Two-level substructuring and parallel mesh generation for domain decomposition methods

Yannis El Gharbi, Augustin Parret-Fréaud, Christophe Bovet, Pierre Gosselet

Two-level substructuring and parallel mesh generation for domain decomposition methods

Y. El Gharbi[1,2], A. Parret-Fréaud[2], C. Bovet[3], P. Gosselet[1,4]

[1] *Université Paris-Saclay, ENS Paris-Saclay, CNRS, LMT - Laboratoire de Mécanique et Technologie, 91190, Gif-sur-Yvette, France, yannis.el-gharbi@ens-paris-saclay.fr*
[2] *Safran Tech, Modeling & Simulation, Rue des Jeunes Bois, Châteaufort, 78114 Magny-Les-Hameaux, France, augustin.parret-freaud@safrangroup.com*
[3] *Onera — The French Aerospace Lab F-92322 Châtillon, France, christophe.bovet@onera.fr*
[4] *LaMcube, Univ. Lille / CNRS / Centrale Lille, pierre.gosselet@univ-lille.fr*

Abstract

This paper presents a new parallel mesh generation method leading to subdomains of shape well-suited to Schur based domain decomposition methods such as the FETI and BDD solvers. Starting from a coarse mesh, subdomains meshes are created in parallel through hierarchical mesh refinement and morphing techniques. The proposed methodology aims at limiting the occurrence of known pathological situations (jagged interfaces, misplaced heterogeneity with respect to the interfaces, ...) that penalize the convergence of the solver. Furthermore, it enables to distribute and parallelize the mesh generation step in the early phases of the whole analysis. Besides its good behavior towards convergence, the mesh generation is thus distributed. The method is assessed, on several academical and industrial test cases, for both its parallel efficiency when creating the mesh and its capability to generate decomposition resulting in less FETI iterations.

Keywords: Domain decomposition methods, parallel mesh generation, FETI

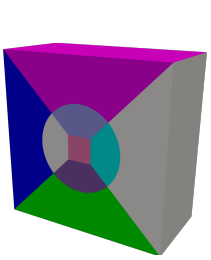
1. Introduction

Non overlapping domain decomposition methods provide a favorable framework to define iterative solvers, such as FETI[1] [14] or BDD[2] [31], adapted to the massively parallel architecture of recent supercomputers, for the resolution of linear and nonlinear structural mechanics problems. They have been successfully applied to solve several challenging mechanical heterogenous problems such as a composite wing model [38] or strongly heterogeneous mockup reentry vehicle of a rocket ship [4]. An overview of these Schur complement based methods is available for instance in [19].

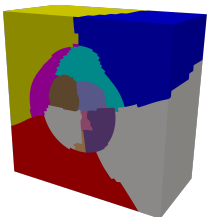
[1] Finite Elements Tearing and Interconnecting
[2] Balanced Domain Decomposition

It has been observed, and mathematically analyzed, that these methods suffer from a lack of robustness in certain cases (typically jagged interfaces [32], bad subdomain aspect ratios, strong heterogeneity misplaced with respect to the interface [35], incompressibility). These problematic cases are inevitably encountered when using automatic graph partitioners to generate the subdomain decomposition of common industrial structures.

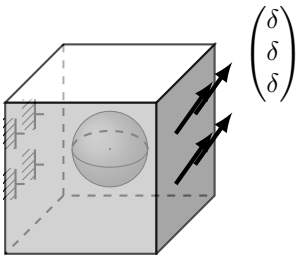
This phenomenon can be illustrated with the simple example of a stiff inclusion embedded in a soft matrix, as shown in Figure 1. On Figure 1d, one can observe that the regularity of the interface and its position with respect to the heterogeneity have a very strong influence on the convergence rate of the original FETI solver (equipped with state-of-the art preconditioner and using conjugate gradient).



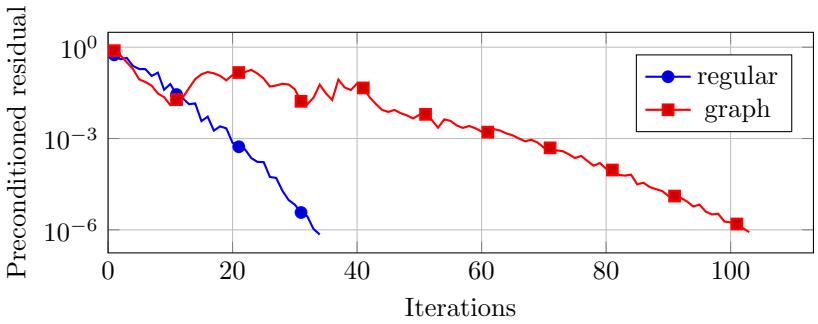
(a) Cross section view of the decomposition with *regular* interfaces and homogeneous domains.



(b) Cross section view of the graph-based decomposition obtained with an automatic graph partitioner.



(c) Boundary conditions, one face is clamped and a longitudinal and transverse displacement is imposed on the opposite face.



(d) Evolution of the preconditioner-norm of the residual over FETI iterations.

Figure 1: Influence of the shape of the subdomains on the convergence of the FETI method. The test case is a spherical rigid inclusion in a soft matrix (linear elastic behavior, the ratio of Young modulus is 10^5, the Poisson coefficient is $\nu = 0.3$, 1,2 million degrees of freedom, 13 subdomains).

This lack of robustness occurs when the classical preconditioner (corresponding to a weighted sum of the inverses of subdomains operator) wrongly estimates

some diffused features [20]. To compensate for this lack of relevance of classical preconditioner, several solutions have been proposed. In the FETI-GenEO[3] method [41], the eigenvectors of the preconditioned operator which cause bad condition number (corresponding to the largest eigenvalues), are detected *a priori* and removed from the resolution by an augmentation of the Krylov solver. Another method, called Block-FETI [20], exploits the additive structure of the problem to generate a family of right-hand sides to be solved by a block conjugate gradient. A more general framework of this approach is proposed in [21]. A last family of methods, called multi-preconditioning, uses the additive structure of the preconditioner to generate a full search subspace at each iteration instead of a single vector (see Simultaneous-FETI [20] and its adaptive version AMPFETI[4] [6]). All these fixes have shown their efficiency, but they induce higher memory consumption (Block FETI, AMPFETI) or extra computations (GenEO) which may become significant on large scale problems.

On the one hand, if it seems possible to tune the weighting of graph edges in order to optimize the position of interfaces with respect to the heterogeneity and to produce homogeneous subdomains [22], it appears impossible to guarantee a regular geometric aspect of these interfaces because the graph partitioner software does not take into account any geometrical features. A BSP-tree (such as a KD-tree) could generate spatially localized subdomains but it could not insure regular interfaces because the underlying mesh is generally not structured. On the other hand, the traditional approach of calculations has an important drawback in that both the mesh generation and the decomposition stage are dominated by sequential steps which constitute a significant part of the total computation time once the resolution is made in parallel. Specifically, the conformal mesh generation of complex 3D structures from geometrical data is a process which can become very expensive and would benefit from a parallel implementation.

Regarding this last point, several coarse grained parallel remeshing approaches have been proposed so far. In [10], repartitioning and remeshing steps are performed by iterations in order to satisfy a given target metric on the whole domain, but the resulting method produces irregular interfaces because of the repartitioning at each iteration. A master-slave approach can also be employed (see [25, 30]), where a master process is in charge of meshing the interface before distributing the volumetric filling on the subdomain's slave processes. This method doesn't appear to be scalable because of the growth of the size of the interface with the number of subdomains. Recent other methods [8, 45] are based on the generation of an initial coarse distributed mesh which will undergo local remeshing techniques. This last class of methods, where the one presented in this paper lies in, are expected to be more efficient for large scale mesh generation.

[3] Generalized Eigenproblems in the Overlaps
[4] Adaptive MultiPreconditioned FETI

The classical computational chain involving a domain decomposition solver is summed up in the upper branch of Figure 2, where the data pre-processing consists in the two following steps:

1. mesh generation on the global structure, possibly of large size, with a sequential mesher (or its fine-grained parallel counterpart [29]);
2. substructuring of the associated connectivity graph with a partitioning software (*e.g.* Metis [27], Scotch [37], Chaco [23]...) or a Binary Space Partitioning tree of the position of elements (KD-tree [17], RP-tree, MM-tree...).

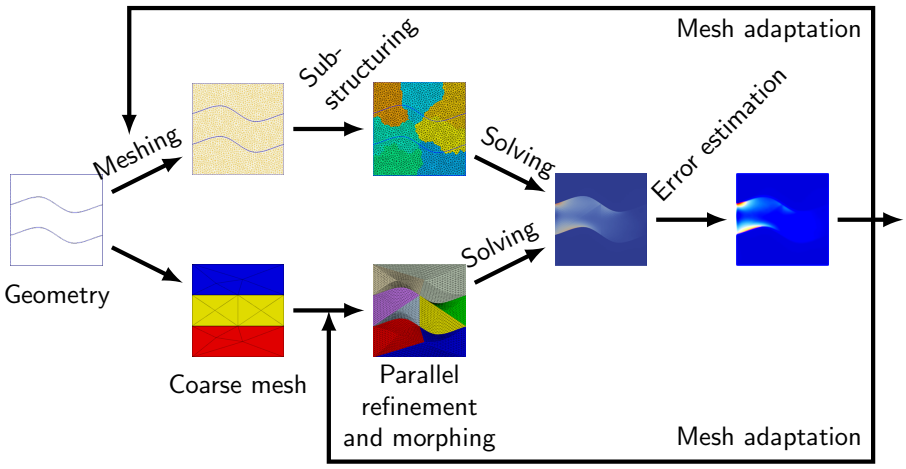


Figure 2: Domain decomposition computations approach: - Upper branch: classical method - Lower branch: developed two-level method.

The method proposed in this paper is described in the lower branch of Figure 2. Bear in mind that even if we focus on making parallel the meshing step, our objective is also to ease the solution step by limiting the need to use the advanced strategies described previously. Our objective is thus to produce, in a way as parallel as possible, a substructured mesh with regular interfaces, possibly adapted to structural heterogeneity, in order to improve the condition number of the condensed problem. To do so, the method — which appears to be completely original in solid computational mechanics — is schematically based on a reversal of the meshing and substructuring steps, while possibly using hierarchical discretization.

First, from a discrete geometry (*i.e.* CAD or discrete CAD), a coarse mesh is generated. This coarse mesh does not accurately represent the underlying geometry but simply allows to define a well-proportioned partitioning. The subdomains are then defined as unions of coarse elements and distributed on the computing cores along with the underlying subdomain CAD information. Thereafter, the fine mesh of the structure is carried out in parallel while keeping

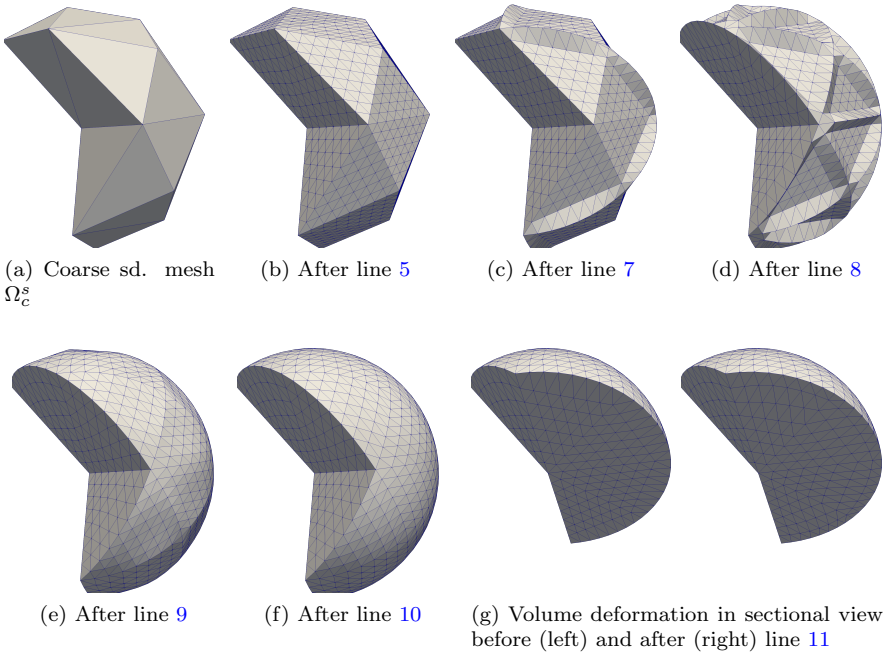
the interface compatibility. Finally, the application of mesh deformation techniques allows to ensure that the fine mesh respects the underlying geometry.

The article is organized as follows: Section 2 gives a detailed description of the method. Numerical results on heterogeneous and homogeneous cases are provided in Section 3. The developments presented in this section make use of the Python interface of the Z-set[5] finite element suite [3, 16]. Finally, Section 4 concludes the paper.

2. Two-level substructuring method

The methodology proposed in this article is described in Algorithm 1, where each step is detailed in the following subsections.

2.1. Overall methodology



(a) Coarse sd. mesh Ω_c^s (b) After line 5 (c) After line 7 (d) After line 8

(e) After line 9 (f) After line 10 (g) Volume deformation in sectional view before (left) and after (right) line 11

Figure 3: Illustration of the parallel steps of the Algorithm 1 on a subdomain of a sphere input geometry, where three levels of hierarchical refinement are used to generate the target mesh.

The starting point of the method is the fine description of all the geometrical entities required for the setting of the mechanical problem to be solved:

[5] https://www.zset-software.com

Algorithm 1: Two-level substructuring method (illustrated by Figure 3)

Data:
– $((\Omega_\phi^d)_d, \mathcal{F}_\phi, \mathcal{E}_\phi)$ geometry of the computational structure;
Input:
– target number of subdomains (sd.) N_s;
– target number of hierarchical refinements N_r;
begin
three-dimensional coarse meshing of the geometry: $(\mathcal{T}_c, \mathcal{F}_c, \mathcal{E}_c)$;
partitioning of the coarse mesh into its N_ϕ different materials $\left(\mathcal{T}_c^d, \mathcal{F}_c^d, \mathcal{E}_c^d\right)_{d\in[\![1,N_\phi]\!]}$;
for $d \in [\![1, N_\phi]\!]$ **do**
partitioning of each material into a number N_s^d of subdomains depending on N_s such as $\sum_{d=1}^{N_\phi} N_s^d = N_s$;
end
Data:
– partitioned coarse mesh $(\mathcal{T}_c^s, \mathcal{F}_c^s, \mathcal{E}_c^s)_{s\in[\![1,N_s]\!]}$;
for $s \in [\![1, N_s]\!]$ **parallel do** `// parallel stage`
restriction of Ω_ϕ in the vicinity of the subdomain's faces $\mathcal{F}_c^s$;
subdomain's meshes N_r hierarchical refinements $(\mathcal{T}_r^s, \mathcal{F}_r^s, \mathcal{E}_r^s)$;
projection of $(\mathcal{F}_r^s)$ on $(\mathcal{F}_\phi^s)$:
– projection of physical edges $(\mathcal{E}_{r,\phi}^s)$ with a curvilinear abscissa based method;
– projection of remaining edges $(\mathcal{E}_{r,\partial}^s)$ along their normals;
– interpolation of $(\mathcal{F}_r^s)$ surfaces;
– projection of $(\mathcal{F}_{r,\phi}^s)$ surfaces on $(\mathcal{F}_\phi^s)$ along their normals;
deformation of $(\mathcal{T}_r^s, \mathcal{F}_r^s, \mathcal{E}_r^s)$ according to projection of $(\mathcal{F}_r^s)$ (line 6 to 10);
end
end
Output: sub-structured refined mesh $(\mathcal{T}_r^s, \mathcal{F}_r^s, \mathcal{E}_r^s)_s$ respecting the physical geometry $((\Omega_\phi^d)_d, \mathcal{F}_\phi, \mathcal{E}_\phi)$;

- Description of the physical domains $(\Omega_\phi^d)_d$. These are connected components which in general are associated with constant material coefficients. The number of physical domains is N_ϕ.
- Set of physical faces $\mathcal{F}_\phi$. It is assumed that the boundaries of the physical subdomains can be decomposed into piecewise regular faces. Some of them are the support of mechanical boundary conditions such as a prescribed load (pressure), displacement or interface properties (contact, friction).
- Set of physical edges $\mathcal{E}_\phi$. These are the intersection of the boundaries of

physical faces. They play an important role in the representation of the geometry.

This information can be provided in various formats: CAD (Computer Aided Design) formats like BREP [34], STEP [24] or software-specific geometry file (like GMSH's `geo` file [18]) or simply —as it is done in this paper— a fine mesh of the skin (like STL format [1]) with adapted tagging of faces and edges.

2.2. Coarse meshing

The first step of the method, illustrated in Figure 4b and occuring at line 1 of Algorithm 1, consists in the generation of a coarse mesh of the physical geometry $((\Omega_\phi^d)_d, \mathcal{F}_\phi, \mathcal{E}_\phi)$. This results in the discrete sets $(\mathcal{T}_c, \mathcal{F}_c, \mathcal{E}_c)$ with the following properties:

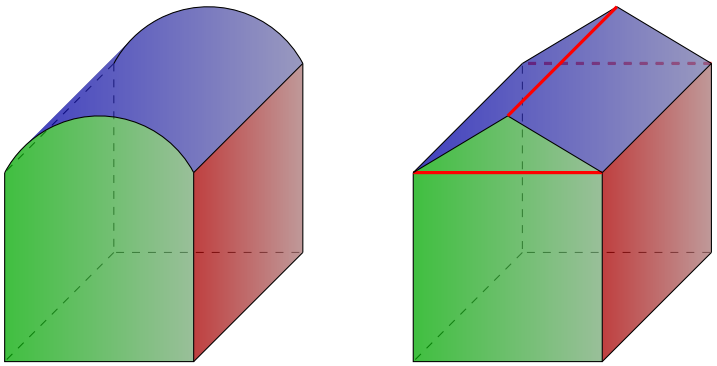
- Coarse elements forming the set $\mathcal{T}_c$ are simple polyhedrons (in general tetrahedrons, sometimes hexahedrons). Each coarse element is associated with one physical subdomain.
- Coarse faces, gathered in the set $\mathcal{F}_c$, are the boundaries of the coarse elements. Let $\mathcal{F}_{c,\phi}$ denote the subset of coarse faces associated with physical faces, note that in that case their vertexes belong to the associated physical face.
- Coarse edges, gathered in the set $\mathcal{E}_c$, are the boundaries of the coarse faces. Let $\mathcal{E}_{c,\phi}$ denote the subset of coarse edges associated with physical edges, and let $\mathcal{E}_{c,\mathcal{F}}$ denote the subset of coarse edges associated with physical faces but not with physical edges (edges connecting two coarse faces associated with the same physical surface). Note that in the latter two cases the vertexes also belong to the associated physical edge or face.

Note that at this point, there is absolutely no need to represent the true geometry correctly. Only the topological and physical information needs to be stored along with by the coarse mesh.

2.3. Sequential decomposition

In the step descibed by the lines 2 and 3 of Algorithm 1, the user should provide the total number of computational subdomains N_s the geometry should be split in. This number needs not be limited by the actual computational resource since over-allocation of subdomains on processing units may be employed. Thus, we assume that $\#\mathcal{T}_c \geqslant N_s \geqslant N_\phi$.

Each coarse representation of a physical subdomain is split into computational subdomains $(\Omega_c^s)_s$ in proportion with the objective to have, as much as possible, a sensibly identical number of coarse elements in each computational domain. The decomposition is achieved by a graph partitioning software (*e.g.* Scotch [37], Metis [27], Chaco [23]). For the sake of simplicity, the parallel versions of these softwares (such as PT-Scotch [9] or Parmetis [26]) are not used here as their sequential counterparts remains efficient on the coarse mesh even



(a) Physical geometry $((\Omega_\phi^d)_d, \mathcal{F}_\phi, \mathcal{E}_\phi)$

(b) A resulting coarse mesh $(\mathcal{T}_c, \mathcal{F}_c, \mathcal{E}_c)$. Red edges corresponds to non-physical edges lying in $\mathcal{E}_{c,\mathcal{F}}$, other edges are in $\mathcal{E}_{c,\phi}$. Here, the only non-physical coarse face is the one shared by the hexahedron and the prism.

Figure 4: Definition of the different geometrical sets, each color represents a different physical surface $\mathcal{F}_{c,\phi}^f$.

wh en the number of subdomains becomes important. Please note that in the case of an heterogeneous structure made from materials of very different computational cost (such as a composite material with elastic fibers and viscoplastic matrix), weights can be added on the elements during the graph partitioning process to achieve a better load balancing.

Also, if trying to preserve the homogeneity of computational subdomains leads to too many small ones, one can consider recombining them under the quasi-monotonicity condition described in [36]. The underlying idea is to build computational subdomains as chains of connected small coarse elements with increasing stiffness (Young modulus). This construction allows to limit the degradation of the constant which controls the scalability of the method.

Remark 1 (Preservation of the normal). *One geometrical piece of information needs to be preserved before distributing the data on several processors. Non-physical coarse edges associated with physical faces* $\mathcal{E}_{c,\partial} = (\partial\mathcal{F}_{c,\phi}) \setminus \mathcal{E}_{c,\phi}$ *must have the capability to project themselves on the physical surface. Since they are not physical edges, these edges are on smooth parts of the surface where a slowly-varying normal vector can be defined. Classically, the normal vector is approximated by the average of the normal vectors of the two coarse physical faces united by the edge. In case the physical faces belonged to two different computational subdomains (see Figure 5), the building of the normal vector would require one small communication. In order to avoid that problem, the normal vectors for edges of* $\mathcal{E}_{c,\partial}$ *are computed before the decomposition (at least for edges*

which are on the interface between computational subdomains).

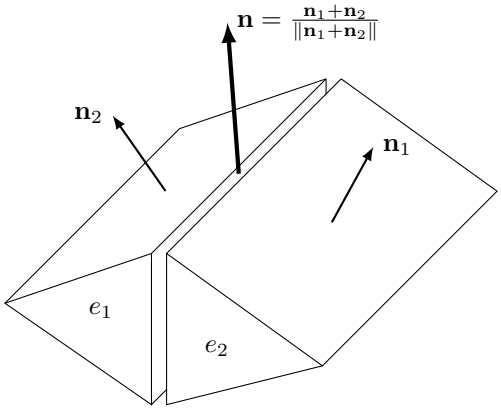


Figure 5: Computation of the normal of an edge shared between coarse elements in neighboring subdomains $e_1 \in \mathcal{T}_c^1$ and $e_2 \in \mathcal{T}_c^2$

Remark 2 (Cutting of the underlying geometry). *In order to reduce the size of the area on which each physical face $f \in \mathcal{F}_{c,\phi}$ and each non physical coarse edge $e \in \mathcal{E}_{c,\partial}$ has to be projected, and thus to reduce the numerical cost of the projection, it is advisable to crop the fine geometry around each of these entities (line 4 of Algorithm 1). This cropping is performed using the space delimited by the planes defined by the edges which are the border of the face (∂f) and their associated normal vector as defined in Remark 1.*

Starting from now, the data are distributed on the N_s computational subdomains. Each subdomain contains its coarse mesh and its associated fine geometry and physical tags. The remaining part of Algorithm 1 is done in parallel. Also, the connectivity between subdomains is known and a topological communicator can be created.

2.4. Local refinement

The coarse mesh was designed in order to distribute data, not for the actual computation. Thus, refinement is required at line 5 of Algorithm 1. Different strategies are possible:

Hierarchical refinement. Both h-refinement and p-refinement (with isoparametric elements) can be considered. In order to automatically enforce the conformity of the fine mesh between subdomains, the same refinement must be used on each subdomain. This is of course the simplest option. The quality of the resulting fine mesh may depend on the quality of the coarse mesh and on the number of successive call to h-refinement. One advantage of this approach is to generate topologically identical refined coarse elements, which are prone to computational acceleration like GPU-processing [28].

Topological meshing. We assume that a target metric size is available at each coarse vertex. Assuming that the same deterministic meshing algorithm is used on each subdomain, a possibility is that each subdomain first mesh its edges, then its faces and finally its volume.

Master-Slave approach. Using a coloring of the subdomain graph connectivity, another possibility is to have some master subdomains that mesh themselves first and then communicate their boundary to their neighbors. These slave subdomains should use the received interfaces as imposed nodes when their turn comes to mesh themselves. One possibility is to reuse the methodology developed in [5].

Mortar interfaces. A last possibility is to have all subdomain meshes generated in parallel and then use one communication to build mortar assembly operators on the interfaces [42, 13, 12]. With this solution, the mesh is not conforming at the interface between subdomains and the domain decomposition solver needs a special treatment.

Note that whatever the chosen strategy, only a very small amount of information must be communicated between neighbors. For the sake of simplicity, we chose h-refinement in the following applications, without exploiting the topological identity of the refined coarse elements for acceleration. The chosen h-refinement scheme is described on Figure 6 for four-noded thetrahedron and eight-noded hexahedron, where each edge is divided into two smaller edges.

Let Ω_r^s(see Figure 3b) denote the refined version of the coarse mesh Ω_c^s (see Figure 3a). All the sets defined on Ω_c^s are ported on Ω_r^s adding the new created nodes.

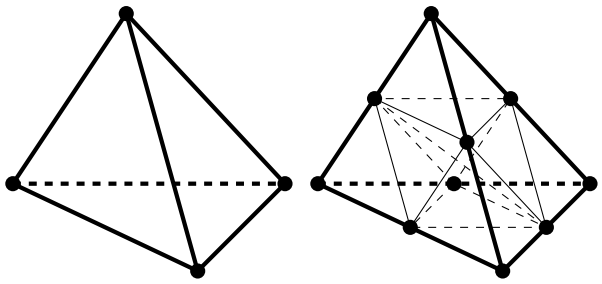
2.5. Mesh deformation

Since the mesh $(\Omega_r^s)_s$ is sufficiently refined, and is conforming at the interface between computational subdomains, it strongly disagrees with the exact geometry described by $(\cup_d \partial\Omega_\phi^d)$. For each subdomain Ω_r^s, the final mesh is obtained by a sequence of projections to the exact geometry which need no communication between subdomains. In that section, the superscript s referring to the subdomain can thus be omitted.

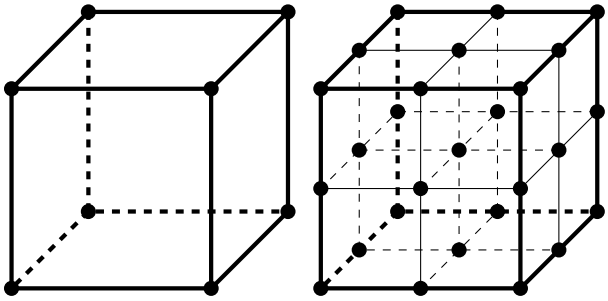
2.5.1. Projection of physical edges $\mathcal{E}_{r,\phi}$ on $\mathcal{E}_\phi$

This step is the one described at line 7 of Algorithm 1. Each edge $e_r \in \mathcal{E}_{r,\phi}$ is associated with one physical edge $e_\phi \in \mathcal{E}_\phi$ which must be correctly represented in the final mesh. Note that by construction, the vertexes at the end of these edges are already matching. In order to make the appropriate projection, a simple solution is to use a normalized curvilinear abscissa mapping between two edges.

Edges are considered as oriented curves to which we assign the same orientation (same starting vertex, same ending vertex). Let $(\mathbf{x}_{r,m})_{1\leqslant m\leqslant M}$ be the ordered sequence of the nodes of e_r. We also use a discretized description $(\mathbf{x}_{\phi,n})_{1\leqslant n\leqslant N}$ of e_ϕ.



(a) h-refinement of a tetrahedron



(b) h-refinement of a hexahedron

Figure 6: h-refinement schemes

Let ℓ denote the normalized curvilinear abscissa, it can be computed as:

$$\ell_{r,i} = \frac{\sum_{m=1}^{i} \|\mathbf{x}_{r,m+1} - \mathbf{x}_{r,m}\|}{\sum_{m=1}^{M-1} \|\mathbf{x}_{r,m+1} - \mathbf{x}_{r,m}\|}, \qquad \ell_{\phi,j} = \frac{\sum_{n=1}^{j} \|\mathbf{x}_{\phi,n+1} - \mathbf{x}_{\phi,n}\|}{\sum_{n=1}^{N-1} \|\mathbf{x}_{\phi,n+1} - \mathbf{x}_{\phi,n}\|},$$

where the norm is the classical Euclidean norm.

The nodes of e_r can be moved on e_ϕ at the corresponding curvilinear abscissa as shown in Figure 7: for the k^{th} node of e_r let K be the rank such that $\ell_{r,k} \in [\ell_{\phi,K}, \ell_{\phi,K+1}]$. Thus, the linear interpolation can be written as:

$$\mathbf{x}_{r,k}^{new} = \alpha \mathbf{x}_{\phi,K} + (1-\alpha)\mathbf{x}_{\phi,K+1}, \text{ with } \alpha = \frac{\ell_{r,k} - \ell_{\phi,K+1}}{\ell_{\phi,K} - \ell_{\phi,K+1}}.$$

The result of this operation is illustrated in Figure 3c.

2.5.2. Projection of the remaining boundary edges

The nodes of the remaining boundary edges $\mathcal{E}_{r,\partial} = (\mathcal{E}_r \cap (\partial\mathcal{F}_{c,\phi})) \setminus \mathcal{E}_{r,\phi}$ need to be projected on the physical boundary (line 8 of Algorithm 1). Thanks to Remark 1, they are all granted a normal vector or the capability to compute it without communication.

More precisely, the normal direction $\mathbf{d}_e$ was chosen to be the sum of the nor-

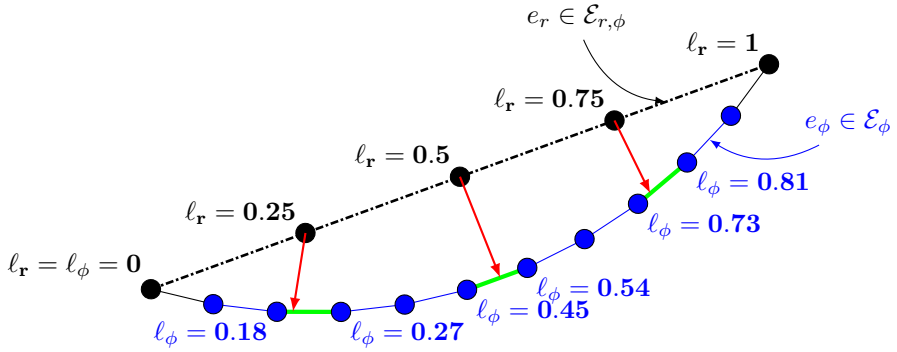


Figure 7: Projection of the edge of $e_r \in \mathcal{E}_{r,\phi}$ (in dashed black line) on the corresponding edge $e_\phi \in \mathcal{E}_\phi$ (in solid blue line), ℓ_r and ℓ_ϕ are the normalized curvilinear abscissa. The green lines are the segments where the condition $\ell_{r,k} \in \left[\ell_{\phi,K}, \ell_{\phi,K+1}\right]$ is satisfied.

mal vectors of the two adjacent coarse physical faces of the edge. For instance, if $e_r \in \mathcal{E}_{r,\partial}$, there exist $(f_{e,1}, f_{e,2}) \in \mathcal{F}^2_{c,\phi}$ such that $e_r = \partial f_{e,1} \cap \partial f_{e,2}$. Let $\mathbf{n}_1$ and $\mathbf{n}_2$ be the normal vectors associated with $f_{e,1}$ and $f_{e,2}$, then $\mathbf{d}_e = \frac{\mathbf{n}_1+\mathbf{n}_2}{\|\mathbf{n}_1+\mathbf{n}_2\|}$. As said earlier, if $f_{e,1}$ and $f_{e,2}$ do not belong to the same computational domain, then $\mathbf{d}_e$ has to be computed before the distribution of the data in order to avoid extra communication between domains.

In order to move the edges, a ray-triangle intersection algorithm [33] is used. This algorithm has the advantage to avoid the computation of the equation of the plane, which can result in significant memory savings. Moreover it seems to be the fastest existing method according to authors. The ray-triangle intersection algorithm is detailed in Appendix B. Note that this method can be vectorized for all triangles and all nodes. The result of this operation is shown in Figure 3d.

2.5.3. Interpolation of the surfaces

Next, we wish the refined faces to have non-distorted meshes and to match the physical surface. This is achieved in two steps. First, the faces are moved according to the displacement already obtained on their edges. This is done by a pointwise interpolation on each face $f_r \in \mathcal{F}_{r,\phi}$. Second, a projection of all surface vertices is performed along their normal.

The first step (the interpolation at line 9 of Algorithm 1) does not take into account the underlying topology nor the physical geometry, the only input is the displacement of all boundary edges in $\mathcal{E}_r \cap (\partial\mathcal{F}_{c,\phi})$. This interpolation is performed using a Radial Basis Function (RBF) as described in [11]. This choice is motivated by the simplicity of this meshless method and by the quality of the resulting deformed elements [39].

In this method, the interpolated displacement field $\mathbf{u}$ of the node at the position $\mathbf{x}$ can be expressed independently from others according to the relation:

$$\mathbf{u}(\mathbf{x}) = \sum_{j=1}^{n_b} \boldsymbol{\alpha}_j \psi\left(\left\|\mathbf{x} - \mathbf{x}_{b_j}\right\|\right) + p(\mathbf{x}), \tag{1}$$

where $(\mathbf{x}_{b_j})_{1 \leqslant j \leqslant n_b}$ are the edge nodes whose displacements $(\mathbf{u}_{b_j})_j$ are known, ψ is a radial basis function (to be chosen in the literature, see Appendix A), and p is a polynomial (a first degree polynomial allows describing rigid body motions of the interpolation, which is sufficient in our applications).

The coefficients $(\boldsymbol{\alpha}_j)$ of the RBF and $(\boldsymbol{\beta}_k)$ of the polynomial are determined by the following interpolation and compatibility conditions [7]:

$$\begin{aligned} &\mathbf{u}(\mathbf{x}_{b_j}) = \mathbf{u}_{b_j}, \quad 1 \leqslant j \leqslant n_b; \\ &\sum_{j=1}^{n_b} \boldsymbol{\alpha}_j q(\mathbf{x}_{b_j}) = 0, \quad \forall q \text{ polynomial of degree lower or equal than } p. \end{aligned} \tag{2}$$

This leads to the following linear system:

$$\begin{bmatrix} \mathbf{M} & \mathbf{P}_b \\ \mathbf{P}_b^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{bmatrix} = \begin{bmatrix} \mathbf{u}_b \\ 0 \end{bmatrix}, \tag{3}$$

where $\boldsymbol{\alpha}$ and $\mathbf{u}_b$ are the $n_b \times 3$ matrices of unknown components $(\alpha_{j,k})$ and known displacements $(u_{b_{j,k}})$; $\mathbf{M}$ is a symmetric positive conditionally definite matrix of size $n_b \times n_b$, such that $M_{ij} = \psi\left(\left\|\mathbf{x}_{b_i} - \mathbf{x}_{b_j}\right\|\right)$, $\mathbf{P}_b$ is a rectangular matrix of size $n_b \times 4$, whose i^{th} row writes $P_{b_i} = \begin{bmatrix} 1 & x_{b_{i1}} & x_{b_{i2}} & x_{b_{i3}} \end{bmatrix}$ (the $x_{b_{ik}}$ are the components of $\mathbf{x}_{b_i}$) and $\boldsymbol{\beta}$ is a 4×3 matrix of unknown polynomial coefficients $(\beta_{i,k})$. Note that the interpolation does not couple the physical dimensions so that the same matrix applies simultaneously to each of the 3 directions in Equation (3), only the right-hand side changes with each direction. So that, the direct inversion could be cheap with multiple right-hand side techniques.

Several choices for the RBF functions are described along with their main properties in Appendix A. We chose ψ as the CP $\mathcal{C}^2$ function which appears to be the best option for mesh deformation purposes according to [11] and to our experiments. We chose the support radius $r = 5 \max(\|\mathbf{u}_b\|)$, which saves the elements quality of the resulting meshes. This radius is chosen to be much larger than the deformation in order to propagate the displacement to the whole surface, see Remark 3 for an illustration. The result of such an operation is illustrated in Figure 3e.

2.5.4. Projection of the surfaces

Once the nodes of a face have been interpolated according to the displacement of the edges, they can be projected on the physical surface (line 10 of Algorithm 1). Once again, the ray-triangle intersection algorithm described in Appendix B is used. The $\mathbf{d}_f$ direction of the projection is chosen for each surface to be the normal of the coarse faces. The result of the operation is illustrated in Figure 3f.

2.6. Volume deformation

Finally, at line 11 of Algorithm 1, a volume interpolation is completed using the RBF interpolation technique described in Section 2.5.3. Here, the given displacements $\mathbf{u}_b$ are the displacements of the boundary nodes of the subdomain.

Remark 3 (Importance of the RBF radius). *In cases where the deformed surface is concave such as in Figure 8, the surface projection step may lead to some invalid elements due to reversal when the boundary deformation is larger than the size of the elements. This can be corrected during the volume deformation step by choosing a sufficiently large RBF radius, so that the deformation is spread in a large enough area including several layers of elements. That is the reason why we choose a radius* $r = 5\max(\|\mathbf{u}_b\|)$*. Note that no mesh quality control process is used in this paper but if the resulting mesh still exhibits a poor element quality, a mesh smoothing procedure can be performed subsequently [2].*

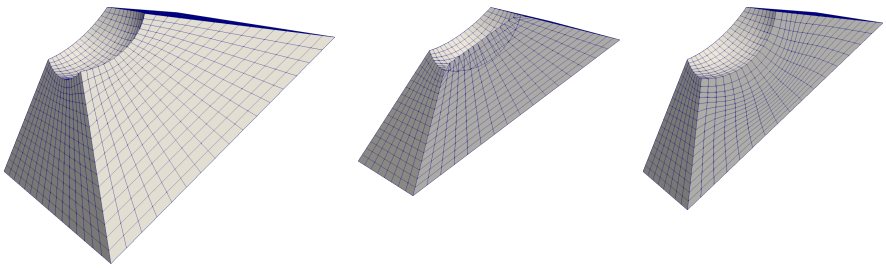


Figure 8: Resulting mesh of the outer of a sphere after the deformation by the RBF method (left) and its cross section view before (middle) and after (right) the volume mesh deformation. The mesh deformation fixes the reversed elements.

Since the problem size becomes very large in a three-dimensional context, it is necessary to accelerate the resolution of Equation (3). To this end, [44] suggests interpolating the target displacement in several steps. Firstly, a coarse set of nodes has to be selected among all the n_b border nodes. Then, the interpolation is performed with a very large radius r, denoted by r_0. Finally, the mesh is deformed again with an enriched set of nodes chosen by various criteria, while decreasing the influence radius r in order to increase the sparsity of the $\mathbf{M}$ operator. This last process is repeated k times until the error displacement $\mathbf{u}_{b\max}^{(k)}$ reaches a given criterion. At each iteration k, radius $r^{(k)}$ is chosen to be:

$$r^{(k)} = r_0.10^{\alpha+\beta\log_{10}\left(\frac{\mathbf{u}_{b\max}^{(k)}}{\mathbf{u}_{b\max}^{(0)}}\right)}, \tag{4}$$

with α and β two coefficients influencing the decrease of $r^{(k)}$.

Here, drawing inspiration from [44], a natural choice to enrich the set of selected nodes is to use the nodes introduced by the successive hierarchical refinements. Besides, the natural way to decrease the influence radius keeping a

constant bandwidth of $\mathbf{M}$ is to divide it by 2 at each iteration because the distance between two neighboring nodes is also divided by 2 between two successive refinements, such that:

$$r^{(k)} = r_0 \frac{1}{2^{k-1}}. \tag{5}$$

Compared to a monolithic resolution, this method allows to save many computational time keeping a similar quality of elements in the resulting mesh.

3. Numerical Results

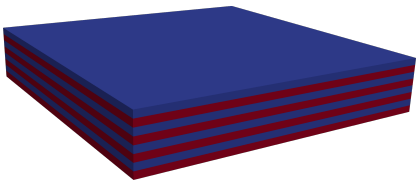
In this section, the parallel performance of the proposed methodology is assessed both in terms of scalability of the mesh generation phase and in terms of the convergence of the FETI solver. For the latter, our method is compared to configurations obtained with classical graph-based partitioning and intuitive decomposition in case of trivial geometry.

Three examples are provided: an academic stratified composite, a simplified turbine blade and an academic model of solid propellant. The proposed method has been implemented in a distributed memory parallelism framework in the Python language, using the Z-set finite element platform [43] for finite element computation and some meshing features together with the MPI standard for communication between processes. All tests have been conducted on a cluster of 2×12 cores Intel Xeon E5-2680 v3 processors of 2.50 GHz connected by an InfiniBand Mellanox network. Only one core is allocated per subdomain (pure MPI parallelism). For all computations, the FETI solver is used with the most advanced preconditioner (Dirichlet operator with stiffness scaling) and first-level projector (where the preconditioner is reused in the computation of the coarse matrix) [19]. This configuration is expected to be the most robust possible one with respect to heterogeneity in the absence of second level coarse problem. The FETI convergence criterion ε is set to 10^{-6}.

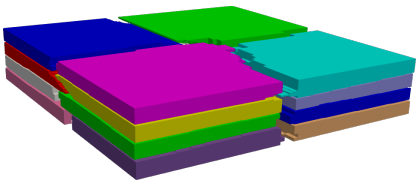
3.1. Stratified composite

First, the influence of the heterogeneity and its position with respect to the interface between subdomains on the convergence rate is discussed. The problem studied is a stratified composite with 8 linear elastic and isotropic layers (see Figure 9a). The Young modulus alternates between a high value E_1 and a lower one E_2. All layers share the same Poisson coefficient $\nu = 0.3$. Several ratios $\rho_E = E_1/E_2$ are used, from 1 to 10^5, to analyze the stability of the convergence. One of the sides is clamped and a tension and bending displacements are prescribed on the opposite face.

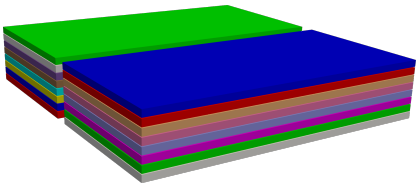
In the following, we perform a weak scalability study on several material configurations corresponding to an increasing ratio of heterogeneity. In order to do that, an initial pattern of 16 subdomains is repeated in two leading dimensions of the plate. The size of the local problems remains constant (32 768 elements per subdomain) while the global problem size increases. Three kinds of decomposition are studied:



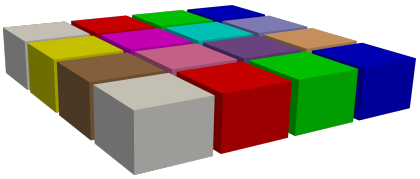
(a) Initial pattern where the soft layers (in red) and the stiff layers (in blue) are reprensented.



(b) *Graph-based* decomposition of the initial pattern into 16 subdomains.



(c) *Two-level* decomposition of the initial pattern in 16 subdomains.



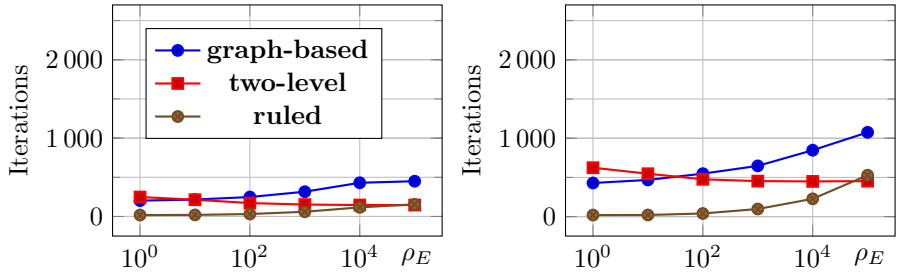
(d) *Ruled* decomposition of the initial pattern in 16 subdomains.

Figure 9: Initial stratified composite pattern with 545 025 nodes and 524 288 elements for different decompositions

- the *graph-based* decomposition, directly generated from the Metis graph partitioner, where the subdomains are heterogeneous and have irregular shape;
- the *two-level* decomposition which is the subject of this article, where no deformation step is required in this simple case, leading to homogeneous subdomains with regular shape which, however, may have very bad aspect ratio;
- the *ruled* decomposition, where subdomains are full-thickness cuboids of regular shape but no longer homogeneous; let us note .

Note that the choice of decomposition does not lead to extreme difference in the number of interface degrees of freedom. For instance, the smallest case leads to 100 000 interface degrees of freedom for the classical graph-based decomposition, 112 000 for the two-level decomposition and 147 000 for the ruled one. Moreover, the *ruled* decomposition is only used here for the sake of comparison, but would be hard, even impossible, to generalize to realistic geometries.

The convergence results of the FETI method with the different decompositions and Young modulus ratios are presented in Figure 10. Subfigures (a–e) refer to the 8-layers geometry, from 16 up to 400 subdomains. First, we observe that the ruled decomposition leads to a much faster convergence than the graph-based one. Both decompositions behave similarly with respect to the heterogeneity ratio. The FETI convergence is almost insensitive to small ratio ($\rho_E < 10^2$), but it is really slowed down beyond this limit, with a significant



(a) 1 pattern (16 subdomains), ∼ 1.6 million dof

(b) 4 patterns (64 subdomains), ∼ 6.4 millions dof

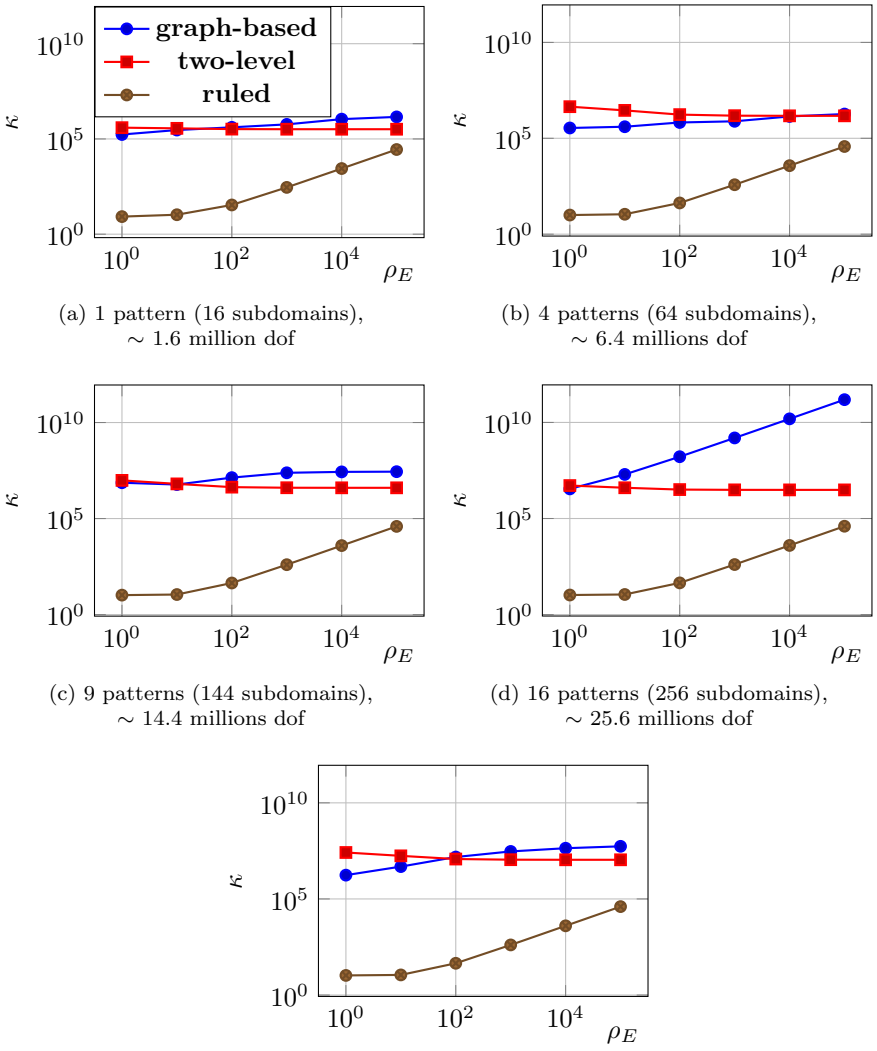
(c) 9 patterns (144 subdomains), ∼ 14.4 millions dof

(d) 16 patterns (256 subdomains), ∼ 25.6 millions dof

(e) 25 patterns (400 subdomains), ∼ 40 millions dof

(f) 9 layers, 81 subdomains, ∼ 1.2 millions dof

Figure 10: Stratified composite structure (a-e) 8 layers, (f) 9 layers: influence of the heterogeneity ratio ρ_E on the convergence of the FETI solver for three types of decomposition and an increasing number of subdomains (the grey background of figure (f) helps to distinguish the 9-layers case from the 8-layers cases).



(a) 1 pattern (16 subdomains), $\sim$ 1.6 million dof

(b) 4 patterns (64 subdomains), $\sim$ 6.4 millions dof

(c) 9 patterns (144 subdomains), $\sim$ 14.4 millions dof

(d) 16 patterns (256 subdomains), $\sim$ 25.6 millions dof

(e) 25 patterns (400 subdomains), $\sim$ 40 millions dof

Figure 11: 8-layered stratified composite structure: influence of the heterogeneity ratio ρ_E on the condition number κ of the FETI preconditioned operator for three types of decomposition and an increasing number of subdomains.

increase of the iteration number. These phenomena is amplified by the increase in the number of subdomains. Conversely, the two-level decomposition is almost insensitive to high stiffness ratio whereas its performance slightly degrades when this ratio tends to 1 (homogeneous behavior). This behavior may be attributed to the slenderness of the subdomains which penalizes the overall convergence. The slenderness tends to be less critical than material coefficient jumps on the interface for higher ratio, which explains the better performance of the two-level decomposition in such cases. These observations are confirmed by Figure 11, where the condition number κ of the FETI operator is estimated by the maximal Ritz value (since the minimal one is 1). Also note that the comparison between two-level and graph-based decomposition is questionable in the purely homogeneous case ($\rho_E = 1$) since there are no longer physical layer in such problems. Hence, the two-level decomposition would probably generate a splitting similar to the graph-based one without the extra-instruction regarding the physical layers.

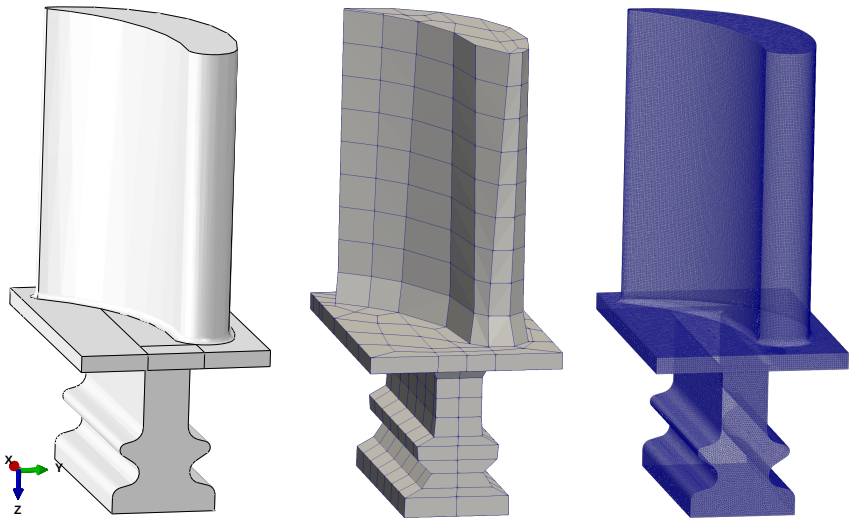
In the 8-layers case of Figure 10(a–e), the ruled decomposition most frequently outperforms the two other ones. By just adding one layer, it is possible to strongly degrades its performance, as illustrated in Figure 10(f) for the case of 81 subdomains. Indeed, in that case, the ruled decomposition (9×9 subdomains "in the plane") contains much more "bad heterogeneity patterns" (3 stiff "3D-channels" per subdomain) known to impede the convergence, according to the GENEO-theory [41] Thus, we expect that the two-level decomposition would perform even better with respect to two other decompositions when increasing the number of material layers. However, the 8-layers configuration remains interesting in the sense that it clearly shows that the two-level approach should not be used in all cases, especially when dealing with small heterogeneity ratio.

This study has highlighted that, from a convergence point-of-view, the two-level decomposition is not of much interest in the quasi-homogeneous case. It however becomes interesting in the presence of sufficiently strong ($\rho_E > 10^3$) and numerous heterogeneities. By comparing the ruled and graph-based decompositions, we see that regular interfaces can significantly improve the convergence (except when the gap of material coefficients becomes too high).

3.2. Aircraft engine turbine blade

The previous section has shown the good behavior of the method when the heterogeneity becomes important compared to a graph-based domain decomposition, despite subdomains with slender shape. Since the previous geometry was made of planar surfaces, no mesh deformation was performed and the parallel efficiency of the mesh generation process could not be demonstrated. We thus propose to demonstrate the meshing performance by a strong scalability study in the case of an aircraft engine turbine blade of a homogeneous material whose geometry was simplified (see Figure 12).

The coarse mesh is represented in Figure 12b. The fine mesh is obtained after four levels of refinement, leading to approximately 1.7 million nodes (5.1



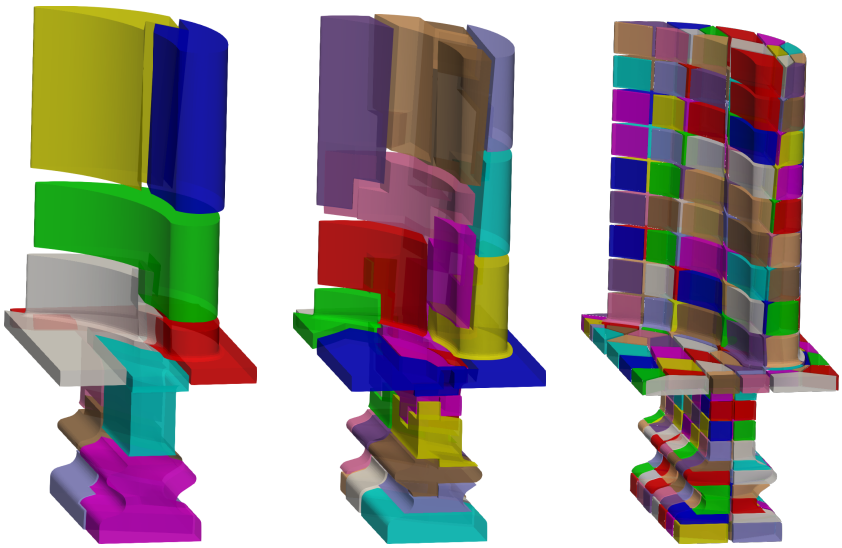
(a) CAD geometry of the blade

(b) Initial Coarse mesh with 400 hexahedral elements

(c) Fine surface mesh with 135 079 nodes

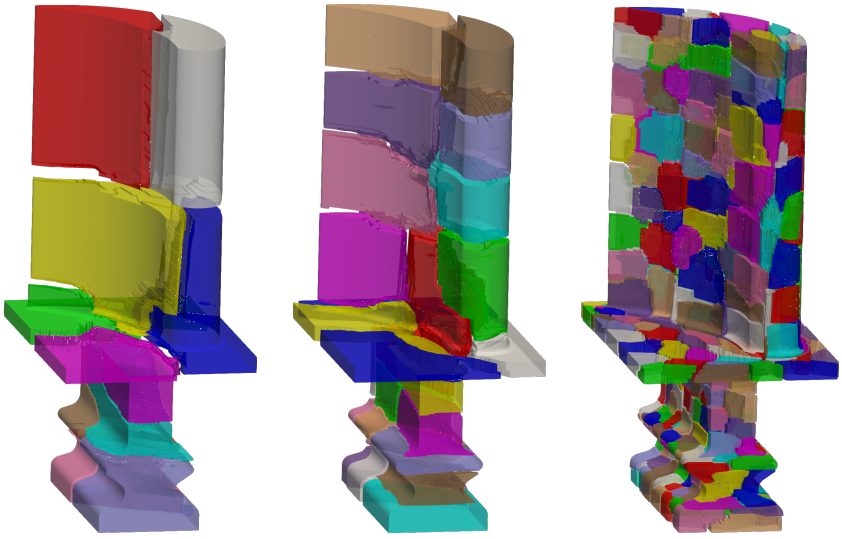
Figure 12: Initial data for the method

millions of degrees of freedom). For the two-level approach, the coarse mesh was divided in 10, 25, 50, 100, 200, 400 computational subdomains. For the graph-based approach, the fine mesh was decomposed in as many subdomains. Some decompositions are illustrated in Figures 13 and 14. Let us note that the computational mesh is unique and used by all decompositions, such that only the number and the shape of the subdomains vary between configurations. The strategies are evaluated both in terms of parallel processing of the mesh and in terms of performance of the solver.



(a) 10 subdomains (b) 25 subdomains (c) 400 subdomains

Figure 13: Two-level decompositions of the turbine blade



(a) 10 subdomains (b) 25 subdomains (c) 400 subdomains

Figure 14: Graph-based decompositions of the turbine blade

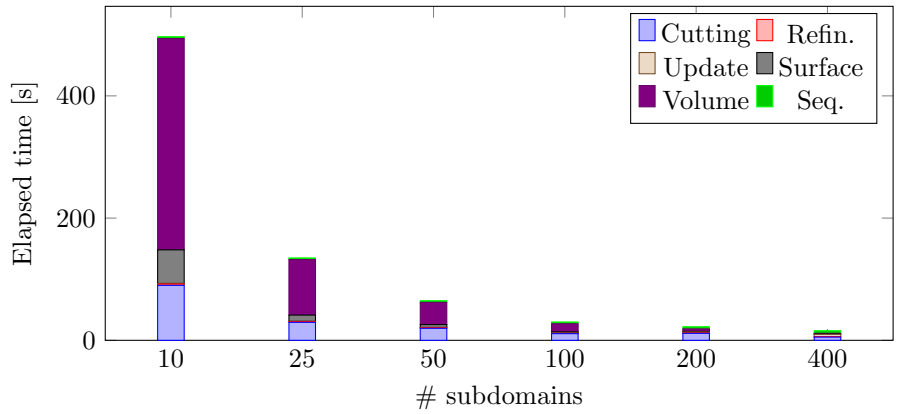


Figure 15: Elapsed time of the meshing process versus the number of subdomains for a constant size of the global problem (approximately 1.7M of nodes)

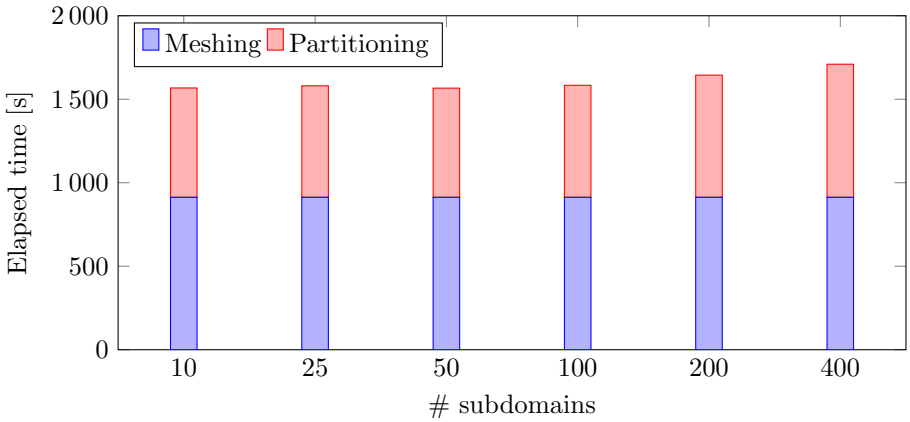
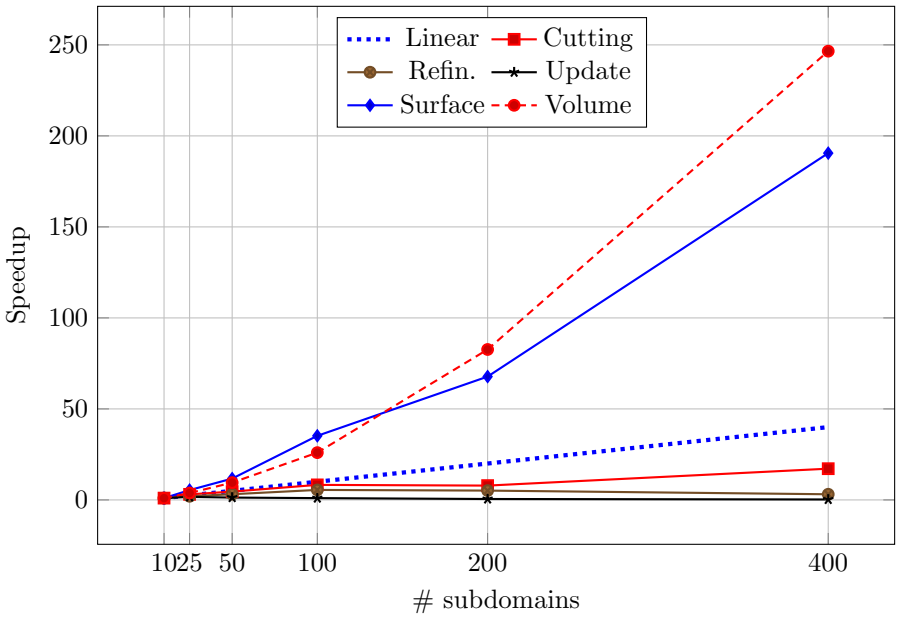
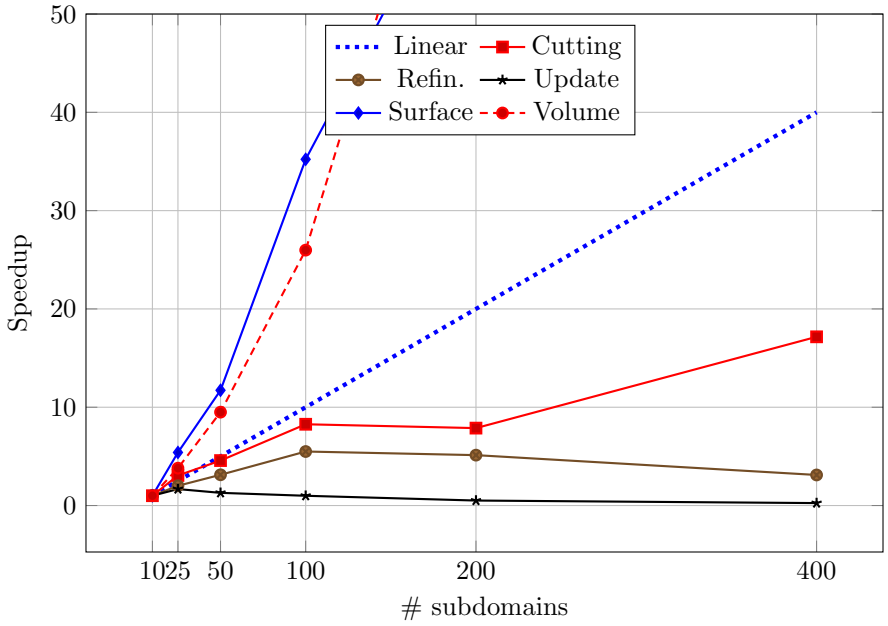


Figure 16: Elapsed time of the sequential meshing and partitioning of the blade with approximately 1.6M of nodes



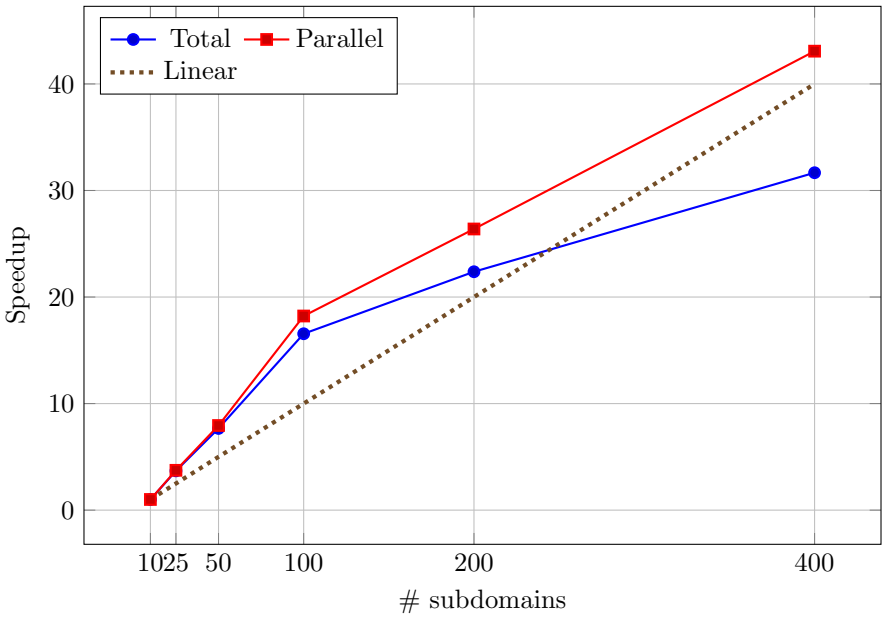
(a) Speedups of all individual timers



(b) zoom on Figure 17a

Figure 17: Speedup of the parallel mesh generation process for the blade example

The execution elapsed time of Algorithm 1 is described for each decompo-



(c) Speedup of cumulated timers (Parallel={Cutting, Refin., Update, Surface, Volume}, Total={Parallel, Seq.})

Figure 17: Speedup of the parallel mesh generation process for the blade example

	Elapsed time [s]							
N_s	Full Seq.	Total	Seq.	Cutting	Refin.	Update	Surface	Volume
10	1 567.5	496.86	0.61%	18.1%	0.44%	0.19%	11.06%	69.59%
25	1 580.52	134.8	1.94%	21.83%	0.81%	0.42%	7.55%	67.45%
50	1 566.53	64.81	4.05%	30.33%	1.08%	1.14%	7.23%	56.17%
100	1 583.47	30.02	9.7%	36.25%	1.33%	3.19%	5.2%	44.34%
200	1 644.28	22.21	15.73%	51.42%	1.92%	8.44%	3.65%	18.84%
400	1 709.77	15.69	26.94%	33.4%	4.48%	24.41%	1.84%	8.94%

Table 1: Elapsed time of the blade's meshing process for different N_s number of subdomains. The first column 'Full Seq.' is the elapsed time of a full sequential meshing and partitioning from the input geometry.

sition in Figure 15 and Table 1. The code is instrumented by six timers (line numbers refer to Algorithm 1):

- *Seq.* is the time spent in the sequential steps (lines 1, 2 and 3);
- *Cutting* is the time spent in the restriction of the fine geometry (line 4);
- *Refin.* and *Update* represent respectively the time spent in the real refinement of the mesh and in the update of the interface data (their sum is the time of line 5);
- *Surface* represents the time corresponding to the surface mesh deformation

on the boundaries (line 6 to 10);

- *Volume* represents the duration of the volumetric RBF mesh interpolation (line 11).

Figures 17a and 17b plot the speedup for the different timers. The speedup is computed with respect to the 10-subdomain (sd) configuration according to the following formula:

$$\text{speedup}(N_s \text{ sd}) = \frac{\text{Elapsed time}(10 \text{ sd})}{\text{Elapsed time}(N_s \text{ sd})}.$$

First, we observe that the total elapsed time decreases quickly when the number of subdomains increases. However, the time spent in the sequential part of the process still increases with the number of subdomains, becoming the major time of the meshing process as can be seen on the last line of Table 1. This increase of the sequential part also causes the saturation of the Total speedup curve of Figure 17a. This may be explained by the partitioning steps of lines 2 and 3 in Algorithm 1, where a union of all coarse domains is performed in order to generate the interface data. The reader may note that this union step could be further improved. *Cutting* and *Surface* timers, which represent a large part of the total time, have almost linear speedups, unlike *Refin.* and *Update* which involve MPI exchanges (Neighbor-All-to-All). We also note that the *Volume* speedup is super-Linear, where a sparse problem has to be solved with at least quadratic complexity.

Then, we compare the elapsed time of the whole generation procedure (mesh and splitting) corresponding to our two-level approach with the "classical" one consisting in generating the fine mesh directly from the CAD definition prior to its (sequential) splitting (see Figure 16). For the time measurements of the classical approach, Abaqus 6.13-1 [40] is used as a mesher and Z-set interfaced with Scotch [37] is used as mesh partitioner. We notice from Figure 15 and Table 1 a significant improvement of the proposed approach on the total return time, even when the number of subdomains is low.

Finally, in order to investigate the efficiency of the FETI resolution, finite element computations are conducted on the various mesh configurations generated. The mechanical problem is linear (small strain) elastic isotropic with a Young's modulus of 200 GPa and a Poisson's coefficient of 0.3, with a thermal expansion coefficient of $12 \cdot 10^{-6}$ K^{-1} and a mass density of 8 000 kg m^{-3}. A centrifugal force is applied corresponding to a rotation around an axis placed 100 mm under the foot along the Z-axis (see Figure 12a) at a rate of $3 \cdot 10^6$ rad s^{-1}. A homogeneous thermal strain, due to a temperature increase of 980 K, is applied together with pressure on the intrados of the blade. Displacement on the bottom face along the Z-axis is set to 0 and an edge on this face is clamped to remove all rigid body motions. Figure 18 shows that the convergence of the FETI solver on a decomposition resulting from the method described in this paper is better than a graph-based decomposition when the subdomains coincide with few coarse elements (large N_s) or when the structure is decomposed

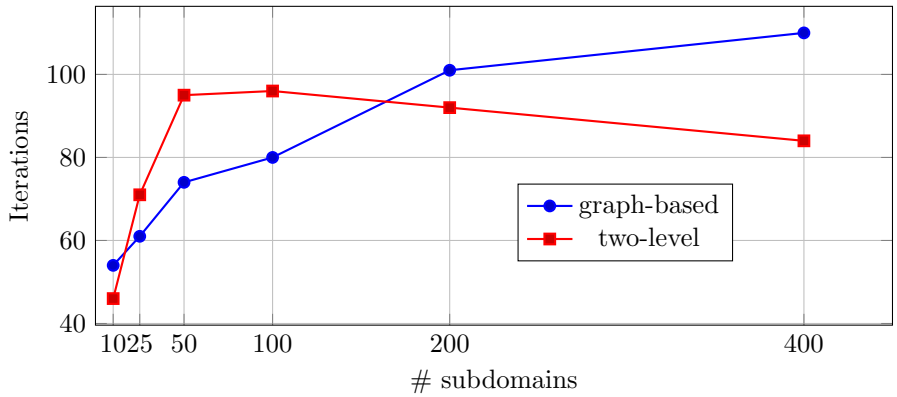


Figure 18: Number of iterations of the FETI method with a relative convergence ratio of 10^{-6} for the turbine blade case for increasing number of subdomains.

into a very few number of subdomains. These configurations indeed lead to the most regular interfaces for the two-level approach.

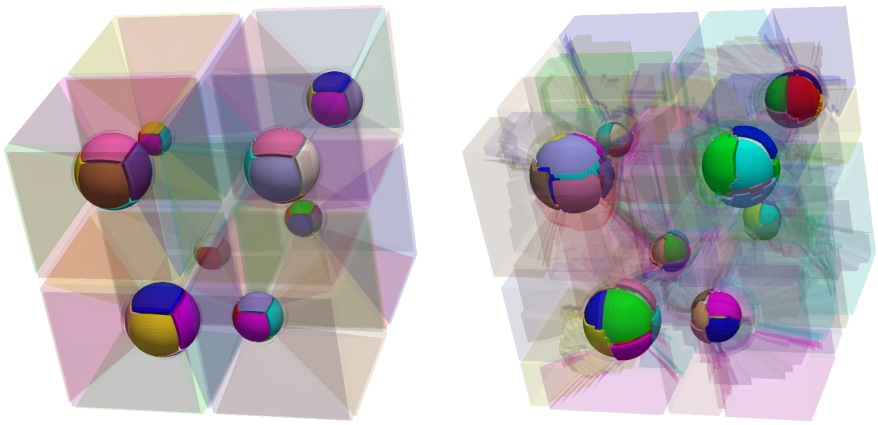
3.3. Multi-inclusion solid propellant

The previous two subsections showed a heterogeneous case without mesh deformation and a homogeneous one with mesh deformation. We now asses the method on a heterogeneous case with mesh deformation. We consider a solid propellant model (see Figure 19) obtained by the replication of the inclusion pattern presented in Figure 1, where the position and the radius of inclusions are randomly selected using a uniform distribution. Additionnaly, the radius of the inclusions is bounded between 25% and 75% of the replicated cubes size. Two linear elastic isotropic materials are used. The inclusions have a Young modulus $E_1 = 69\,000$ MPa and a Poisson coefficient $\nu_1 = 0.346$. The matrix is quasi-incompressible with a Poisson coefficient of $\nu_2 = 0.499$ and a softer Young Modulus $E_2 = 0.96$ MPa, which leads to a ratio of heterogeneity $\rho_E = E_1/E_2 = 7.2.10^5$). The resulting cubic structure is clamped on one face and a displacement in the three directions is prescribed on the opposite one. Since, the matrix is quasi-incompressible, three fields mixed elements [47] are used in it (pressure and volumetric variation are the additional fields).

A weak scalability study is performed. The initial pattern of Figure 1 is divided into 13 subdomains and replicated in the three space directions to obtain a set of global structures with a total number of subdomains of $13 \times N_c^3$, N_c being the number of replications in one direction.

Figure 20 and Table 2 display the convergence of the FETI solver for both the two-level method and the graph-based one (directly applied on each global mesh) for values of N_c between 1 and 4. We observe that the two-level decomposition always makes the FETI solver converge faster than the graph-based one. Depending on the case, the number of iterations saved ranges between 20% and 55% and continuously increases with the problem size.

The elapsed time of the mesh generation process is plotted in Figure 21 and detailed in Table 3 for N_c between 1 and 3. Note that the results for $N_c = 4$ has been obtained by oversubscription of MPI process on the computing hosts so that the associated timers are not relevant. We notice that the *Surface* and *Volume* timers associated to the mesh deformation part, which represent a significant part of the total time but involving local problems of constant size, remains constant. The *Refin.* and *Update* are slightly increasing, which may be explained by the MPI data exchanges involved in those steps. Indeed, the MPI latency is expected to increase from $N_c = 1$, where each MPI process remains on the same host, to $N_c = 3$, where 351 MPI processes spread over 15 computing host, interconnected at least by two level of Infiniband switches. Given the elapsed time's order of magnitude, this latency may be significant. Finally, the increase of the *Cutting* counter is due to the increase of the 2D skin mesh (storing the global geometry) as the geometry of the total problem also increases, making the cutting step (line 4 of Algorithm 1) more expensive.



(a) Two-level decomposition (b) Graph-based decomposition

Figure 19: The solid propellant containing $2 \times 2 \times 2$ inclusions, the different colors represent the different subdomains; the transparent elements represent the matrix and the others are in the inclusions

	Iterations	
N_s	Graph-based	Two-level
13	100	45
104	314	254
351	489	395
832	849	554

Table 2: Number of iterations to reach convergence for the solid propellant test case

A strong scalability study of the meshing process is conducted for the $N_c = 3$ test case with approximately 34.6 millions of degrees of freedom. The elapsed

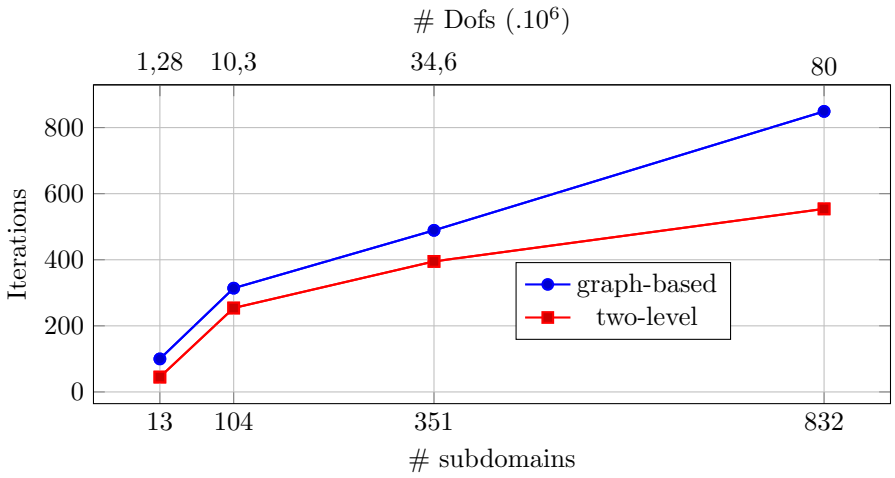


Figure 20: Convergence of the solid propellant test case for a constant size of local problems

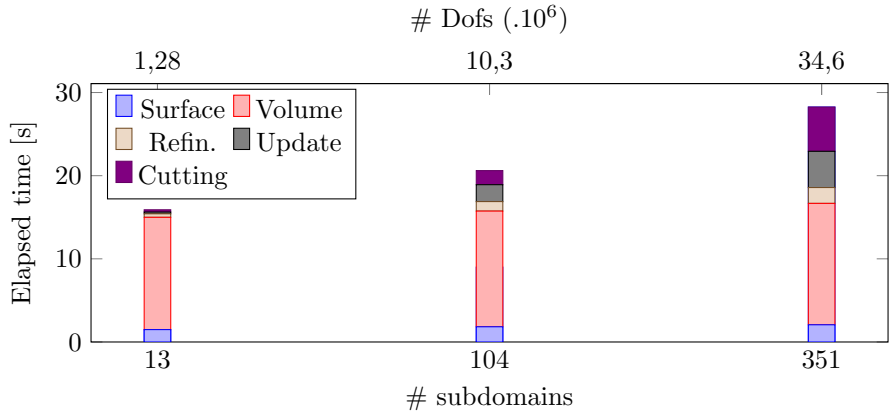


Figure 21: Time of the meshing process of the solid propellant versus the number of subdomains for a constant size of local problems (weak scalability).

	Elapsed time [s]					
N_s	Total	Cutting	Refin.	Update	Surface	Volume
13	15.9	1.73%	2.66%	1.31%	9.36%	84.94%
104	20.61	8.22%	5.55%	9.86%	8.87%	67.5%
351	28.25	18.83%	6.74%	15.43%	7.32%	51.68%

Table 3: Detailed timers of the meshing process of the solid propellant versus the number of subdomains for a constant size of local problems (weak scalability).

time of the mesh generation process is shown in Figure 22 and detailed in Table 4. Note that, unfortunately, it has not been possible to properly balance the load between subdomains for the 108- and 54-subdomain test-cases. Indeed,

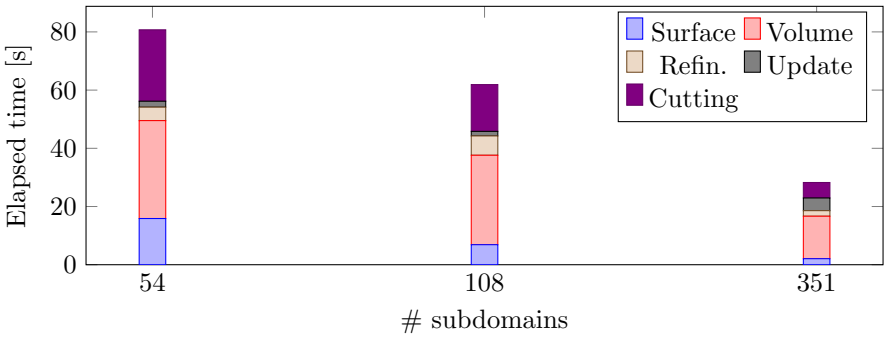


Figure 22: Time of the meshing process of the solid propellant versus the number of subdomains for a constant size the global problem (strong scalability, 34.6 millions of degrees of freedom).

in the case of 108 subdomains, subdomains are made out of either 3 or 4 coarse elements. In the 54-subdomains case, each inclusion is a subdomain made out of 7 coarse elements whereas the matrix subdomains have only 6 coarse elements.

We notice that the *Cutting* and *Surface* timers decrease for the same reasons as in the turbine blade case. However, between 108- and 54-subdomains cases, the other timers (*Refin.*, *Update* and *Volume*) remain almost constant because the most computationally expensive subdomains are made up of the same number of coarse faces (those in the $\mathcal{F}_r^s$ sets). This leads to RBF matrix M of the same size between the 54- and 108-subdomains cases.

	Elapsed time [s]					
N_s	Total	Cutting	Refin.	Update	Surface	Volume
54	80.71	30.4%	5.72%	2.53%	19.67%	41.69%
108	61.89	25.97%	10.72%	2.53%	11.12%	49.66%
351	28.25	18.83%	6.74%	15.43%	7.32%	51.68%

Table 4: Detailed timers of the meshing process of the solid propellant versus the number of subdomains for a constant size the global problem (strong scalability, 34.6 millions of degrees of freedom).

4. Conclusion

This paper has presented a new parallel two-level substructuring approach for non-overlapping domain decompostion computations. The main principle of this method is, schematically, to swap the splitting and fine mesh generation steps compared to the classical preprocessing approach. In order to do that, the domain splitting step is done at the very beginning of the whole process, using a coarse mesh that takes the underlying material distribution into account. Thereafter, a fine conforming mesh is generated in parallel on each subdomain followed by a final morphing step to ensure an accurate representation of the

input geometry. As a result, this method produces regular-shaped and homogeneous subdomains. These subdomains are well suited to Domain Decomposition Method such as FETI or BDD because they reduce the condition number of the system to be solved, leading to faster convergence of the solver compared to classical graph-based partitioning. Also, a significant part of the mesh generation can be conducted in parallel, with very few neighbors exchanges.

Our numerical experiments confirm that, in most cases, the proposed decomposition improves the convergence rate of FETI resolutions on homogeneous and strongly heterogeneous test cases compared to automatic decompositions arising from graph partitioning software. However, in some few situations, the two-level decomposition may lead to worse convergence compared to the classical graph-based approach. Those situations may be encountered on cases where, either subdomains are built from the aggregation of too many coarse elements leading to poor geometrical shapes, or two-level subdomains have too slender aspect ratio which are not legitimated by sufficiently strong heterogeneity. Regarding the parallel mesh generation, the method exhibits good strong scalability and promising weak scalability that could be extensively tested on an higher number of computing cores (several thousands).

From the author's point of view, this work opens up two outlooks. The first one could be to directly partition the geometry when it is available, for instance in CAD format, which enables to avoid handling a too fine discretized description of the geometry. The second one is to investigate alternative refinement methods, as suggested in Section 2.4, to obtain meshes of better quality when dealing with complex geometries.

Appendix A. RBF functions

Table A.5 presents the 14 RBF functions which allow the use of a 1-degree polynomial function p [11] to make the system unconditionally definite.

These functions are generally scaled by the influence radius of the RBF r, using the change of variable $\xi \equiv x/r$. In Table A.5, the first eight functions are of compact support on the $[0, 1]$ interval. Referring to [46], they are defined such that:

$$f(\xi) = \begin{cases} f(\xi) & \text{if} \quad \xi \in [0,1], \\ 0 & \text{else.} \end{cases}$$

With these functions, only the nodes inside the r-radius sphere are affected by the displacement of its center. Moreover, this choice of compact support functions leads to an unconditionally definite matrix $\mathbf{M}$ [15]. Higher values for r lead generally to more accurate solutions. However, higher values of the support radius also result in denser matrix $\mathbf{M}$, whereas lower values of r result in sparser matrix $\mathbf{M}$ which can be solved more efficiently (for instance by a Krylov solver).

Among these eight functions, the four first are minimal degree polynomials which are $\mathcal{C}^n$ with $n \in \{0, 2, 4, 6\}$ (CP≡Continuous Polynomial). The four subsequent are Thin Plate Spline based function series (CTPS≡Continuous Thin

No.	Name	$f(\xi)$
1	CP $\mathcal{C}^0$	$(1-\xi)^2$
2	CP $\mathcal{C}^2$	$(1-\xi)^4(4\xi+1)$
3	CP $\mathcal{C}^4$	$(1-\xi)^6(\frac{35}{3}\xi^2+6\xi+1)$
4	CP $\mathcal{C}^6$	$(1-\xi)^8(32\xi^3+25\xi^2+8\xi+1)$
5	CTPS $\mathcal{C}^0$	$(1-\xi)^5$
6	CTPS $\mathcal{C}^1$	$1+\frac{80}{3}\xi^2-40\xi^3+15\xi^4-\frac{8}{3}\xi^5+20\xi^2\log(\xi)$
7	CTPS $\mathcal{C}^2_a$	$1-30\xi^2-10\xi^3+45\xi^4-6\xi^5-60\xi^3\log(\xi)$
8	CTPS $\mathcal{C}^2_b$	$1-20\xi^2+80\xi^3-45\xi^4-16\xi^5+60\xi^4\log(\xi)$
9	TPS	$x^2\log(x)$
10	MQB	$\sqrt{a^2+x^2}$
11	IMQB	$\sqrt{\frac{1}{a^2+x^2}}$
12	QB	$1+x^2$
13	IQB	$\frac{1}{1+x^2}$
14	Gaussian	$\mathrm{e}^{-\frac{x^2}{2}}$

Table A.5: Radial Basis Functions (CP: Continous Polynomial, TPS: Thin Plate Spline, CTPS: Continuous TPS, MQB: Multiquadratic Biharmonics, IMQB: Inverse Multiquadratic Biharmonics, QB: Quadratic Biharmonics, IQB: Inverse Quadratic Biharmonics) [11, 46]

Plate Spline). Similarly, these functions have the minimal form for the $\mathcal{C}^n$ continuity with $n \in \{0,1,2\}$. In this theory [46], two $\mathcal{C}^2$ continuous functions are possible; they are subscripted by $_a$ and $_b$.

The other six global support functions cover the entire interpolation domain, which leads to dense operator $\mathbf{M}$. MQB and IMQB functions have a shape parameter a which controls the thickness of these functions.

Appendix B. Ray-triangle intersection algorithm

This algorithm, presented in [33], allows to find, from a node $\mathbf{x}_{r,k}$ and an oriented direction $\mathbf{d}_\varphi$, the intersection of the ray $(\mathbf{x}_{r,k}, \mathbf{d}_\varphi)$ with a triangular mesh.

Let us consider nodes $(\mathbf{x}_{r,k})_k$ belonging to an edge of $\mathcal{E}_{r,\partial}$ (in Section 2.5.2) or a face of $\mathcal{F}_{r,\phi}$ (in Section 2.5.4), and let $\mathbf{d}_\varphi$ be the associated normal vector. The new positions are sought under the form:

$$\mathbf{x}^{new}_{r,k} = \mathbf{x}_{r,k} + \ell_k \mathbf{d}_\varphi, \qquad \forall k.$$

Moreover, the node $\mathbf{x}^{new}_{r,k}$ has to be in a triangle defined by its 3 vertices $(\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2)$, so that its position in barycentric coordinates (u_k, v_k) can be written as:

$$\mathbf{x}^{new}_{r,k}(u_k, v_k) = (1-u_k-v_k)\mathbf{v}_0 + u_k\mathbf{v}_1 + v_k\mathbf{v}_2.$$

The node is inside the triangle if and only if $u_k \geqslant 0$ and $v_k \geqslant 0$ and $u_k + v_k \leqslant 1$.

This leads to solving the following equation:

$$\begin{bmatrix} -\mathbf{d}_\varphi & (\mathbf{v}_1 - \mathbf{v}_0) & (\mathbf{v}_2 - \mathbf{v}_0) \end{bmatrix} \begin{bmatrix} \ell_k \\ u_k \\ v_k \end{bmatrix} = \mathbf{x}_{r,k} - \mathbf{v}_0. \tag{B.1}$$

This 3×3 system can be solved algebraically using Cramer's rule. The solution can be written using the notations $\mathbf{e}_1 = \mathbf{v}_1 - \mathbf{v}_0$, $\mathbf{e}_2 = \mathbf{v}_2 - \mathbf{v}_0$ and $\mathbf{e}_{r,k} = \mathbf{x}_{r,k} - \mathbf{v}_0$, as:

$$\begin{bmatrix} \ell_k \\ u_k \\ v_k \end{bmatrix} = \frac{1}{\det(-\mathbf{d}_\varphi, \mathbf{e}_1, \mathbf{e}_2)} \begin{bmatrix} \det(\mathbf{e}_{r,k}, \mathbf{e}_1, \mathbf{e}_2) \\ \det(-\mathbf{d}_\varphi, \mathbf{e}_{r,k}, \mathbf{e}_2) \\ \det(-\mathbf{d}_\varphi, \mathbf{e}_1, \mathbf{e}_{r,k}) \end{bmatrix}. \tag{B.2}$$

Moreover, with the relation between determinant and cross product, introducing $\mathbf{p}_1 = \mathbf{e}_{r,k} \times \mathbf{e}_1$ and $\mathbf{p}_2 = \mathbf{d}_e \times \mathbf{e}_2$, the previous system can be rewritten as:

$$\begin{bmatrix} \ell_k \\ u_k \\ v_k \end{bmatrix} = \frac{1}{(\mathbf{d}_e \times \mathbf{e}_2) \cdot \mathbf{e}_1} \begin{bmatrix} -(\mathbf{e}_1 \times \mathbf{e}_{r,k}) \cdot \mathbf{e}_2 \\ (\mathbf{d}_\varphi \times \mathbf{e}_2) \cdot \mathbf{e}_{r,k} \\ (\mathbf{e}_{r,k} \times \mathbf{e}_1) \cdot \mathbf{d}_\varphi \end{bmatrix} = \frac{1}{\mathbf{p}_2 \cdot \mathbf{e}_1} \begin{bmatrix} \mathbf{p}_1 \cdot \mathbf{e}_2 \\ \mathbf{p}_2 \cdot \mathbf{e}_{r,k} \\ \mathbf{p}_1 \cdot \mathbf{d}_\varphi \end{bmatrix}.$$

The resulting algorithm corresponds to Algorithm 2.

Algorithm 2: Ray-triangle intersection algorithm

Data: direction $\mathbf{d}_\varphi$ and node $\mathbf{x}_{r,k}$
for *triangle* $\{(\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2)_i\} \in (\mathcal{F}_\phi)$ **do**
  $\mathbf{e}_1 = \mathbf{v}_1 - \mathbf{v}_0$, $\mathbf{e}_2 = \mathbf{v}_2 - \mathbf{v}_0$, $\mathbf{e}_{r,k} = \mathbf{x}_{r,k} - \mathbf{v}_0$
  $\mathbf{p}_2 = \mathbf{d}_\varphi \times \mathbf{e}_2$, $\mathbf{p}_1 = \mathbf{e}_{r,k} \times \mathbf{e}_1$
  $\Delta = \mathbf{p}_2 \cdot \mathbf{e}_1$
  $u_k = \frac{\mathbf{p}_2 \cdot \mathbf{e}_{r,k}}{\Delta}$, $v_k = \frac{\mathbf{p}_1 \cdot \mathbf{d}_\varphi}{\Delta}$
  if $u_k \geqslant 0$ **and** $v_k \geqslant 0$ **and** $(u_k + v_k) \leqslant 1$ **then**
    return $\ell_k = \frac{\mathbf{p}_1 \cdot \mathbf{e}_2}{\Delta}$ // The projection is in the triangle
  end
end

References

[1] 3D Systems, Inc. *StereoLithography Interface Specification*. 3D Systems, Inc., July 1988.

[2] E. Amezua, M. V. Hormaza, A. Hernández, and M. B. G. Ajuria. A method for the improvement of 3D solid finite-element meshes. *Advances in Engineering Software*, 22(1):45–53, Jan. 1995. doi:10.1016/0965-9978(95)00004-G.

[3] J. Besson and R. Foerch. Large scale object-oriented finite element code design. *Computer Methods in Applied Mechanics and Engineering*, 142(1-2):165–187, Mar. 1997. doi:10.1016/S0045-7825(96)01124-3.

[4] M. Bhardwaj, D. Day, C. Farhat, M. Lesoinne, K. Pierson, and D. Rixen. Application of the FETI method to ASCI problems/scalability results on 1000 processors and discussion of highly heterogeneous problems. *International Journal for Numerical Methods in Engineering*, 47(1-3):513–535, Jan. 2000. doi:10.1002/(SICI)1097-0207(20000110/30)47:1/3<513::AID-NME782>3.0.CO;2-V.

[5] C. Bovet, O. Ciobanu, A. Parret-Freaud, and V. Chiaruttini. A parallel mesh refinement process tailored to domain decomposition methods. In *Proceedings of the Sixth International Conference on Parallel, Distributed, GPU and Cloud Computing for Engineering*, page 20, Pollack Mihály Faculty of Engineering and Information Technology, University of Pécs, Hungary, June 2019. doi:10.4203/ccp.112.20.

[6] C. Bovet, A. Parret-Fréaud, N. Spillane, and P. Gosselet. Adaptive multipreconditioned FETI: Scalability results and robustness assessment. *Computers & Structures*, 193:1–20, Dec. 2017. doi:10.1016/j.compstruc.2017.07.010.

[7] M. D. Buhmann. Radial basis functions. *Acta Numerica*, 9:1–38, Jan. 2000. doi:10.1017/S0962492900000015.

[8] J. Chen, Z. Xiao, Y. Zheng, J. Zou, D. Zhao, and Y. Yao. Scalable generation of large-scale unstructured meshes by a novel domain decomposition approach. *Advances in Engineering Software*, 121:131–146, July 2018. doi:10.1016/j.advengsoft.2018.04.005.

[9] C. Chevalier and F. Pellegrini. PT-Scotch: A tool for efficient parallel graph ordering. *Parallel Computing*, 34(6):318–331, July 2008. doi:10.1016/j.parco.2007.12.001.

[10] T. Coupez, H. Digonnet, and R. Ducloux. Parallel meshing and remeshing. *Applied Mathematical Modelling*, 25(2):153–175, Dec. 2000. doi:10.1016/S0307-904X(00)00045-7.

[11] A. De Boer, M. S. Van Der Schoot, and H. Bijl. Mesh deformation based on radial basis function interpolation. *Computers & Structures*, 85(11-14):784–795, June 2007. doi:10.1016/j.compstruc.2007.01.013.

[12] M. Dryja. A Neumann-Neumann algorithm for a mortar discretization of elliptic problems with discontinuous coefficients. *Numer. Math.*, 99(4):645–656, Feb. 2005. doi:10.1007/s00211-004-0573-2.

[13] M. Dryja and W. Proskurowski. A FETI-DP Method for the Mortar Discretization of Elliptic Problems with Discontinuous Coefficients. In T. J. Barth, M. Griebel, D. E. Keyes, R. M. Nieminen, D. Roose, T. Schlick, R. Kornhuber, R. Hoppe, J. Périaux, O. Pironneau, O. Widlund, and J. Xu, editors, *Domain Decomposition Methods in Science and Engineering*, volume 40, pages 345–352. Springer-Verlag, Berlin/Heidelberg, 2005. doi:10.1007/3-540-26825-1_34.

[14] C. Farhat and F.-X. Roux. A method of finite element tearing and interconnecting and its parallel solution algorithm. *International Journal for Numerical Methods in Engineering*, 32(6):1205–1227, Oct. 1991. doi:10.1002/nme.1620320604.

[15] G. E. Fasshauer. *Meshfree Approximation Methods with Matlab: (With CD-ROM)*, volume 6 of *Interdisciplinary Mathematical Sciences*. WORLD SCIENTIFIC, Apr. 2007. doi:10.1142/6437.

[16] F. Feyel. Some new technics regarding the parallelisation of ZéBuLoN, an object oriented finite element code for structural mechanics. *ESAIM: M2AN*, 36(5):923–935, 2002. doi:10.1051/m2an:2002040.

[17] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An Algorithm for Finding Best Matches in Logarithmic Expected Time. *ACM Trans. Math. Softw.*, 3(3):209–226, Sept. 1977. doi:10.1145/355744.355745.

[18] C. Geuzaine and J.-F. Remacle. Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309 – 1331, 2009.

[19] P. Gosselet and C. Rey. Non-overlapping domain decomposition methods in structural mechanics. *Archives of Computational Methods in Engineering*, 13(4):515–572, Dec. 2006. doi:10.1007/BF02905857.

[20] P. Gosselet, D. J. Rixen, F.-X. Roux, and N. Spillane. Simultaneous FETI and block FETI: Robust domain decomposition with multiple search directions. *International Journal for Numerical Methods in Engineering*, 104(10):905–927, May 2015. doi:10.1002/nme.4946.

[21] L. Grigori, S. Moufawad, and F. Nataf. Enlarged krylov subspace conjugate gradient methods for reducing communication. *SIAM J. Matrix Anal. & Appl*, 2016.

[22] P. Have, R. Masson, F. Nataf, M. Szydlarski, H. Xiang, and T. Zhao. Algebraic Domain Decomposition Methods for Highly Heterogeneous Problems. *SIAM Journal on Scientific Computing*, 35(3):C284–C302, 2013. URL `https://hal.archives-ouvertes.fr/hal-00611997`.

[23] B. Hendrickson. Chaco. In D. Padua, editor, *Encyclopedia of Parallel Computing*, pages 248–249. Springer US, Boston, MA, 2011. doi:10.1007/978-0-387-09766-4_310.

[24] ISO/TC 184/SC 4 Industrial data. Industrial automation systems and integration – Product data representation and exchange – Part 21: Implementation methods: Clear text encoding of the exchange structure. ISO 10303-21, International Organization for Standardization, Geneva, CH, 2016.

[25] Y. Ito and K. Nakahashi. Direct surface triangulation using stereolithography data. *AIAA Journal*, 40(3):490–496, 2002. doi:10.2514/2.1672.

[26] G. Karypis and V. Kumar. A Parallel Algorithm for Multilevel Graph Partitioning and Sparse Matrix Ordering. *Journal of Parallel and Distributed Computing*, 48(1):71–95, Jan. 1998. doi:10.1006/jpdc.1997.1403.

[27] D. LaSalle, M. M. A. Patwary, N. Satish, N. Sundaram, P. Dubey, and G. Karypis. Improving Graph Partitioning for Modern Graphs and Architectures. In *Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms*, IA3 '15, pages 14:1–14:4, New York, NY, USA, 2015. ACM. doi:10.1145/2833179.2833188. event-place: Austin, Texas.

[28] J.-M. Le Gouez. Numerical properties and gpu implementation of a high order finite volume scheme. In *NASA Advanced Modeling & Simulation (AMS) Seminar Series*, 2016.

[29] A. Loseille, V. Menier, and F. Alauzet. Parallel Generation of Large-size Adapted Meshes. *Procedia Engineering*, 124:57–69, 2015. doi:10.1016/j.proeng.2015.10.122.

[30] R. Löhner. Recent Advances in Parallel Advancing Front Grid Generation. *Archives of Computational Methods in Engineering*, 21(2):127–140, June 2014. doi:10.1007/s11831-014-9098-8.

[31] J. Mandel. Balancing domain decomposition. *Communications in Numerical Methods in Engineering*, 9(3):233–241, Mar. 1993. doi:10.1002/cnm.1640090307.

[32] J. Mandel and B. Sousedík. Adaptive selection of face coarse degrees of freedom in the BDDC and the FETI-DP iterative substructuring methods. *Computer Methods in Applied Mechanics and Engineering*, 196(8):1389–1399, Jan. 2007. doi:10.1016/j.cma.2006.03.010.

[33] T. Möller and B. Trumbore. Fast, Minimum Storage Ray-Triangle Intersection. *Journal of Graphics Tools*, 2(1):21–28, Jan. 1997. doi:10.1080/10867651.1997.10487468.

[34] M. Mäntylä. *An Introduction to Solid Modeling*. Computer Science Press, Inc., New York, NY, USA, 1987.

[35] C. Pechstein and R. Scheichl. Analysis of FETI methods for multiscale PDEs. Part II: interface variation. *Numerische Mathematik*, 118(3):485–529, July 2011. doi:10.1007/s00211-011-0359-2.

[36] C. Pechstein and R. Scheichl. Weighted Poincaré inequalities. *IMA Journal of Numerical Analysis*, 33(2):652–686, 2013.

[37] F. Pellegrini and J. Roman. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In G. Goos, J. Hartmanis, J. van Leeuwen, H. Liddell, A. Colbrook, B. Hertzberger, and P. Sloot, editors, *High-Performance Computing and Networking*, volume 1067, pages 493–498. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996. doi:10.1007/3-540-61142-8_588.

[38] D. J. Rixen and C. Farhat. A simple and efficient extension of a class of substructure based preconditioners to heterogeneous structural mechanics problems. *International Journal for Numerical Methods in Engineering*, 44(4):489–516, Feb. 1999. doi:10.1002/(SICI)1097-0207(19990210)44:4<489::AID-NME514>3.0.CO;2-Z.

[39] M. Selim and K. Komullil. Mesh Deformation Approaches A Survey. *Journal of Physical Mathematics*, 7(2), 2016. doi:10.4172/2090-0902.1000181.

[40] Simulia. Abaqus 6.13-1, 2013. URL `http://dsk.ippt.pan.pl/docs/abaqus/v6.13/books/usi/default.htm`.

[41] N. Spillane and D. J. Rixen. Automatic spectral coarse spaces for robust finite element tearing and interconnecting and balanced domain decomposition algorithms. *International Journal for Numerical Methods in Engineering*, 95(11):953–990, 2013. doi:10.1002/nme.4534.

[42] D. Stefanica. A Numerical Study of FETI Algorithms for Mortar Finite Element Methods. *SIAM Journal on Scientific Computing*, 23(4):1135–1160, Jan. 2001. doi:10.1137/S1064827500378829.

[43] Transvalor S.A. *Z-set 9.0 user manual*, 2019. URL `http://www.zset-software.com/`.

[44] G. Wang, X. Chen, and Z. Liu. Mesh deformation on 3d complex configurations using multistep radial basis functions interpolation. *Chinese Journal of Aeronautics*, 31(4):660–671, Apr. 2018. doi:10.1016/j.cja.2018.01.028.

[45] X.-Q. Wang, X.-L. Jin, D.-Z. Kou, and J.-H. Chen. A Parallel Approach for the Generation of Unstructured Meshes with Billions of Elements on Distributed-Memory Supercomputers. *International Journal of Parallel Programming*, 45(3):680–710, June 2017. doi:10.1007/s10766-016-0452-3.

[46] H. Wendland. Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. *Advances in Computational Mathematics*, 4(1):389–396, Dec. 1995. doi:10.1007/BF02123482.

[47] O. C. Zienkiewicz, R. L. Taylor, and J. Z. Zhu. Chapter 10 - Incompressible Problems, Mixed Methods, and Other Procedures of Solution. In O. C. Zienkiewicz, R. L. Taylor, and J. Z. Zhu, editors, *The Finite Element Method: its Basis and Fundamentals (Seventh Edition)*, pages 315–359. Butterworth-Heinemann, Oxford, Jan. 2013. doi:10.1016/B978-1-85617-633-0.00010-1.