;
5 **oneach** *processor p* **do**
6 Local parts of $\mathcal{I}_m$: $\mathcal{I}_m^p$;
7 **foreach** *element* $E_I \in \mathcal{I}_m^p$ **do**
  `/* Get the intersecting element pair */`
8 $\{E_m, E_l\} = iTbl[E_I]$;
9 **foreach** *DoF pair* (i, j) *of* $\{E_m, E_l\}$ **do**
10 Transform the form functions ν_i^m and ν_j^l to E_I's form function basis;
11 Calculate the contribution of E_I;
12 Add it to $\mathbf{C}^l_{i,j}$;
13 **return** $\mathbf{C}^l$

5 Mesh intersection search

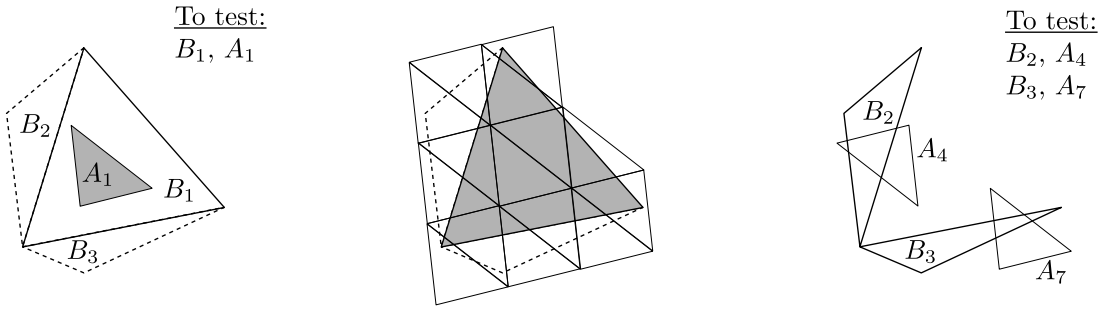
The construction of each operator $\mathbf{C}^l$ depends on building all the intersections between the two model's meshes. In general, it is useful to separate this build process into two parts: a search step, which returns a list of intersecting elements, and a build step, which takes this list and build the geometrical intersection. The latter step is essentially the same, independently from the choice of search algorithm.

Several serial search algorithms for two meshes exist in the literature [29][26][33][18], but we must take into account the fact that our intersections are restricted to the coupling region Ω_{12}^c. This region can be represented by a third mesh, $\mathcal{T}_C$, and the problem can be formulated in a general form as: build all the intersections between three given meshes, $\mathcal{T}_A$, $\mathcal{T}_B$ and $\mathcal{T}_C$. The mesh $\mathcal{T}_C$, though, does not have the same status as the other two, since it has no physical system associated to it. We will present in this section a parallel algorithm that takes this into account to solve the intersection search problem, but first we will describe some serial search methods, upon which our algorithm is based. Note that, as will be detailed below, the most critical component to guarantee the efficiency of the parallel algorithm is the distribution of the intersection construction workload over the processors.

5.1 Serial mesh intersection search

A straightforward method to find the intersections between two meshes would be to test each one of the element pairs $(E_A,\ E_B)$ for an intersection, resulting in an algorithm with complexity $O(|\mathcal{T}_A|\,|\mathcal{T}_B|)$, where $|\cdot|$ is the number of elements of a mesh. More efficient algorithms organize the bounding boxes of the elements into hierarchical spatial data structures [29][26], or use advancing front methods and neighbor information [18] to reduce the number of operations. In any case, the resulting set of n_I polyhedrons must be checked for intersections with the third mesh $\mathcal{T}_C$. This step is effectively a second intersection search, and its complexity can be reduced by saving, during the first step, only the intersections found inside the region Ω_{12}^c.

We will focus here mainly on the advancing front method from ref. [18]. This algorithm takes advantage of the neighbor structure of the meshes by noting that, if it is known that two elements $A_1 \in \mathcal{T}_A$ and $B_1 \in \mathcal{T}_B$ intersect, then A_1's neighbors are likely to also intersect B_1 or B_1's neighbors. An implementation adapted to include a restriction to a convex region Ω_C (which can be, for example, the bounding box of $\mathcal{T}_C$) is presented in Alg. 3. It works as follows:



(a) (b) (c)

Figure 4: Intersection search: (a) algorithm starts at the intersection between A_1 and B_1. (b) It explores the region around the initial intersection and finds all the intersections near it (gray area). (c) During this process, B_1's neighbors (here, B_2 and B_3) are tested for intersections with the elements of $\mathcal{T}_A$ being explored, and any pairs found are added to a list for the next iteration - in this case, $\{B_2, A_4\}$ and $\{B_3, A_7\}$ were added.

Algorithm 3: `AFIntersectionSearch`: Intersection search algorithm based on ref. [18]. The inputs are the two meshes to be tested, $\mathcal{T}_A$ and $\mathcal{T}_B$, and the region over which we want to restrict the intersections, Ω_C. The list $TestA$ is a list of elements of $\mathcal{T}_A$ to be tested for intersections with the element B_t. The output is a table $iPairs$, containing the intersecting pairs $\{A_t, B_t\}$.

Input:
Meshes $\mathcal{T}_A$ and $\mathcal{T}_B$, convex region Ω_C
Output:
Intersection pairs table $iPairs$

```
1  List of pairs to test: TestPairs;
2  List of elements from T_B already treated: TreatedB;
3  Search first intersection inside Ω_C:
     {A_1, B_1}, A_1 ∈ T_A, B_1 ∈ T_B;
4  TestPairs.insert({A_1, B_1});
5  while TestPairs is not empty do
6      Elems. from T_A to test: TestA;
7      List of new pairs: NewPairs;
8      {A_init, B_t} = TestList.pop();
9      TestA.insert(A_init);
10     while TestA is not empty do
11         Elements from T_A already treated: TreatedA;
12         A_t = TestA.pop();
           /* Test for intersection, and update
              iPairs if positive                    */
13         UpdtInter(A_t, B_t, Ω_C, iPairs);
           /* Add A_t's neighbors to test list      */
14         UpdtTest(A_t, TestA, TreatedA) ;
           /* Test B_t's neighbors for new pairs    */
15         UpdtPairs(A_t, B_t, Ω_C, NewPairs, TreatedB);
       /* Update the pair test list                 */
16     TestPairs.insert(NewPairs);
17     TreatedB[B_t] = true;
18 return iPairs
```

– Starting from an initial pair of elements intersecting inside Ω_C, $\{A_1, B_1\}$ (Fig. 4a), find all the intersections involving B_1 by recursively testing the elements around A_1 (Fig. 4b), taking care of marking which elements were tested already.
– While doing this, test the new elements A_i for intersections with B_1's direct neighbors inside the region Ω_C (line 15 of Alg. 3). Save the positive cases in a list of element pairs to be tested (Fig. 4c). This step and the previous one correspond to the inner loop of Alg. 3, between lines 10 and 15.
– After exhausting all the intersections possibilities involving B_1, mark it as "already tested".
– Repeat the algorithm with the next element pairs to be tested, ignoring the elements of $\mathcal{T}_B$ already treated.

Notice that, in this algorithm, the meshes do not have the same standing: $\mathcal{T}_B$ is used as a guide for the intersection search, while $\mathcal{T}_A$ is only explored locally at each step. In most practical cases, the number of elements $A_i \in \mathcal{T}_A$ tested in Alg. 3 inner loop is considerably smaller than $|\mathcal{T}_A|$, resulting in a linear average complexity $O(|\mathcal{T}_B|)$ [18]. The worst case scenario happens when all the elements of $\mathcal{T}_A$ and $\mathcal{T}_B$ intersect each other: in this extreme case we have $|\mathcal{T}_A|\,|\mathcal{T}_B|$ intersections, and so $O(|\mathcal{T}_A|\,|\mathcal{T}_B|)$ tests are done. From the memory point of view, no new data structures are created, and the algorithm uses $O(|\mathcal{T}_A| + |\mathcal{T}_B|)$ space.

The implementation of the advancing front algorithm presented in Alg. 3 differs mainly on the dependency on the region Ω_C. This is done to restrict the intersections to this region (line 13), and to stop the algorithm from exploring intersections outside of it (line 15). The second test might seem redundant at first, but it is needed because the advancing front algorithm is

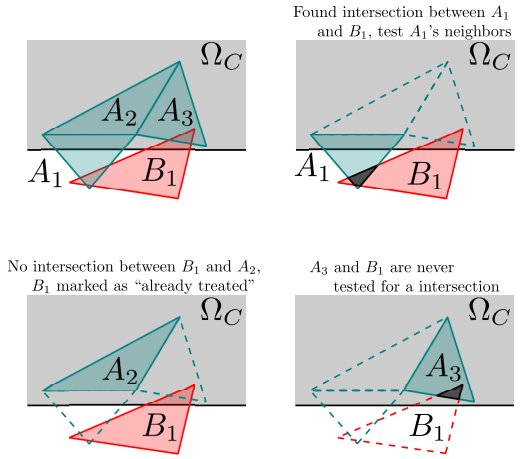


Figure 5: (color online) Behavior of the advancing front algorithm without the intersection region check done at line 15 of Alg. 3. In this case, the intersection between B_1 and A_3 is ignored.

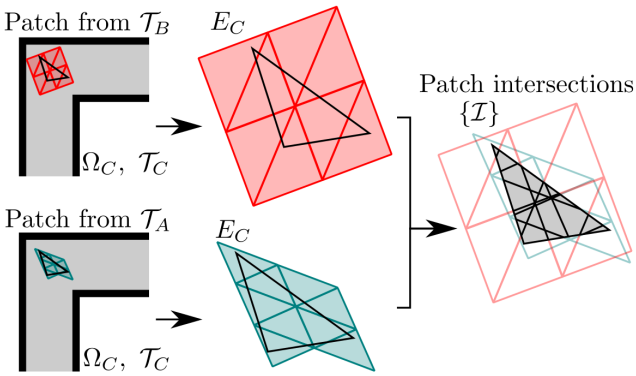


Figure 6: (color online) Graphical representation of the patch construction and intersection used in Alg. 4, and described in Alg. 5.

very sensitive on the direction over which the so-called front is advancing. Without it, is entirely possible that algorithm will add to the test list $TestPairs$ a pair of elements (A_1, B_1) that intersect *outside* the coupling region, and such that none of A_1's neighbors intersect B_1 (see Fig. 5). In this case, B_1 will be marked as "already treated" after testing with A_2, and the intersection with A_3 will not be found. This can be somewhat mitigated by expanding the test list at line 14 with k-nearest neighbors, but this slows down the algorithm, while not guaranteeing that this problem will always be avoided.

Algorithm 4: `ParallelInterSearch`: Parallelized intersection search. The inputs are the meshes $\mathcal{T}_A$, $\mathcal{T}_B$ and $\mathcal{T}_C$. The latter mesh is used to partition the search over the processors. The function `InterSearch` can be any serial search function, adapted to save only intersections inside a a given convex region (such as Alg. 3). The output is a table $iPairs$, containing the intersecting pairs.

Input:
Meshes $\mathcal{T}_A$, $\mathcal{T}_B$ and $\mathcal{T}_C$

Output:
Intersection pairs table $iPairs$

```
1  Partition the mesh T_C ;
2  oneach processor p do
3      Local partition of T_C: T_C^p;
4      Local intersection pairs: iPairs^p;
5      foreach element E_C ∈ T_C^p do
6          Overlapping patches: P_A, P_B;
           /* Build the patches                    */
7          BuildPatch (T_A, E_C, P_A);
8          BuildPatch (T_B, E_C, P_B);
           /* Search the intersections             */
9          InterSearch (P_A, P_B, E_C, iPairs^p);
       /* Gather intersections lists               */
10     Gather(iPairs^p, iPairs);
11 return iPairs
```

5.2 Parallel mesh intersection search

Let us describe now the parallel intersection search algorithm, Alg. 4. As before, for simplicity we suppose that the geometrical information of all elements of $\mathcal{T}_A$ and $\mathcal{T}_B$ are accessible by all processors. The main idea behind this algorithm is to distribute the intersection search over the processors by dividing the meshes $\mathcal{T}_A$ and $\mathcal{T}_B$ into overlapping patches $\mathcal{P}_A$ and $\mathcal{P}_B$. These patches are built in such a way that each overlapping patch pair $(\mathcal{P}_A, \mathcal{P}_B)$ covers exactly one element E_C of the mesh $\mathcal{T}_C$ (see Fig. 6). The patch construction process itself can be done using a simplified advancing front algorithm (Alg. 5), which is essentially the inner loop of 3, using the element E_C as the guide and $\mathcal{T}_i$ as the probed mesh. The intersection search is then reduced to an embarrassingly parallel problem, composed by $|\mathcal{T}_C|$ completely independent problems of finding the intersections between $(\mathcal{P}_A, \mathcal{P}_B)$ inside E_C. The only step in Alg. 4 which needs communications between the processors is the partitioning of $\mathcal{T}_C$ - which is used to distribute the workload over the processors.

Algorithm 5: `BuildPatch`: patch construction algorithm based on ref. [18]. The inputs are the element E_C which the patch will cover and the mesh $\mathcal{T}_i$ from which it will be constructed. The list $TestElem$ is a list of elements of $\mathcal{T}_i$ to be tested for intersections with E_C. The output is the patch $\mathcal{P}_i$.

Input:
Mesh $\mathcal{T}_i$, element E_C
Output:
Patch $\mathcal{P}_i$

Elements from $\mathcal{T}_i$ to test: $TestElem$;
Elements from $\mathcal{T}_i$ already treated: $TreatedElem$;
Search first intersection with E_C: $E_{init} \in \mathcal{T}_i$;
$\mathcal{P}_i.insert(E_{init})$;
$TreatedElem.insert(E_{init})$;
`/* Add` E_{init}`'s neighbors to test list */`
`UpdtTest(`$E_{init}, TestElem, TreatedElem$`);`
while $TestElem$ *is not empty* **do**
  $E_t = TestElem.pop()$;
  `/* Test for intersection, and update` $\mathcal{P}_i$ `if positive */`
  `UpdtPatch(`$E_t, E_C, \mathcal{P}_i$`);`
  `/* Add` E_t`'s neighbors to test list */`
  `UpdtTest(`$E_t, TestElem, TreatedElem$`);`
return $\mathcal{P}_i$

5.3 Scaling of the parallel mesh intersection algorithm

We will present now the implementation and scalability of our parallel intersection search and construction algorithm. In general, numerical geometry algorithms are susceptible to numerical imprecision and rounding errors, resulting into incorrect results, and this includes the intersection search and construction algorithms. The Computational Geometry Algorithms Library (CGAL) [30] avoids this problem by implementing its geometrical entities and algorithms using exact number types. We also chose it due to its Nef polyhedrons module. The Nef polyhedrons are defined by Boolean operations on a series of half-planes, and, by consequence, the intersection between two Nef polyhedrons can be constructed by applying an "AND" operation between them. This choice allows us to easily construct the intersection of an element triplet (E_A, E_B, E_C), regardless of the geometry of the elements chosen.

The usage of exact geometry algorithms comes at the cost of more expensive operations, though, and as such a proper optimization and balancing of the code is needed. Exact geometrical tests, such as verifying if two elements intersect, are considerably cheaper than exact constructions of new geometrical entities, such as the construction of their intersection. Due to this, when using exact operations, an efficient intersection search algorithm is considerably cheaper numerically than the intersection construction itself. Table 1 illustrates this with the CPU time taken by our search algorithm and by the intersection construction, for meshes of the domains presented in Fig. 7. Due to this difference, we will study the scaling of the search algorithms and of the intersection construction separately. Also, we will present a mesh repartitioning based on the number of intersections that allows us to properly distribute the workload of the intersection construction. All simulations presented here were done on the FUSION cluster, using Intel Xeon E5-2670v3 @ 2.30 GHz.

Intersection search:	154 s
Intersection build:	46351 s

Table 1: CPU time of the intersection search and construction steps, for the meshes described in Fig. 7 and on Table 2.

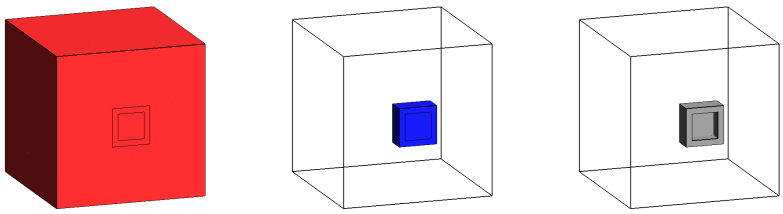
Mesh	Elements	Intersections
$\mathcal{T}_1$	$1,753,486$	—
$\mathcal{T}_2$	$438,426$	—
$\mathcal{T}_C$	$2,734$	—
$\mathcal{T}_I$	$6,662,531$	$1,715,343$

Table 2: Number of elements of the meshes from Fig. 7, and for the resulting intersection mesh. For the latter, the number of intersections is also shown.

Intersection search scaling: Strong scaling is the property of an algorithm to solve the same problem n times faster when using n times as many processors. We tested the strong scaling of the intersection search algorithm Alg. 4 using meshes of the domains presented in Fig. 7. The two system meshes, $\mathcal{T}_1$ and $\mathcal{T}_2$, have $\sim 1.75 \cdot 10^6$ and $\sim 4.38 \cdot 10^5$ elements, respectively, and the coupling mesh $\mathcal{I}_C$ has $2.7 \cdot 10^3$ elements. The resulting intersection mesh (not pictured) has $6.66 \cdot 10^6$ elements, which is considerably larger than the other three meshes used to generate it.

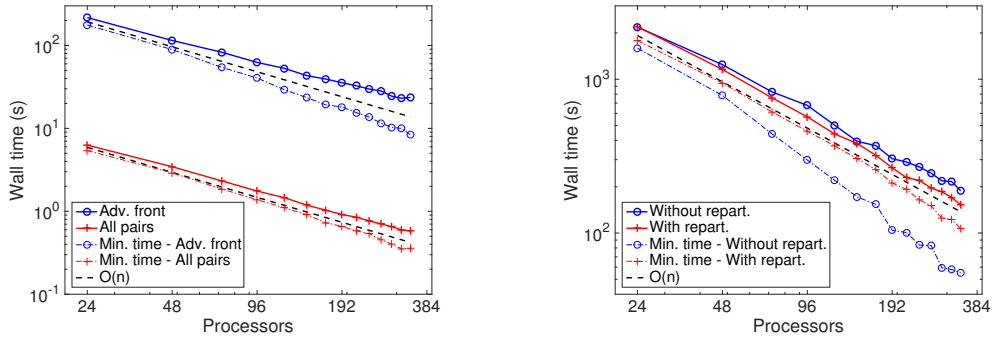
We tested the algorithm using two different search methods for the intersections between the patches $\mathcal{P}_A$ and $\mathcal{P}_B$: with the "naïve", direct pair search method (testing all the the element pairs for intersections), and with the advancing front method, described in Alg. 3.

Figs. 8a presents the wall time vs. number of CPUs for these methods (respectively, the red lines with the crosses, and the blue lines with circles) and the dashed



(a) (b) (c)

Figure 7: (color online) Domains of the meshes used for the intersection search and construction scaling: (a) macroscopic and (b) microscopic domains, and (c) the coupling region. The black lines indicate their geometrical position compared to one another. The number of elements of the corresponding meshes are shown in Table 2.



(a) (b)

Figure 8: (color online) Intersection algorithm strong scaling, wall time vs. number of processors. (a) Scaling of intersection search, for different methods used to find the intersections between two patches $\mathcal{P}_A$ and $\mathcal{P}_B$: test all element pairs and advancing front method (b) Scaling of intersection construction, with and without the repartitioning step. The wall times are indicated by the thick lines with symbols. The thin, dashed lines with symbols, marked with "Min. time" in the legends, indicate the time taken by the fastest processor in each simulation.

black lines show the ideal, $O(n)$ complexity scaling. The wall time represents, effectively, the time taken by the slowest processor, and the difference between it and the time taken by the fastest processor in each simulation (represented by the thin lines in these figures) gives to us an idea of how the workload is distributed in each simulation - and hence of the implementation's efficiency.

The small differences between the wall times and the fastest processor times when compared to the $O(n)$ complexity curve indicate that the algorithm is scalable. This reflects the fact that the processors do not communicate during the parallelized loop in Alg. 4, as we discussed in sub-section 5.2. These curves diverge from the $O(n)$ line, though, as the number of processors increase. This is due to an non-ideal distribution of the search workload, which is defined by $\mathcal{I}_C$'s partitioning. Here, we used libMesh's default partitioning weights, which associates to each element a weight equal to its number of nodes. This choice of weights has no information about the intersections associated to element $E_C \in \mathcal{I}_C$, and this leads to an uneven workload distribution as the number of processors (and hence partitions) increases. This diverge is small enough, though, that it poses no problem given the speed of the intersection searches.

Notice that the advancing front algorithm is more than an order of magnitude slower than the direct pair search method. This happens because the advancing front algorithm needs to construct an intersection to stop it from exploring intersections outside of the coupling region (as was discussed in sub-section 5.1), which increases the algorithm's cost.

Intersection construction scaling: The blue curves with circle markers in Fig. 8b present the wall time of the intersection construction as a function of the number of processors, for the same meshes as above, and using libMesh's default partitioning weights. This graph presents the same behavior as Figs. 8a, but since the intersection construction takes more time than the

	Wall time	Fastest proc. time
Without repart. (s)	188	55
With repart. (s)	152	107

Table 3: Wall and fastest processor times for the intersection construction. The latter is least time taken by a processor during the simulation. Done with 336 processors.

search, the uneven workload distribution becomes more pronounced.

Differently from the intersection search, though, we know at this step how many intersections are associated to each element $E_C \in \mathcal{I}_C$. The time taken to process each element E_C is directly proportional to this quantity, and hence using it as the weights of the partitioning results in a more balanced construction algorithm. This is illustrated by the red curves with plus markers in Fig. 8b. While there is still some unbalancing for a large number of processors when using this new choice of weights, it is considerably smaller than before. This results in an algorithm that is $\sim 20\%$ faster with the repartition (see Table 3, with the intersection constructions times for 336 processors.). More specifically, the wall time for the intersection construction with repartition is 2195 s on 24 processors and 152 s on 336 processors, while it is respectively 2182 s and 188 s without repartition. The speedup is therefore $14.4/14 > 1$ with repartition and $11.6/14 \approx 0.83$ without repartition. Finally, it can be concluded that the strategy proposed for the intersection search and construction makes the coupling algorithm strongly scalable on 24 to 336 processors.

6 Numerical results

Let us now present our numerical results. We will study the properties and scaling of the FETI / Arlequin solver, based on the projected Conjugated Gradient (CG) algorithm (Alg. 1), and present an application case composed by a macroscopic, homogeneous model coupled with a microscopic heterogeneous model representing a polycrystalline material. To implement the Arlequin and FETI algorithms, we used the libMesh library [24]. All simulations presented here were done on the FUSION cluster, using Intel Xeon E5-2670v3 @ 2.30 GHz. The libraries that we developed to do these simulations (including the ones used for the intersection search scaling in the previous section) are freely available at the "Code Arlequin" GitHub repository [1].

Weak scalability is the property of an algorithm to require a constant wall time to solve a problem with n times as many degrees of freedom over n times as many processors. We will start by studying the weak scaling of this algorithm for different implementations - with and without the reorthogonalization operation, and for different convergence tests. After this, we will analyze these implementations in more details, by focusing on the behavior of a system where their differences are obvious. Finally, we will study the computational impact of the different preconditioner choices.

For all these cases, we applied the FETI solver on a coupled traction test, presented in Fig. 9. Both systems are modeled with the same 3D linear elasticity model, with a Young's modulus $E = 200$ GPa and a shear modulus $\mu = 80$ GPa. Their domains Ω_1 and Ω_2 are discretized by the meshes $\mathcal{T}_1$ and $\mathcal{T}_2$. Both models are coupled at the region Ω_{12}^c. A force density $F = 100$ kPa is applied on the face Γ_F of the domain Ω_2, on the $\hat{\mathbf{x}}$ direction, while the face Γ_0 of the domain Ω_1 is clamped. Since the domain Ω_2 has no Dirichlet boundary conditions, we will use its rigid body modes for the projected CG algorithm. In all cases, the mesh $\mathcal{T}_C$ of the coupling region has $|\mathcal{T}_C| \sim 3.5 \cdot 10^3$ elements.

6.1 Weak scaling

Usually, the weak scaling of iterative methods such as the CG algorithm is done by analyzing the evolution of the average time per iteration for different system sizes. This is done because increasing the system size increases the number of iterations of these solvers, and thus the wall time does not offer a suitable measurement of the scalability. In our case, though, , the duration of each iteration depends on the choice of solvers for the system models - which are assumed here to have weak scalings of their own. Thus, this quantity is not a suitable measurement for the weak scaling of solely the projected CG algorithm implemented here.

Considering this, we chose to use the number of iterations of the coupled solver until the convergence, n_{conv} for the weak scaling. This quantity should stay stable if one of the model's meshes increases in number of elements while the mediator space stays the same. In the context of our traction test, this corresponds to increasing the number of elements of the mesh $\mathcal{T}_2$ while keeping the mesh $\mathcal{T}_1$ unchanged (and thus the mediator space). We will present in the following sub-sections the numerical costs of the coupling phase, compared to the model solver costs.

To do this scaling test, we used a mesh $\mathcal{T}_1$ with $|\mathcal{T}_1| \sim 5 \cdot 10^5$ elements, and we varied the number of elements of $\mathcal{T}_2$ from $7.2 \cdot 10^3$ to $\sim 4.9 \cdot 10^5$, roughly doubling $|\mathcal{T}_2|$ for each case. The number of processors was

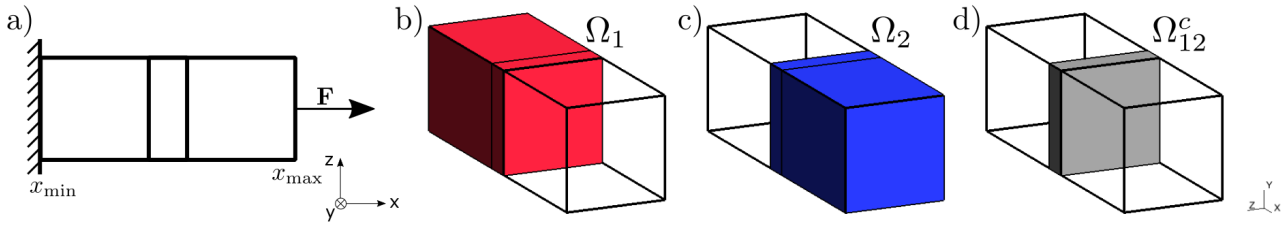
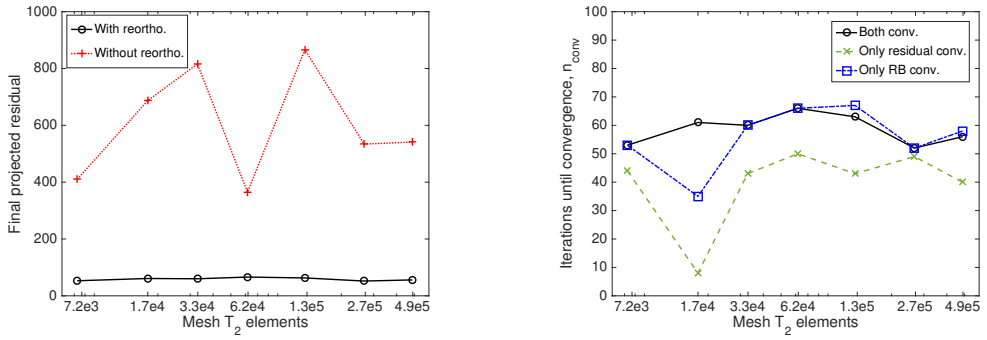


Figure 9: (color online) Domains of the meshes used for the traction coupling tests: (a) schematic representation of the test, (b) domain Ω_1 of the first model, (c) domain Ω_2 of the second model, and (d) the coupling region, Ω_{12}^c. The black lines on Figs (b), (c) and (d) indicate their geometrical position compared to one another.



(a) (b)

Figure 10: (color online) Weak scaling for different implementations of the projected CG algorithm: number of iterations until convergence vs. numbers of elements for the mesh $\mathcal{T}_2$: (a) with and without the reorthogonalization step, and (b) for different convergence tests (only checking the residual convergence, only checking the $\mathbf{R}_2\boldsymbol{\alpha}$ correction convergence (RB check), and doing both checks.). In (a), both checks were used. In all cases, $|\mathcal{T}_1| \sim 5{\cdot}10^5$ elements.

also increased accordingly, increasing from 6 to 384 processors for the smallest and the largest values of $|\mathcal{T}_2|$. The simulations were done using the inverse coupling matrix preconditioner, $\mathbf{M}_{PC} = (\mathbf{C}_m)^{-1}$. In all cases, we used the relative $\rho^{(k)}$ convergence check $\rho(k) < \epsilon_\rho \rho(0)$, with $\epsilon_\rho = 10^{-8}$, and, when applicable, the $\mathbf{R}_2\boldsymbol{\alpha}$ convergence check from Eq. (26), with $\epsilon_{RB} = 10^{-6}$.

Fig. 10 presents n_{conv} for each simulation. Fig. 10a shows the difference due to the reorthogonalization (or lack of thereof), while Fig. 10b shows the differences due to the different convergence check tests - namely, with only the $\rho^{(k)}$ check, only the $\mathbf{R}_2\boldsymbol{\alpha}$ check, and with both checks. In Fig. 10a, both checks were used.

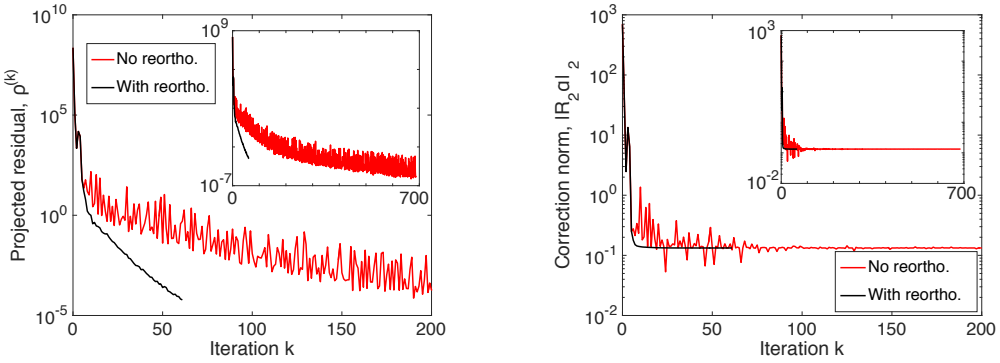
For the case without the reorthogonalization, the algorithm does not present any form of weak scaling. The number of iterations varies wildly between 365 and 866, with no correlation with the number of elements from the mesh $\mathcal{T}_2$. The reorthogonalization reduces and stabilizes considerably the number of iterations, which now vary between 52 and 66, which corresponds to a better weak scaling.

Concerning the convergence checks, using either only the $\rho^{(k)}$ check or the $\mathbf{R}_2\boldsymbol{\alpha}$ check results in series of simulations with worse scaling than using both checks at the same time (notably for $|\mathcal{T}_2| \sim 1.7 \cdot 10^4$). While the norm of $\mathbf{R}_2\boldsymbol{\alpha}$ is a stronger convergence parameter than $\rho^{(k)}$ (reflected by the larger number of iterations for the former), it does not supersede it, since the $\mathbf{R}_2\boldsymbol{\alpha}$ check curve does not coincide with the curve with both checks. Considering this, we decided to keep both checks.

6.2 Reorthogonalization and convergence check

In this section, we will present in more details the results of the scaling for $|\mathcal{T}_2| \sim 1.7 \cdot 10^4$. This series of simulations presents the largest difference of number of iterations until convergence between the different algorithm implementations (Fig. 10), and hence is better adapted to illustrate the differences between them.

Fig. 11 shows the evolution of the projected residual $\rho(k)$ (following Eq. (25)) and of the rigid body (RB) modes correction norm $|\mathbf{R}_2\boldsymbol{\alpha}|$, as a function of the it-



(a) (b)

Figure 11: (color online) Effect of the descent reorthogonalization on the evolution of the final (a) residual norm, $\rho^{(k)}$, and (b) rigid body correction norm $|\mathbf{R}_2\boldsymbol{\alpha}|_2$. The algorithm converged after 687 iterations without the reorthogonalization (red curves), and after 61 with it and both convergence checks (black curves). The main figures show the curves up to 200 iterations, while the insets show the full curves. The simulations with only the residual or the RB convergence check follow the black curves, stopping after 8 and 35 iterations, respectively. Number of elements in each mesh: $|\mathcal{T}_1| \sim 5 \cdot 10^5$ elements, $|\mathcal{T}_2| \sim 1.7 \cdot 10^4$ elements.

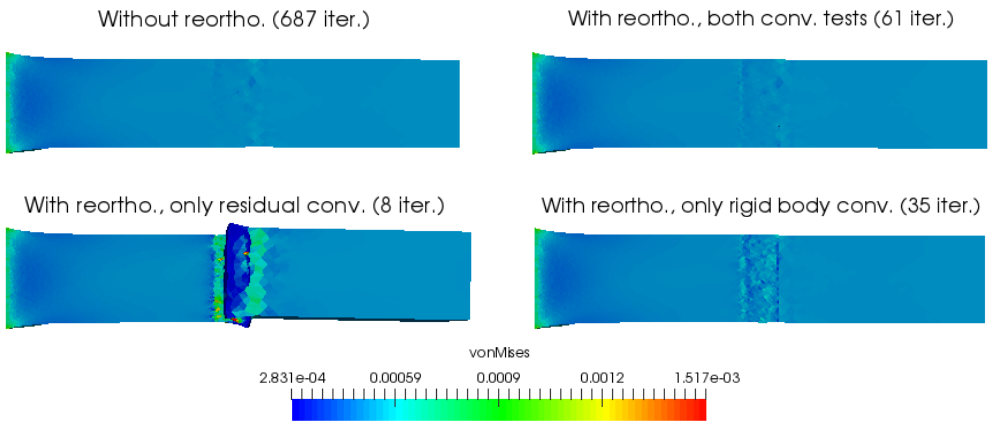
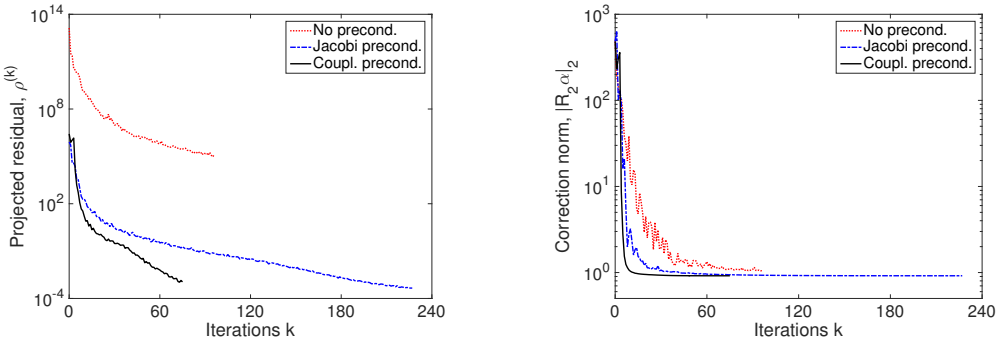


Figure 12: (color online) Final, deformed traction test mesh for different projected CG algorithm implementations. The colors represent the von Mises stress. Number of elements in each mesh: $|\mathcal{T}_1| \sim 5 \cdot 10^5$ elements, $|\mathcal{T}_2| \sim 1.7 \cdot 10^4$ elements. The displacements were scaled by a factor of 10^3.

eration. The black and the red curves show the results for reorthogonalized and non-reorthogonalized simulations, with both the convergence checks. The simulations with only one of the convergence checks follow the black line, ending after 8 iterations for the residual check, and after 35 for the RB check. Fig. 12 shows the final, deformed system for each implementation case, with deformations scaled by a factor of 10^3.

We can see from Fig. 11 that the non-reorthogonalized implementation differs after only a few iterations from the other implementations, indicating that the orthogonality of the descents is lost shortly after the start of the simulation. After this point, we have recurrent peaks for the residual, with amplitude variations of a factor of $\sim 10^2$. The RB correction present smaller oscillations.

The reorthogonalization operation not only reduces considerably the number of iterations until convergence from 687 to 61 (for the same convergence checks), but it also eliminates these peaks from all but the first few iterations. As we can see from Fig.12, the converged solution is similar in both cases. Using only one of the convergence checks yields results that are not properly converged. All these factors indicate that the reorthogonalization is needed to guarantee the robustness and efficiency of the projected CG algorithm, and that both convergence checks are needed to guarantee a correct result.



(a) (b)

Figure 13: (color online) Effects of the preconditioner choice on (a) the residual norm, $\rho^{(k)}$, and (b) the L^2 norm of the rigid body correction term, $|\mathbf{R}_2\boldsymbol{\alpha}|_2$. Number of elements in each mesh: $|\mathcal{T}_1| \sim |\mathcal{T}_2| \sim 1 \cdot 10^6$ elements. Preconditioned simulations were done with $\epsilon_\rho = 10^{-8}$ and $\epsilon_{RB} = 10^{-6}$, and the non-preconditioned one with $\epsilon_\rho = 10^{-8}$ and $\epsilon_{RB} = 10^{-4}$.

$\mathbf{M}_{PC}$	n_{conv}	Total time (s)	Time / iter. (s)	Model 1 (s)	Model 2 (s)	Precond. (s)	Proj. (s)
$\mathbf{C}_m^{-1}$	79	136.639	1.730	39.769	32.452	64.199	0.219
				29.105%	23.750%	46.985%	0.160%
$\text{diag}\,(\mathbf{C}_m)^{-1}$	227	169.337	0.746	85.422	83.370	0.002	0.543
				50.445%	49.233%	0.001%	0.321%

Table 4: Wall time costs for each component of the projected CG algorithm (Alg. 1), for different choices of preconditioner matrices, $\mathbf{M}_{PC}$. "Model 1" and model "Model 2" correspond to calls to the model's solvers, "Precond." to the preconditioner operations, and "Proj." to the projection operations, $\boldsymbol{\Pi}_R$.

6.3 Preconditioner effects

Before we pass to an applied case, let us discuss the effects of different preconditioners $\mathbf{M}_{PC}$ on the projected CG algorithm convergence, and also analyze the (possibly) overhead time added to the algorithm for each choice. We tested two different cases: $\mathbf{M}_{PC} = \mathbf{C}_m^{-1}$, based on Eq. (27); $\mathbf{M}_{PC} = \text{diag}\,(\mathbf{C}_m)^{-1}$, which is a Jacobi-like approximation of the previous case. To use the first preconditioner efficiently, we must define a third equation system, using $\mathbf{C}_m$ as the system matrix, and solve it. The Jacobi-like preconditioner, on the other hand, only adds a vector term-by-term multiplication.

To do these tests, we chose to use a coupled system similar to the previous traction test, with both system meshes $\mathcal{T}_1$ and $\mathcal{T}_1$ containing $\sim 1{\cdot}10^6$ elements. The simulations were done on 384 processors. The convergence parameters chosen were $\epsilon_\rho = 10^{-8}$ and $\epsilon_{RB} = 10^{-6}$.

Figs. 13 show the residual $\rho^{(k)}$ and the RB correction $|\mathbf{R}_2\boldsymbol{\alpha}|_2$ for all simulation cases. For reference, these figures also show the results with no preconditioning and $\epsilon_\rho = 10^{-8}$ and $\epsilon_{RB} = 10^{-4}$ (this case did not converge for $\epsilon_{RB} = 10^{-6}$). As one should expect, the preconditioned simulations present much lower values of $\rho^{(k)}$, and the $\mathbf{R}_2\boldsymbol{\alpha}$ correction norm goes down rapidly to reasonable values for the preconditioned simulations, while it stays slightly above the correct values during the non-preconditioned simulation.

This can be explained by the imprecisions of the solution approximation $\boldsymbol{\Phi}^{(k)}$: Eq. (23) has been deduced for the real solution $\boldsymbol{\Phi}$, and it will only result on valid rigid body modes corrections if $\boldsymbol{\Phi}^{(k)}$ is a good approximation of it - which is false during the first iterations of the solver in all cases. Since the non-preconditioned simulation takes longer to yield a reasonable approximation of $\boldsymbol{\Phi}$, it presents the largest interval with incorrect $\mathbf{R}_2\boldsymbol{\alpha}$ corrections - and hence the larger oscillations and slower convergence.

Table 4 presents the timing and iteration statistics for each of the preconditioned simulations, including the timing of their main components: the system solvers, the preconditioner, and the projection operators. In both cases, the projection operation amount to less than 0.4% of the wall time. This indicates that our implementation of the projection operator $\boldsymbol{\Pi}_R$ does not incur a performance bottleneck, and that the algorithm optimization should focus on reducing the cost of the models' solvers and the number of iterations of the coupled solver. We should stress here that the costs of

model solvers and of the $\mathbf{M}_{PC} = \mathbf{C}_m^{-1}$ preconditioner depend strongly on the number of processors chosen for the models and the coupling. Note that the matrix $\mathbf{C}_m$ is inverted using the generalized minimal residual method (GMRES), PETSc's default parallel iterative linear equation system solver.

Regarding the preconditioner costs, the $\mathbf{M}_{PC} = \mathbf{C}_m^{-1}$ case is considerably more expensive than the Jacobi-like $\mathbf{M}_{PC} = \mathrm{diag}\,(\mathbf{C}_m)^{-1}$ case (and even more expensive than the solves of the models). This is compensated by the reduced number of iterations, and hence reduce number of calls to both models' solvers, resulting into a simulation $\sim 19.3\%$ faster in the former case. The high cost of the $\mathbf{M}_{PC} = \mathbf{C}_m^{-1}$ preconditioner is mainly due to the relatively low sparsity of this matrix when compared to the other matrices, which reduces the efficiency of the linear solver associated to it. If such costs becomes too high, the results for the Jacobi-like preconditioner $\mathbf{M}_{PC} = \mathrm{diag}\,(\mathbf{C}_m)^{-1}$ show that it is a viable numerical alternative.

7 Application: crack test

For an application test, we considered the coupled crack test, with the meshes shown in Fig. 14. The macroscopic system is described by an homogeneous 3D linear elasticity model, with a Young's modulus $E = 200$ GPa and a shear modulus $\mu = 80$ GPa. The corresponding domain Ω_1 is represented by the mesh $\mathcal{T}_1$ (Fig. 14a), with $|\mathcal{T}_1| \sim 3.4 \cdot 10^4$ elements. Its leftmost side is clamped, while displacements are imposed on right side holes, on the $\hat{\mathbf{z}}$ and $-\hat{\mathbf{z}}$ directions. The microscopic system is described by an heterogeneous 3D anisotropic linear elasticity model, with physical parameters $c_{11} = 198$ GPa, $c_{12} = 125$ GPa, $c_{44} = 122$ GPa. Its domain Ω_2 (Fig. 14b) is divided into 250 crystals, each with its own random anisotropy direction following an uniform direction distribution on a sphere. The corresponding mesh has $|\mathcal{T}_2| \sim 7.2 \cdot 10^6$ elements. This model is located at the crack junction of the macroscopic mesh $\mathcal{T}_1$, and both meshes are coupled by the region marked in Fig. 14c, with a coupling mesh with $|\mathcal{T}_C| \sim 1.1 \cdot 10^3$ elements. Overall, the domains were meshed in such a way that the mesh $\mathcal{T}_1$'s element size is $\sim 10\times$ smaller near the crack junction than on the outer region, while the mesh $\mathcal{T}_2$'s element size is $\sim 10\times$ smaller than the smallest element size of $\mathcal{T}_1$. For the preconditioner, we used $\mathbf{M}_{PC} = \mathbf{C}_m^{-1}$.

The simulation was run on 96 processors, and converged after 63 iterations, with the convergence parameters, we used $\epsilon_\rho = 2 \cdot 10^{-4}$ and $\epsilon_{RB} = 10^{-4}$. Fig. 15 present the resulting deformed meshes, as well as the von Mises stress for both the models. The displacement conditions on the macroscopic mesh $\mathcal{T}_1$ result on a coherent deformation of the mesh $\mathcal{T}_2$, which presents a maximum von Mises stress near the tip of the crack. We can also visualize the grain anisotropy effects near this region, with different grains presenting different deformation responses.

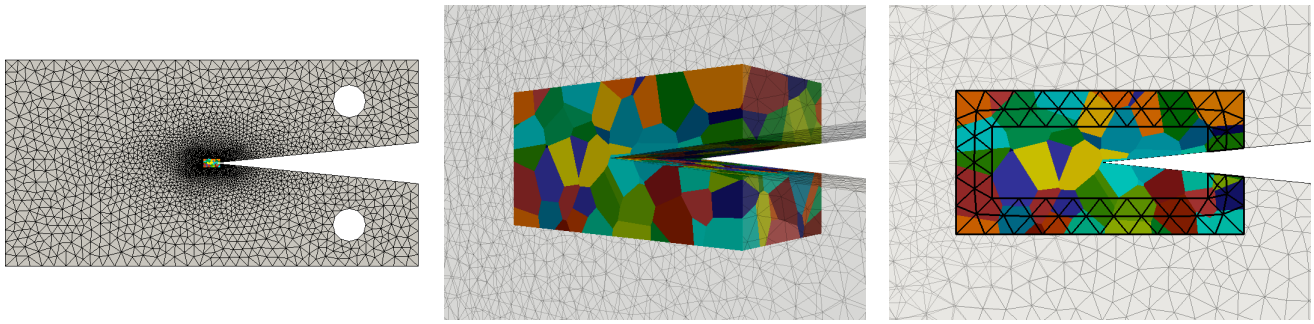
Operation	Time (s)
System initializations	8.09
Coupling matrix assemble	27.16
I/O	52.48
Arlequin / FETI solver	398.61
Setup	2.53
Solve model 1	52.99
Solve model 2	340.14
Precond.	2.05
Other	0.90
Total	486.34

Table 5: Wall times for the coupled crack test simulation, with convergence parameters $\epsilon_\rho = 2 \cdot 10^{-4}$ and $\epsilon_{RB} = 10^{-4}$. "System initialization" consists on the equation systems' initialization. The "Arlequin / FETI solver" time is subdivided into its main components, including the setup, the models' solves, preconditioning operations, and other (projections and vector manipulations)

8 Conclusion

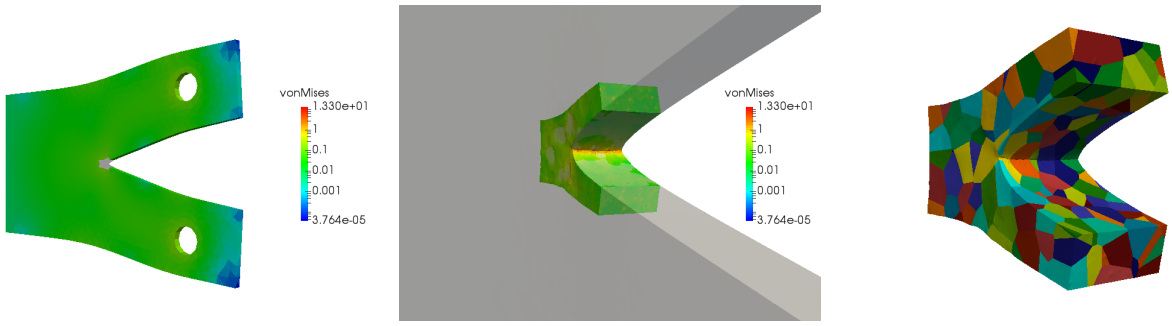
We presented in this article a fully scalable framework to assemble and solve a coupled problem using the Arlequin method. The two main components of this framework are a new parallel algorithm to find the intersections between the meshes of the coupled models and a more detailed and scalable implementation of a FETI / Arlequin solver. While the total CPU time costs of the intersection construction algorithm are considerable (mainly due to the choice of the exact geometry libraries), this algorithm follows a strong scaling, which guarantees a fast assemble of the coupling operations if a suitable number of processors is chosen.

Concerning the FETI / Arlequin solver, we presented a parallel implementation which has a weak scaling and for which most of time cost is due to the calls of the models' solvers. This translates into an algorithm with a small numerical overhead for the coupling operations, and which will not worsen the scalability of the models' solvers. This is possible due to our detailed study of the optimizations needed to avoid the numerical bottlenecks of the algorithm and a careful choice of convergence criteria.



(a) (b) (c)

Figure 14: (color online) Meshes used for the crack test: (a) macroscopic homogeneous model mesh, $\mathcal{T}_1$, with the microscopic, polycrystalline and anisotropic model mesh inset, $\mathcal{T}_2$. (b) zoom on the microscopic mesh. (c) coupling region mesh, $\mathcal{T}_C$ (black, thick lines). Number of elements in each mesh: $|\mathcal{T}_1| \sim 3.4 \cdot 10^4$ elements, $|\mathcal{T}_2| \sim 1.1 \cdot 10^6$ elements.



(a) (b) (c)

Figure 15: (color online) Results of the crack test: (a) deformed macroscopic model mesh, $\mathcal{T}_1$, (b) deformed microscopic model mesh, $\mathcal{T}_2$ (inserted into the macroscopic mesh), and (c) deformed microscopic model mesh with grains. In (a) and (b), the colors represent the von Mises stress (log scale) of the macro and micro models, respectively. Results obtained after 63 iterations, with $\epsilon_\rho = 2 \cdot 10^{-4}$ and $\epsilon_{RB} = 10^{-4}$, and using the coupling matrix preconditioner.

The usage of the FETI method also allows this algorithm to be easily adapted to multi-physics problems with very different coupled model pairs, such as linear / non-linear models, atomistic / continuum and deterministic / stochastic model pairs. Furthermore, the formulation of the coupling method is completely independent from the models, referencing them only through the calls to their solvers. This allows the usage of each models' own solver and code implementation without changing the coupling algorithm. The deterministic / stochastic model pair application is especially important for the numerical homogenization of random materials [10,8,9]. Another interesting extensions of this algorithm consist on generalizing it for a parallel solver with time coupled models, or for reduced-order solvers, such as the PGD [28]. Our future works will focus on these applications.

Finally, the work presented here can be extended to other physical couplings using the formulation from Eq. (1), such as the ones presented in refs. [31,6,21], and the intersection search algorithm can be adapted to other numerical methods, such as the Nitsche method [26].

Acknowledgements This work benefited from French state funding managed by the National Research Agency under project number ANR-14-CE07-0007 CouESt. The simulations were done using the CentraleSupelec-ENS Paris-Saclay's computing mesocenter, FUSION. We would like to thank its support team for the help provided during the simulations.

References

1. Code Arlequin. https://github.com/cottereau/CArl
2. Alart, P., Iceta, D., Dureisseix, D.: A nonlinear Domain Decomposition formulation with application to granular dynamics. Computer Methods in Applied Mechanics and Engineering **205-208**, 59–67 (2012). DOI 10.1016/j.cma.

2011.04.024. URL https://hal.archives-ouvertes.fr/hal-00597519
3. Ben Dhia, H.: Problèmes mécaniques multi-échelles: la méthode Arlequin (written in French). Comptes Rendus de l'Académie des Sciences - Series IIB - Mechanics-Physics-Astronomy **326**(12), 899–904 (1998). DOI 10.1016/S1251-8069(99)80046-5. URL http://www.sciencedirect.com/science/article/pii/S1251806999800465
4. Ben Dhia, H., Elkhodja, N., Roux, F.X.: Multimodeling of multi-alterated structures in the Arlequin framework. European Journal of Computational Mechanics **17**(5-7), 969–980 (2008). DOI 10.3166/remn.17.969-980. URL http://www.tandfonline.com/doi/abs/10.3166/remn.17.969-980
5. Ben Dhia, H., Rateau, G.: The Arlequin method as a flexible engineering design tool. Int. J. Numer. Meth. Engng. **62**(11), 1442–1462 (2005). DOI 10.1002/nme.1229. URL http://onlinelibrary.wiley.com/doi/10.1002/nme.1229/abstract
6. Chamoin, L., Prudhomme, S., Ben Dhia, H., Oden, T.: Ghost forces and spurious effects in atomic-to-continuum coupling methods by the Arlequin approach. Int. J. Numer. Meth. Engng. **83**(8-9), 1081–1113 (2010). DOI 10.1002/nme.2879. URL http://onlinelibrary.wiley.com/doi/10.1002/nme.2879/abstract
7. Chessa, J., Belytschko, T.: An extended finite element method for two-phase fluids: flow simulation and modeling. Journal of Applied Mechanics **70**(1), 10–17 (2003). DOI 10.1115/1.1526599
8. Cottereau, R.: Numerical strategy for unbiased homogenization of random materials. Int. J. Numer. Meth. Engng **95**(1), 71–90 (2013). DOI 10.1002/nme.4502. URL http://onlinelibrary.wiley.com/doi/10.1002/nme.4502/abstract
9. Cottereau, R.: A Stochastic-deterministic Coupling Method for Multiscale Problems. Application to Numerical Homogenization of Random Materials. Procedia IUTAM **6**, 35–43 (2013). DOI 10.1016/j.piutam.2013.01.004. URL http://www.sciencedirect.com/science/article/pii/S2210983813000059
10. Cottereau, R., Clouteau, D., Ben Dhia, H., Zaccardi, C.: A stochastic-deterministic coupling method for continuum mechanics. Computer Methods in Applied Mechanics and Engineering **200**(47–48), 3280–3288 (2011). DOI 10.1016/j.cma.2011.07.010. URL http://www.sciencedirect.com/science/article/pii/S0045782511002519
11. Díez, P., Cottereau, R., Zlotnik, S.: A stable extended FEM formulation for multi-phase problems enforcing the accuracy of the fluxes through Lagrange multipliers. International Journal for Numerical Methods in Engineering **96**(5), 303–322 (2013). DOI 10.1002/nme.4554
12. Dolean, V., Jolivet, P., Nataf, F.: An Introduction to Domain Decomposition Methods. Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics (2015). URL http://epubs.siam.org/doi/book/10.1137/1.9781611974065
13. E, W., Engquist, B.: The Heterognous Multiscale Methods. Communications in Mathematical Sciences **1**(1), 87–132 (2003). DOI 10.4310/CMS.2003.v1.n1.a8. URL http://www.intlpress.com/site/pub/pages/journals/items/cms/content/vols/0001/0001/a008/
14. Elkhodja, N.: Approches de structures complexes dans des cadres adaptés de la méthode Arlequin (PhD thesis, written in French)
15. Embar, A., Dolbow, J., Harari, I.: Imposing Dirichlet boundary conditions with Nitsche's method and spline-based finite elements. International Journal for Numerical Methods in Engineering **83**(7), 877–898 (2010). DOI 10.1002/nme.2863
16. Farhat, C., Roux, F.X.: A method of finite element tearing and interconnecting and its parallel solution algorithm. International Journal for Numerical Methods in Engineering **32**(6), 1205–1227 (1991). DOI 10.1002/nme.1620320604. URL http://doi.wiley.com/10.1002/nme.1620320604
17. Fernández-Méndez, S., Huerta, A.: Imposing essential boundary conditions in mesh-free methods. Computer Methods in Applied Mechanics and Engineering **193**(12-14), 1257–1275 (2004). DOI 10.1016/j.cma.2003.12.019
18. Gander, M.J., Japhet, C.: An Algorithm for Non-Matching Grid Projections with Linear Complexity. In: M. Bercovier, M.J. Gander, R. Kornhuber, O. Widlund (eds.) Domain Decomposition Methods in Science and Engineering XVIII, no. 70 in Lecture Notes in Computational Science and Engineering, pp. 185–192. Springer Berlin Heidelberg (2009). URL http://link.springer.com/chapter/10.1007/978-3-642-02677-5_19. DOI: 10.1007/978-3-642-02677-5_19
19. Ghanem, A., Torkhani, M., Mahjoubi, N., Baranger, T., Combescure, A.: Arlequin framework for multi-model, multi-time scale and heterogeneous time integrators for structural transient dynamics. Computer Methods in Applied Mechanics and Engineering **254**, 292–308 (2013). DOI 10.1016/j.cma.2012.08.019. URL http://linkinghub.elsevier.com/retrieve/pii/S004578251200271X
20. Gill, P.E., Murray, W., Institute of Mathematics and Its Applications, National Physical Laboratory (Great Britain) (eds.): Numerical methods for constrained optimization. Academic Press, London ; New York (1974)
21. Hu, H., Belouettar, S., Potier-Ferry, M., Daya, E.M., Makradi, A.: Multi-scale nonlinear modelling of sandwich structures using the Arlequin method. Composite Structures **92**(2), 515–522 (2010). DOI 10.1016/j.compstruct.2009.08.051. URL http://www.sciencedirect.com/science/article/pii/S0263822309003249
22. Hughes, T.J.R.: Multiscale phenomena: Green's functions, the Dirichlet-to-Neumann formulation, subgrid scale models, bubbles and the origins of stabilized methods. Computer Methods in Applied Mechanics and Engineering **127**(1–4), 387–401 (1995). DOI 10.1016/0045-7825(95)00844-9. URL http://www.sciencedirect.com/science/article/pii/0045782595008449
23. Juntunen, M., Stenberg, R.: Nitsche's mthod for general boundary conditions. Mathematics of Computation **78**, 1353–1374 (2009). DOI 10.1090/S0025-5718-08-02183-2
24. Kirk, B.S., Peterson, J.W., Stogner, R.H., Carey, G.F.: `libMesh`: A C++ Library for Parallel Adaptive Mesh Refinement/Coarsening Simulations. Engineering with Computers **22**(3–4), 237–254 (2006). http://dx.doi.org/10.1007/s00366-006-0049-3
25. Marchais, J., Rey, C., Chamoin, L.: Representation of localized phenomena in dynamics using multi-scale coupling. In: B.H.V. Topping (ed.) Proceedings of the Eleventh International Conference on Computational Structures Technology, 252, pp. 1–12 (2012)
26. Massing, A., Larson, M.G., Logg, A.: Efficient implementation of finite element methods on non-matching and overlapping meshes in 3d. arXiv:1210.7076 [math]

(2012). URL http://arxiv.org/abs/1210.7076. ArXiv: 1210.7076
27. Moës, N., Dolbow, J., Belytschko, T.: A finite element method for crack growth without remeshing. Int. J. Numer. Meth. Engng. **46**(1), 131–150 (1999). DOI 10.1002/(SICI)1097-0207(19990910)46:1⟨131::AID-NME726⟩3.0.CO;2-J. URL http://onlinelibrary.wiley.com/doi/10.1002/(SICI)1097-0207(19990910)46:1<131::AID-NME726>3.0.CO;2-J/abstract
28. Néron, D., Ben Dhia, H., Cottereau, R.: A decoupled strategy to solve reduced-order multimodel problems in the PGD and Arlequin frameworks. Comput Mech pp. 1–13 (2016). DOI 10.1007/s00466-015-1236-0. URL http://link.springer.com/article/10.1007/s00466-015-1236-0
29. Samet, H.: Hierarchical spatial data structures. In: A.P. Buchmann, O. Günther, T.R. Smith, Y.F. Wang (eds.) Design and Implementation of Large Spatial Databases, no. 409 in Lecture Notes in Computer Science, pp. 191–212. Springer Berlin Heidelberg (1989). URL http://link.springer.com/chapter/10.1007/3-540-52208-5_28. DOI: 10.1007/3-540-52208-5_28
30. The CGAL Project: CGAL User and Reference Manual, 4.7 edn. CGAL Editorial Board (2015). URL http://doc.cgal.org/4.7/Manual/packages.html
31. Wellmann, C., Wriggers, P.: A two-scale model of granular materials. Computer Methods in Applied Mechanics and Engineering **205–208**, 46–58 (2012). DOI 10.1016/j.cma.2010.12.023. URL http://www.sciencedirect.com/science/article/pii/S0045782510003798
32. Xiao, S., Belytschko, T.: A bridging domain method for coupling continua with molecular dynamics. Computer Methods in Applied Mechanics and Engineering **193**(17-20), 1645–1669 (2004). DOI 10.1016/j.cma.2003.12.053. URL http://linkinghub.elsevier.com/retrieve/pii/S004578250400026X
33. Zomorodian, A., Edelsbrunner, H.: Fast software for box intersections. pp. 129–138. ACM Press (2000). DOI 10.1145/336154.336192. URL http://portal.acm.org/citation.cfm?doid=336154.336192