A ROM-based BDDC solver for unfitted p-FEM level-set-based two-dimensional lattice structures

Gonzalo Bonilla Moreno *[1], Giuliano Guarino †[1], and Pablo Antolin ‡[1]

[1]Institute of Mathematics, École polytechnique fédérale de Lausanne, Station 8, CH-1015 Lausanne, Switzerland

August 21, 2026

Abstract

We present a domain decomposition method for the fast simulation of large two-dimensional lattice structures described by level set functions. The method does not rely on homogenization or multiscale techniques, and therefore avoids their underlying assumptions such as scale separation and periodicity. Individual cells are defined through level set functions and mapped into physical space using arbitrary order mappings, which allows the creation of complex graded designs with varying geometries and topologies. The discretization is based on unfitted p-FEM, where each cell is approximated by a single high order element. This choice naturally handles the implicit geometric description and provides high accuracy with a moderate number of degrees of freedom. The solver is built on the Balanced Domain Decomposition by Constraints (BDDC) method, where each cell corresponds to one subdomain. To accelerate the assembly of the cell stiffness matrices, we combine a fast assembly technique that separates the contributions of the geometric mapping from the trimmed domain with a reduced order model (ROM) based on the matrix discrete empirical interpolation method (MDEIM). The ROM surrogate is trained offline and can be reused for any geometric mapping, restricting the expensive quadrature on cut elements to the training stage. A stabilization term is introduced to ensure the scalability of the solver when using the ROM approximation, at the cost of a small and controllable error. We validate the method through a series of numerical experiments and demonstrate its performance on a 2D problem with more than 17,000 cells of varying geometry, which is solved in approximately 30 seconds on a standard laptop. The number of solver iterations grows only mildly as the number of subdomains increases, provided the ratio between subdomain and mesh sizes is kept constant, consistent with the scalability properties of BDDC methods.

1 Introduction

Over the last few decades, the cost of additive manufacturing has decreased while the quality of their products has improved [1]. This, combined with the emergence of techniques for the fabrication of lattices at the nanoscale [2], has paved the way to the creation of architected materials with unprecedented strength to weight ratio, heat exchange capacity, among others. While porous geometries are abundant in nature (e.g., bones and bird beaks), artificial porous (heterogeneous) artifacts were very difficult to create prior to the additive manufacturing era. Now, their cell topology, geometry, and material can be tailored to achieve specific performance requirements [3]: significant weight savings while maintaining the stiffness and strength of homogeneous structures [4, 5, 6, 7, 8]. In addition, lattices can behave mechanically in very atypical ways (e.g., highly stretchable, auxetic [9]) and can be multi-functional, which makes them attractive also for applications such as energy absorption [10] and storage [11], vibration reduction, thermal management, etc. [12].

From a simulation perspective, lattice structures presenting a large number of cells are quite challenging [13]. A straightforward application of finite element methods is greedy in terms of computing resources

*gonzalo.bonilla.moreno@gmail.com
†giuliano.guarino@epfl.ch
‡pablo.antolin@epfl.ch

(both CPU time and memory). To circumvent this problem, different numerical approaches are applied in practice, namely: multiscale FEM [14, 15, 16], Generalized and Extended FEM [17, 18, 19], multilevel FEM (FE^2) [20, 21], or numerical homogenization [22, 23, 24, 25, 26], where the macroscopic behavior of the heterogeneous materials is characterized through numerical simulations on representative volume elements. However, to alleviate undesirable size effects [27, 28], all these techniques rely on i) the separation of scales, which is usually not the case in practice due to the achievable length scale of current 3D printers; and ii) periodic cells, an assumption which does not hold for graded lattices. In addition, macro-geometries are often slender geometries (plate, shell) that present just a few cells along the thickness direction. We refer the interested reader to [13] for an in-depth discussion of such approaches in the context of hierarchical metamaterials. In the case of truss-based lattices, an appealing alternative is the use of 3D beam models [29, 30, 31]. However, those methods are no longer suitable in the case of thick trusses, and cannot be applied to cellular lattices.

On the other hand, full scale 3D finite element simulations remain rare due to their high computational cost. They are typically limited to a few cells, or even a single one, and are used to estimate the macro-behavior properties of unit micro-structures or small samples [10, 32, 13, 33]. For instance, in [10] the authors performed a nonlinear simulation using Abaqus, including plasticity effects, with $7 \times 7 \times 7$ octet cells and, even though no computing times were reported, they required a cluster computer with 120 cores. Or [34, 35], where the authors performed a high-fidelity simulation of 24 CT-scanned octets (98M unknowns) using the finite cell method [36], which required 52 minutes using 1120 processors of a supercomputer.

Recent works aim to overcome these limitations using full scale finite element methods that leverage state of the art domain decomposition techniques combined with reduced order modeling (ROM) that exploits cell similarities to accelerate the solution. In [37, 38] the authors proposed an accelerated inexact FETI-DP preconditioner that allows to analyze linear problems with thousands of cells (millions of degrees of freedom) in just a few minutes, and with very low memory requirements, using an off-the-shelf laptop. This work has been recently extended in [39] to the case of nonlinear hyperelastic materials undergoing large deformations. In [40] similar ideas are exploited in a matrix-free multigrid solver able to simulate designs with hundreds of thousands of cells (billions of degrees of freedom) in just a few minutes using a few thousand processors from a supercomputer.

However, the application of this family of fast methods is limited to cell geometries simple enough to be described with conforming discretizations and parametric morphings, which is essential for exploiting cell similarities through ROM techniques. This prevents its applicability to the case of cellular like structures, with arbitrarily varying geometries and topologies that cannot be parameterized using such mappings. This is the case of, for instance, cellular structures [6, 41] or Triply Periodic Minimal Surfaces (TPMS) [42].

A natural way of handling such complex geometries is through level-set functions, which turn the geometric parameterization from explicit (parametric) to implicit. Such representations are particularly well suited for unfitted discretizations, in which the geometry is embedded in a background grid that serves as base for discretizing the PDE at hand [36, 43, 44, 45]. In this way, all cell geometries share the same discretization grid, regardless of their shape or topology.

This common grid structure enables the use of projection based ROM techniques. In particular, it allows the construction of surrogate models for the fast assembly of cell stiffness matrices using the matrix version of the discrete empirical interpolation method (MDEIM) [46, 47]. This bypasses the expensive tailored quadrature rules for cut elements required by unfitted methods, restricting their use to an offline training stage. The combination of MDEIM with unfitted methods has been previously addressed in [48, 49, 50].

In this work, we propose a novel fast domain decomposition method for the linear elasticity problem on two-dimensional lattice structures described by level-set functions. The method relies on unfitted p-FEM [51, 52, 53] (see also [54] for very high order unfitted discretizations of 3D elasticity; the motivation for high order discretizations will be addressed in Section 2) and exploits cell similarities through the ROM techniques described above. The domain decomposition solver is based on the Balanced Domain Decomposition by Constraints method (BDDC), introduced in [55, 56, 57] for FEM and subsequently adapted to various numerical methods, including isogeometrical analysis [58, 59] and unfitted methods [60]. . Here, the BDDC preconditioner is combined with the acceleration ideas first introduced in [37, 38] for two-dimensional lattice structures.

The remainder of the paper is organized as follows. Section 2 introduces the geometric modeling of single unit cells as well as their assembly into full lattice structures. The elasticity problem and its discretization

by means of unfitted p-FEM are also discussed. In Section 3 the BDDC domain decomposition method considered is introduced, alongside the required hypotheses and specific characteristics for this problem. The fast assembly of the cell stiffness matrices is discussed in Section 4. Finally, a series of numerical experiments that validate the considered hypotheses and assess the numerical performance of the proposed solver are presented in Section 5, while main conclusions are drawn in Section 6.

2 Modeling and Discretization

2.1 Geometric modeling

This section describes the geometric construction of the lattice cells investigated in this work. The geometry is obtained by juxtaposing cells, each constructed using a two-step procedure: first, mapping a parametric domain, and second, trimming it with the desired porous shape. To ensure the compatibility of the final assembled structure, the mapping and trimming processes must satisfy certain conditions, which are detailed in the remainder of this section.

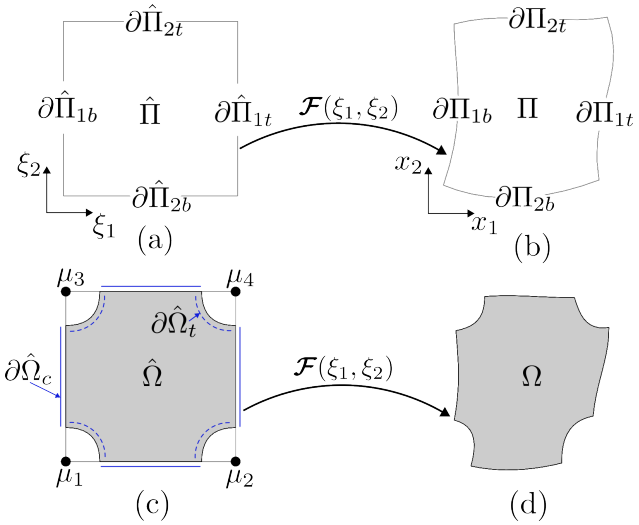


Figure 1: Construction of a single cell of the lattice structure. The untrimmed parametric domain $\hat{\Pi}$ in (a) is mapped through $\boldsymbol{\mathcal{F}}$ into the physical domain Π (b). The boundary of the untrimmed parametric domain is composed by four edges $\partial\hat{\Pi}_{1b}$, $\partial\hat{\Pi}_{1t}$, $\partial\hat{\Pi}_{2b}$, and $\partial\hat{\Pi}_{2t}$, which get mapped into the four edges of the untrimmed physical domain. The trimmed parametric domain $\hat{\Omega}$ is shown in (a). The threshold parameters μ_1, μ_2, μ_3, and μ_4 correspond to the nodes of first-order Lagrangian functions therefore the corners of $\hat{\Pi}$. The boundary of the trimmed domain is partitioned in its conformal part $\partial\hat{\Omega}_c$ and its trimmed part $\partial\hat{\Omega}_t$. Finally, in (d) it is shown the geometry of the trimmed physical domain Ω.

2.1.1 Mapping of a single cell

The starting point for the geometric definition of an individual cell is the untrimmed parametric domain, $\hat{\Pi}$, which is defined as the unit square:

$$\hat{\Pi} = [0,1] \times [0,1]. \tag{1}$$

This domain represents the set where the curvilinear coordinates (ξ_1, ξ_2) take values. These coordinates are used to explicitly define the map $\mathcal{F} : \hat{\Pi} \to \mathbb{R}^2$ as:

$$\boldsymbol{\mathcal{F}}(\xi_1,\xi_2) = \mathcal{F}_i(\xi_1,\xi_2)\,\boldsymbol{e}_i \quad \text{with} \quad i = 1,2, \tag{2}$$

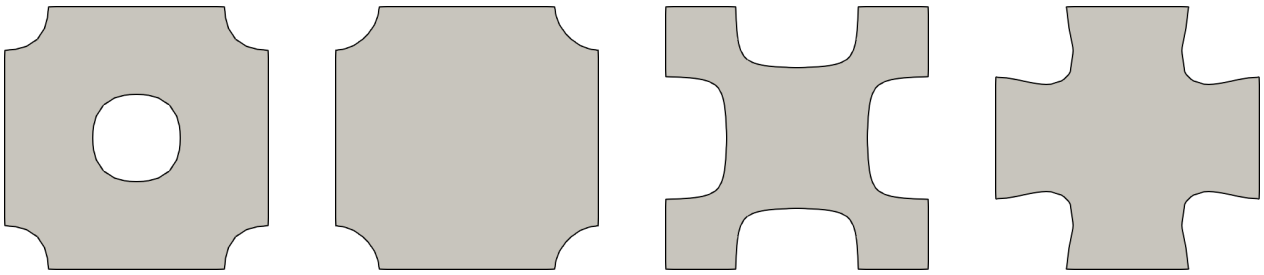
where $\boldsymbol{e}_i$ represents the vectors of the standard Euclidean basis. The four edges of the parametric domain are denoted as: the left vertical edge $\partial\hat{\Pi}_{1b} = \{(0,\xi)\}$, the right vertical edge $\partial\hat{\Pi}_{1t} = \{(1,\xi)\}$, the bottom

horizontal edge $\partial\hat{\Pi}_{2b} = \{(\xi, 0)\}$, and the top horizontal edge $\partial\hat{\Pi}_{2t} = \{(\xi, 1)\}$, where $\xi \in [0, 1]$. The untrimmed physical domain in $\mathbb{R}^2$ is the image of the unit square $\hat{\Pi}$ under this map:

$$\Pi = \boldsymbol{\mathcal{F}}(\hat{\Pi}). \tag{3}$$

Figure 1 shows an example of such construction. At this stage, no particular assumptions are made concerning the function $\boldsymbol{\mathcal{F}}$. However, a discussion regarding its required regularity is provided in the remainder of this section.

2.1.2 Trimming of a single cell



(a) Schwarz Diamond (b) Schwarz Primitive (c) Schoen FRD (d) Schoen IWP

Figure 2: 2D TPMS corresponding to the level-set functions in Table 1.

Table 1: Common TPMS in 2D. The level-sets have been adapted from their 3D counterparts by evaluation at $\xi_3 = 0$. The sole exception is the Schwarz Primitive that is evaluated at $\xi_3 = 1/2$ to ensure the geometry remains connected. For the same reason, the sign of the Schoen FRD is the opposite of its standard definition and the standard threshold parameter for the Schwarz Diamond is taken as 0.1.

Name	**level-set** $\phi_0(\xi_1, \xi_2)$
Schwarz Diamond	$\cos(2\pi\xi_1)\cos(2\pi\xi_2)$
Schwarz Primitive	$\cos(2\pi\xi_1) + \cos(2\pi\xi_2) - 1$
Schoen FRD	$\cos(4\pi\xi_1)\cos(4\pi\xi_2) + \cos(4\pi\xi_2) + \cos(4\pi\xi_1) - 4\cos(2\pi\xi_1)\cos(2\pi\xi_2)$
Schoen IWP	$2\big(\cos(2\pi\xi_1)\cos(2\pi\xi_2) + \cos(2\pi\xi_2) + \cos(2\pi\xi_1)\big) - \cos(4\pi\xi_1) - \cos(4\pi\xi_2) - 1$

The mapping introduced above is combined with an immersed boundary approach to define the active portion of the cell. Specifically, the trimmed parametric domain is defined as:

$$\hat{\Omega} = \left\{(\xi_1, \xi_2) \in \hat{\Pi} : \phi_0\left(\xi_1, \xi_2\right) < \mu\left(\xi_1, \xi_2\right)\right\} , \tag{4}$$

where ϕ_0 and μ are the *level-set* and *threshold* functions, respectively. Or equivalently, by introducing the function $\phi\left(\xi_1, \xi_2\right) = \phi_0\left(\xi_1, \xi_2\right) - \mu\left(\xi_1, \xi_2\right)$ as

$$\hat{\Omega} = \left\{(\xi_1, \xi_2) \in \hat{\Pi} : \phi\left(\xi_1, \xi_2\right) < 0\right\} . \tag{5}$$

While ϕ_0 can be any arbitrary implicit function, for illustrative purposes, in this work it is selected from the class of triply periodic minimal surfaces (TPMS) adapted for 2D applications, which does not constitute a limitation for the proposed method. Common examples are defined in Table 1 and depicted in Figure 2.

Typically, the threshold is kept uniform, therefore determining which isoline of ϕ_0 is adopted as domain boundary. However, to introduce more flexibility in the design of the cells, this work allows μ to vary linearly as follows:

$$\mu\left(\xi_1, \xi_2\right) = \sum_{i=1}^{4} \ell_i\left(\xi_1, \xi_2\right) \mu_i \ , \tag{6}$$

where ℓ_i are the classical first-degree Lagrangian shape functions, and the threshold parameters μ_i correspond to the nodal values at the bottom-left, bottom-right, top-left, and top-right corners of $\hat{\Pi}$. As for its untrimmed counterpart, the trimmed physical domain is obtained through the map:

$$\Omega = \boldsymbol{\mathcal{F}}(\hat{\Omega}) \, . \tag{7}$$

The boundary of the parametric domain is identified as two separate portions: the conformal portion lying on the untrimmed parametric domain boundary, and the trimmed portion that is internal to the domain:

$$\partial\hat{\Omega}_c = \left\{ (\xi_1, \xi_2) \in \partial\hat{\Pi} : \phi_0 \left(\xi_1, \xi_2\right) < \mu \left(\xi_1, \xi_2\right) \right\} \, , \tag{8a}$$

$$\partial\hat{\Omega}_t = \left\{ (\xi_1, \xi_2) \in \text{int}(\hat{\Pi}) : \phi_0 \left(\xi_1, \xi_2\right) = \mu \left(\xi_1, \xi_2\right) \right\} \, , \tag{8b}$$

where $\text{int}(\bullet)$ denotes the internal part of the domain $\bullet$, such that $\partial\hat{\Omega} = \partial\hat{\Omega}_c \cup \partial\hat{\Omega}_t$ and $\partial\hat{\Omega}_c \cap \partial\hat{\Omega}_t = \emptyset$. In Figure 1, an example of a trimmed domain is shown in parametric and physical coordinates and its correspondent threshold parameters.

2.1.3 Assembled lattice structure

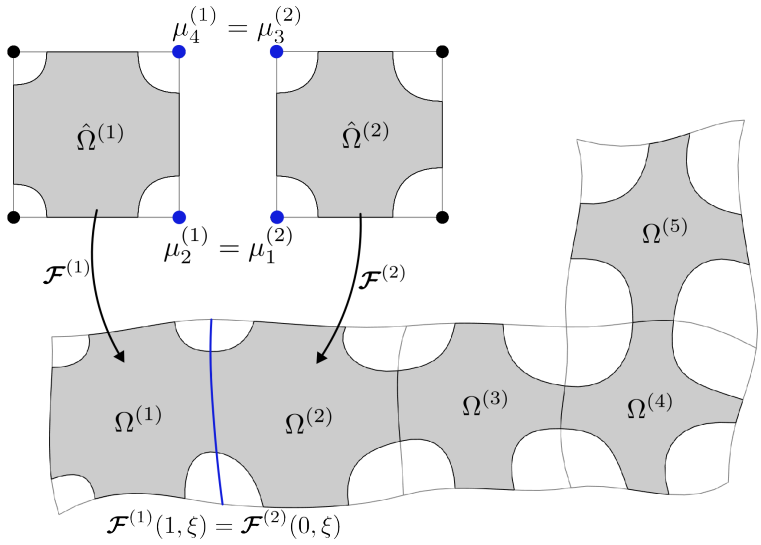


Figure 3: A lattice structure is constructed by juxtaposing six cells in a compatible way. The compatibility conditions are illustrated for cells (1) and (2). The map of the common edge has to be the same for both cells, meaning $\boldsymbol{\mathcal{F}}^{(1)}(1,\xi) = \boldsymbol{\mathcal{F}}^{(2)}(0,\xi)$ with $\xi \in [0,1]$. To guarantee that the threshold function varies in the same way along the common edge, the following equivalences are enforced between threshold parameters: $\mu_2^{(1)} = \mu_1^{(2)}$, and $\mu_4^{(1)} = \mu_3^{(2)}$.

The discussion so far has focused on the geometric definition of a single cell. To describe a complete structure composed of multiple cells, we now introduce a superscript (i_c) to distinguish all quantities associated with a specific cell, where $i_c \in \{1, \ldots, n_c\}$ is the cell index and n_c is the total number of cells. Therefore, the untrimmed and trimmed physical domains of the overall structure are defined, respectively, as:

$$\Pi = \text{int}\left(\bigcup_{i_c=1}^{n_c} \bar{\Pi}^{(i_c)} \right), \quad \text{and} \quad \Omega = \text{int}\left(\bigcup_{i_c=1}^{n_c} \bar{\Omega}^{(i_c)} \right) \, , \tag{9}$$

where $\bar{\bullet}$ denotes the closure of the domain $\bullet$. While this procedure can generate complex topologies and is not restricted to tensor-product grids, the cells must satisfy certain compatibility conditions at both the untrimmed and trimmed levels.

First, for untrimmed compatibility, the physical mapping of connected edges must coincide. If the i_{c_1}-th and i_{c_2}-th cells are joined at their respective edges $\partial\hat{\Pi}_i^{(i_{c_1})}$ and $\partial\hat{\Pi}_j^{(i_{c_2})}$, then:

$$\boldsymbol{\mathcal{F}}^{(i_{c_1})}\left(\partial\hat{\Pi}_i^{(i_{c_1})}\right) = \boldsymbol{\mathcal{F}}^{(i_{c_2})}\left(\partial\hat{\Pi}_j^{(i_{c_2})}\right). \tag{10}$$

where $i, j \in \{1b, 1t, 2b, 2t\}$.

Second, for trimmed compatibility, the trimming function ϕ must be continuous across the common interface

$$\phi^{(i_{c_1})}(\xi) = \phi^{(i_{c_2})}(\xi), \tag{11}$$

where $\xi \in [0, 1]$ is an auxiliary curvilinear coordinate that parameterizes the interface. This condition, in turn, is satisfied if both the level-set and threshold functions are independently equal: $\phi_0^{(i_{c_1})}(\xi) = \phi_0^{(i_{c_2})}(\xi)$ and $\mu^{(i_{c_1})}(\xi) = \mu^{(i_{c_2})}(\xi)$. In the particular case of TPMS geometries, the first condition (ϕ_0) is automatically satisfied by their intrinsic periodicity, provided that the edge pairs (i, j) in Equation (10) are restricted to the admissible set: $(1b, 1t), (1t, 1b), (2b, 2t), (2t, 2b)$. The second equality ($\mu$) is satisfied by simply enforcing identical threshold parameters at the common nodes of neighboring cells.

To illustrate, in Figure 3, the two adjacent cells $\Pi^{(1)}$ and $\Pi^{(2)}$ are connected at their edges $\partial\Pi_{1t}^{(1)}$ and $\partial\Pi_{1b}^{(2)}$. Compatibility of the untrimmed interface is ensured because the maps are chosen such that the following relationship holds: $\boldsymbol{\mathcal{F}}^{(1)}(1, \xi) = \boldsymbol{\mathcal{F}}^{(2)}(0, \xi)$ with $\xi \in [0, 1]$. Furthermore, compatibility of the trimmed interface is achieved by imposing $\mu_2^{(1)} = \mu_1^{(2)}$, and $\mu_4^{(1)} = \mu_3^{(2)}$.

Notably, a spline surface can be easily adopted as the untrimmed physical domain, as it naturally satisfies the compatibility condition in Equation (10). The only preprocessing required is a Bézier extraction for each of its elements, which in turn defines the mapping for each cell in the lattice structure.

2.2 Model Problem and Discretization

With the geometric framework for the lattice structure established, we now turn to the mechanical problem defined on the trimmed physical domain Ω. Specifically, we consider the linear elasticity equations, which govern the static response of the structure under prescribed loads and boundary conditions, and describe their discretization using a high-order unfitted finite element method.

2.2.1 Linear Elasticity

The variational statement for two-dimensional linear elasticity reads: find the displacement field $\boldsymbol{u} \in [\mathcal{H}^1]^2_{\bar{\boldsymbol{u}}}$ such that

$$\mathfrak{L}_{int}(\boldsymbol{v}, \boldsymbol{u}) = \mathfrak{L}_{ext}(\boldsymbol{v}) \,, \qquad \forall \boldsymbol{v} \in [\mathcal{H}^1]^2_{\boldsymbol{0}} \,, \tag{12}$$

where $[\mathcal{H}^1]^2_{\bar{\boldsymbol{u}}}$ and $[\mathcal{H}^1]^2_{\boldsymbol{0}}$ represent the subspaces of $[\mathcal{H}^1]^2$ for which Dirichlet boundary conditions are satisfied, in the first case, and where the test function $\boldsymbol{v}$ is null on the Dirichlet boundary, in the second one. It is further assumed that only pure Dirichlet or Neumann boundary conditions are considered, meaning that mixed boundary conditions, where Dirichlet is applied to one component of the displacement and Neumann to another, are excluded for simplicity. We note that supporting such conditions is a straightforward extension of the present framework.

The bilinear and the linear forms in Equation (12) are also referred to as virtual work of the internal and external forces, respectively, and are defined as:

$$\mathfrak{L}_{int}(\boldsymbol{v}, \boldsymbol{u}) = \sum_{i_c=1}^{n_c} \mathfrak{L}_{int}^{(i_c)}(\boldsymbol{v}, \boldsymbol{u}), \tag{13a}$$

$$\mathfrak{L}_{ext}(\boldsymbol{v}) = \sum_{i_c=1}^{n_c} \mathfrak{L}_{ext}^{(i_c)}(\boldsymbol{v}) \,, \tag{13b}$$

where the virtual work of the internal and external forces for the i_c-th cell are defined as:

$$\mathfrak{L}_{int}^{(i_c)}(\boldsymbol{v},\boldsymbol{u}) = \int_{\Omega^{(i_c)}} \boldsymbol{\sigma}(\boldsymbol{u}) : \boldsymbol{\varepsilon}(\boldsymbol{v})\,\mathrm{d}\Omega\,, \tag{14a}$$

$$\mathfrak{L}_{ext}^{(i_c)}(\boldsymbol{v}) = \int_{\Omega^{(i_c)}} \boldsymbol{v}\cdot\boldsymbol{b}\,\mathrm{d}\Omega + \int_{\partial\Omega_N^{(i_c)}} \boldsymbol{v}\cdot\boldsymbol{t}\,\mathrm{d}\partial\Omega\,, \tag{14b}$$

where $\boldsymbol{b}$ and $\boldsymbol{t}$ are the distributed domain force and boundary traction, respectively. The components of the strain tensor $\boldsymbol{\varepsilon}$ are:

$$\varepsilon_{ij}(\boldsymbol{u}) = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)\,, \tag{15}$$

whereas the components of the stress tensor are obtained through the constitutive relationship:

$$\sigma_{ij} = C_{ijkl}\,\varepsilon_{kl}\,, \tag{16}$$

being C_{ijkl} the elasticity tensor that for isotropic materials reads as:

$$C_{ijkl} = \lambda\,\delta_{ij}\delta_{kl} + \mu\,(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}), \tag{17}$$

where λ and μ are the Lamé parameters of the material. The linear elasticity variational statement can be reformulated through an adequate transformation directly in the parametric domain of the cell, which will be useful later in this article for the fast assembly procedure. To do so, one shall notice that

$$\frac{\partial u_i}{\partial x_j} = \frac{\partial u_i}{\partial \xi_k}\frac{\partial \xi_k}{\partial x_j}\,, \quad \mathrm{d}\Omega = j_\Omega \mathrm{d}\hat{\Omega}\,, \quad \mathrm{d}\partial\Omega = j_{\partial\Omega}\mathrm{d}\partial\hat{\Omega}\,, \quad \text{and} \quad \varepsilon_{ij}C_{ijkl}\varepsilon_{kl} = \frac{\partial u_i}{\partial x_j}C_{ijkl}\frac{\partial u_k}{\partial x_l} \tag{18}$$

where the last equivalence comes from the symmetries of the elasticity tensor, j_Ω is the determinant of the Jacobian of the map in Equation (2), and $j_{\partial\Omega}$ is the curve Jacobian associated with the boundary. Substituting into Equation (14), the virtual works of the internal and external forces become:

$$\mathfrak{L}_{int}^{(i_c)}(\boldsymbol{v},\boldsymbol{u}) = \int_{\hat{\Omega}^{(i_c)}} \frac{\partial u_i}{\partial \xi_j}\hat{C}_{ijkl}\frac{\partial u_k}{\partial \xi_l}\,\mathrm{d}\hat{\Omega}\,, \tag{19a}$$

$$\mathfrak{L}_{ext}^{(i_c)}(\boldsymbol{v}) = \int_{\hat{\Omega}^{(i_c)}} \boldsymbol{v}\cdot\hat{\boldsymbol{b}}\,\mathrm{d}\hat{\Omega} + \int_{\partial\hat{\Omega}_N^{(i_c)}} \boldsymbol{v}\cdot\hat{\boldsymbol{t}}\,\mathrm{d}\partial\hat{\Omega}\,, \tag{19b}$$

where the elasticity tensor and external forces have been modified as:

$$\hat{C}_{ijkl} = j_\Omega\,\frac{\partial \xi_m}{\partial x_j}C_{imkn}\frac{\partial \xi_n}{\partial x_l}\,, \tag{20a}$$

$$\hat{\boldsymbol{b}} = j_\Omega\,\boldsymbol{b}\,, \tag{20b}$$

$$\hat{\boldsymbol{t}} = j_{\partial\Omega}\,\boldsymbol{t}\,. \tag{20c}$$

It shall be mentioned that in the previous equations, the term relative to the differential geometry of the map should be enriched with a superscript (i_c) since this is specific to the cell. However, to enhance readability and simplify the notation the superscript has not been used here.

2.2.2 p-FEM finite element method

The approximation space employed in this work is based on Lagrange polynomials defined on Gauss-Lobatto-Legendre (GLL) nodes, constructed over the untrimmed parametric domain $\hat{\Pi}^{(i_c)}$. Accordingly, the polynomials are expressed as functions of the curvilinear coordinates ξ_1 and ξ_2. Figure 4 illustrates the univariate polynomials of degree 6 in both directions, along with the associated tensor-product GLL nodes. The corresponding bivariate polynomials are defined as:

$$\mathscr{B}_{ij}(\xi_1,\xi_2) = \mathscr{b}_i(\xi_1)\cdot\mathscr{b}_j(\xi_2), \quad \text{with} \quad i,j\in\{0,\dots,p\}, \tag{21}$$

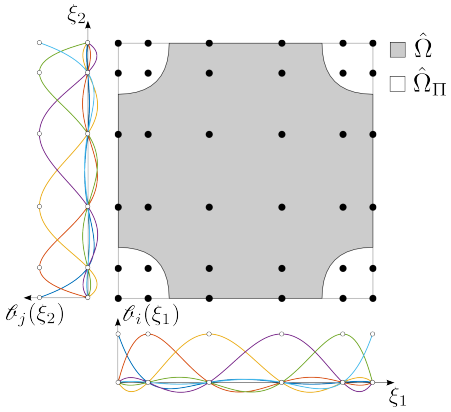


Figure 4: Position of the Gauss-Lobatto-Legendre nodes of the Lagrangian basis superimposed to the trimmed parametric domain $\hat{\Omega}$ and its complement $\hat{\Omega}_\Pi$. The associated univariate polynomials in ξ_1 and ξ_2 are also shown.

where $\mathscr{b}_i(\xi_1)$ and $\mathscr{b}_j(\xi_2)$ denote the i-th and j-th univariate GLL polynomials in the ξ_1 and ξ_2 directions, respectively, and p represents the degree of the approximation space. Formally, the approximation space for the i_c-th cell is given by

$$\mathcal{S}_h = \text{span}\{\mathscr{B}_{ij} \circ \boldsymbol{\mathcal{F}}^{-1} : i \in \{0, \dots, p\},\ j \in \{0, \dots, p\}\}, \tag{22}$$

It is worth mentioning that in the p-FEM context, the mesh is not refined beyond the level of the individual cell, thereby making the approximation element and the geometric cell equivalent. Despite this seemingly coarse discretization of the individual cell, the high polynomial order ensures substantial approximation accuracy as it will be shown in Section 5.

Moreover, unlike other high-order bases, the GLL basis features nodes located at the domain boundaries, as illustrated in Figure 4 for $p = 6$. Consequently, degrees of freedom at cell interfaces could, in principle, be strongly enforced. However, as will be detailed in Section 3, the domain decomposition strategy adopted in this work does not impose a strong coupling of all interface degrees of freedom, but still exploits the coincidence of their position.

2.2.3 Unfitted FEM

The polynomial basis and the location of the nodes are initially defined on the untrimmed parametric domain $\hat{\Pi}^{(i_c)}$. After the trimming operation, however, the support of each basis function is restricted to its active portion corresponding to the trimmed parametric domain of the cell, $\hat{\Omega}^{(i_c)}$. Importantly, regardless of the shape of the trimming, each basis function remains active, even if the associated node lies outside the active domain, leading to fully dense stiffness matrices.

One might argue that, for a comparable approximation accuracy, using lower-degree polynomials with mesh refinement could provide the advantage of increased sparsity. Nevertheless, the property that the basis functions remain active irrespective of the trimming geometry is particularly advantageous for building a surrogate model of the stiffness, as discussed in Section 4, and is a primary motivation for the adoption of p-FEM elements.

Unfitted discretizations naturally introduce three main challenges: quadrature, application of Dirichlet boundary conditions, and conditioning.

Quadrature algorithm Quadrature over the complex trimmed domains is handled by the recently released QUGaR library [61], which provides a high-order quadrature algorithm for level-set-defined geometries within the FEniCSx framework [62], building on the algoim library [63, 64, 65]. The algorithm recasts the integral over the implicitly defined domain as a recursive sequence of one-dimensional integrations, thereby avoiding any explicit reconstruction of the trimmed boundary. Once an integration direction has been selected, the height function induced by the level-set, that is, the location of its roots along that direction, provides the

integration bounds in that coordinate. The remaining base direction is then partitioned into subintervals delimited by the critical points of the level-set, where the height function loses smoothness or the number of roots changes. In the interior of each subinterval the integrand is infinitely smooth, so that a standard quadrature rule (e.g. Gauss-Legendre or tanh-sinh), applied in a tensor-product fashion, evaluates the integral to high accuracy. The procedure is illustrated in Figure 5 for one of the level-set functions listed in Table 1. This methodology has proven to yield high-order accuracy and spectral convergence of the integration error. Consequently, the consistency error associated with the numerical integration can be safely assumed to be negligible. Furthermore, since changes in the threshold functions alter the underlying geometry, a distinct set of quadrature nodes and weights must be generated for each set of threshold parameters.

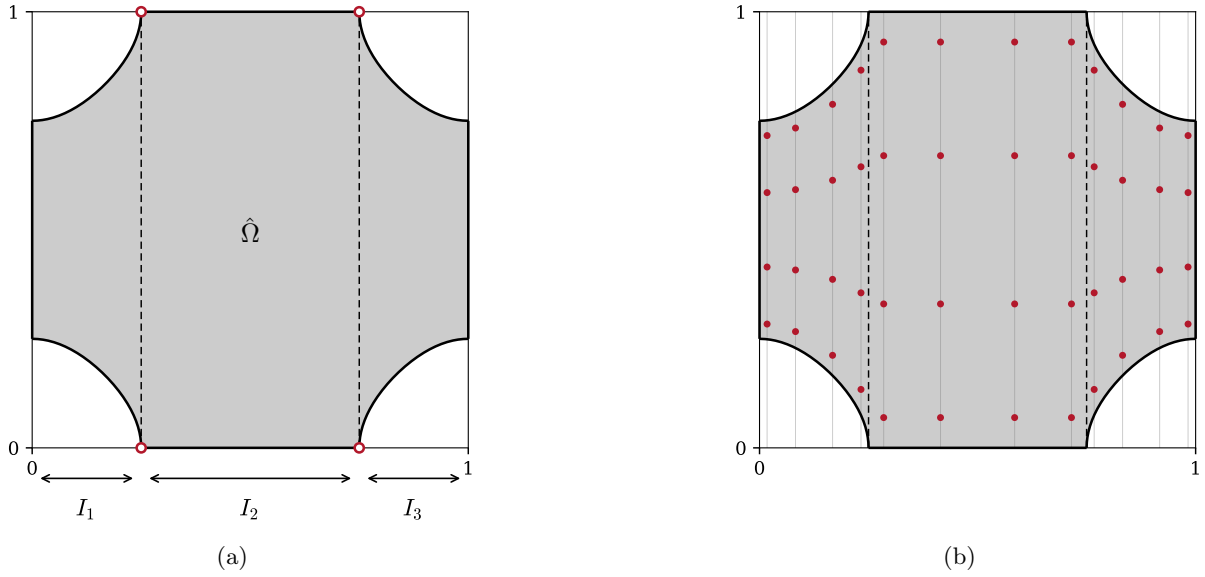


Figure 5: Height-function quadrature of QUGaR/algoim over the active domain $\hat{\Omega}$ of a Schwarz Primitive cell, with ξ_2 selected as the integration direction. (a) The base direction ξ_1 is split at the critical points of the level-set (red circles) into subintervals I_1, I_2, I_3 over which the height function is smooth. (b) A tensor-product Gauss rule then places the quadrature points within each subinterval, clustering them in the active domain.

Application of boundary conditions We restrict our formulation to boundary conditions applied on the conformal portions of the boundary, $\partial\hat{\Omega}_c$. Dirichlet conditions on $\partial\hat{\Omega}_c$ can be applied directly in a strong sense by acting on the degrees of freedom lying on $\partial\hat{\Pi}$. Conversely, we do not consider Dirichlet boundary conditions on trimmed boundary portions, $\partial\hat{\Omega}_t$. This is a deliberate choice since, in an unfitted setting, the degrees of freedom do not generally align with the cut boundary. As such, enforcing Dirichlet conditions there requires weak imposition techniques. Integrating such methods into the BDDC preconditioner is mathematically non-trivial and is deferred to future work. Non-homogeneous Neumann conditions are supported on the conformal portions of the boundary, including their active parts. On trimmed boundary portions, such conditions pose no mathematical difficulty, as they only involve an integral over the cut boundary, and they are indeed employed in the single-cell study of Section 5.1.1; however, they are not covered by the ROM acceleration of Section 4 and are therefore not considered in the remainder of this work. While boundary conditions on trimmed boundaries might be relevant for local fixtures, local loads, or contact zones, they are uncommon for the interior surfaces of lattice structures, which are typically left unloaded.

Stabilization Depending on the configuration of the trimming, poorly conditioned stiffness matrices may arise even with a spectral basis. To address this issue, we employ the α-stabilization proposed in [66]. This

non-consistent term is defined as:

$$\mathfrak{L}_{sta}(\boldsymbol{v},\boldsymbol{u}) = \rho \sum_{i_c=1}^{n_c} \int_{\hat{\Omega}_{\Pi}^{(i_c)}} \frac{\partial u_i}{\partial \xi_j} \hat{C}_{ijkl} \frac{\partial u_k}{\partial \xi_l} \, \mathrm{d}\hat{\Omega}, \tag{23}$$

where ρ is a constant that balances the condition number with the consistency error. The stabilization is applied over the complement of the active domain, $\hat{\Omega}_{\Pi}^{(i_c)} = \hat{\Pi}^{(i_c)} \setminus \hat{\Omega}^{(i_c)}$. Physically, this stabilization can be interpreted as immersing the domain in a softer material, which, when ρ is small, does not significantly contribute to the overall stiffness of the structure.

3 Domain Decomposition Method

This section presents the Balancing Domain Decomposition by Constraints (BDDC) method [67, 55], a non-overlapping domain decomposition technique used to solve the linear system arising from the assembly of the lattice structures. In the present framework, each cell is associated with a corresponding subdomain in the domain decomposition setting. The global Schur complement system, obtained after condensing the internal degrees of freedom (DoFs) of each subdomain, is solved iteratively using the preconditioned conjugate gradient (PCG) method. The BDDC preconditioner decomposes the correction of the residual into two complementary parts: (i) independent local Neumann-type solves in each subdomain, made invertible by constraining the correction to vanish at a selected set of coarse DoFs, and (ii) a global coarse solve that balances the local corrections through a basis spanning the coarse DoF space. Since the resulting correction is only guaranteed to be continuous at the coarse DoFs, a weighted averaging step using a discrete partition of unity restores full continuity across the entire interface skeleton.

Two important aspects of the problem formulation must be noted already at this stage. First, for the 2D elasticity problems considered, there is no one-to-one correspondence between degrees of freedom and nodes; each node is associated with two DoFs, corresponding to the components of the displacement vector. Second, eventual Dirichlet boundary conditions are enforced strongly along entire edges even though only part of it is effectively active. Furthermore, only pure Dirichlet and Neumann boundary conditions are considered, while mixed boundary conditions are neglected.

To improve readability, quantities associated with an individual cell will be denoted by a superscript (i), which differs from the notation used in the previous section. Throughout this section, the following notational convention is adopted: a tilde denotes the redundant (vertically stacked) version of a quantity, obtained by collecting contributions from all cells, e.g., $\tilde{u}$; the absence of a tilde denotes the assembled (non-redundant) counterpart, e.g., u; and a superscript (i) denotes the local version associated with i-th cell, e.g., $u^{(i)}$. The same convention applies to operators such as $\tilde{\mathcal{R}}_U$ and $\mathcal{R}_U^{(i)}$, and to DoF counts such as $\tilde{n}_U$, n_U, and $n_U^{(i)}$. Furthermore, the following *vertical stacking* and *diagonal stacking* operators are introduced here and adopted throughout this section:

$$\text{vstack}(v_1 \ldots, v_n) = \begin{bmatrix} v_1 \\ v_2 \\ \\ \vdots \\ v_n \end{bmatrix}, \quad \text{and} \quad \text{blockdiag}(M_1 \ldots, M_n) = \begin{bmatrix} M_1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & M_n \end{bmatrix}.$$

3.1 Preliminary decompositions

Figure 6a illustrates a single cell together with the nodes associated with the DoFs of its spectral approximation space. These DoFs are grouped into three distinct vectors: $u_D^{(i)}$, $u^{(i)}$, and $u_I^{(i)}$. The first vector, $u_D^{(i)}$ (triangles), contains the local Dirichlet DoFs, i.e., those at which Dirichlet boundary conditions are imposed in strong form. The second vector, $u^{(i)}$ (squares), contains the local skeleton DoFs, defined as those that do not belong to $u_D^{(i)}$ and lie on an interface with another cell. Finally, the third vector, $u_I^{(i)}$ (solid circles), contains the remaining internal DoFs. The sizes of such vectors are denoted as $n_D^{(i)}$, $n_U^{(i)}$, and $n_I^{(i)}$, respectively. For the assembled structure, the internal and skeleton DoFs are stacked into their global counterparts,

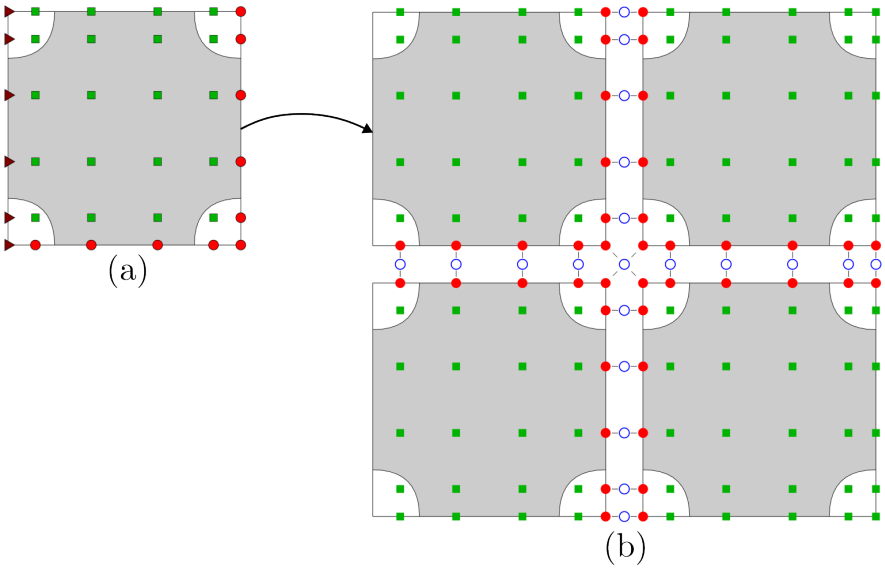


Figure 6: Classification of the DoFs associated with a single cell (a) and with a two-by-two cells structure (b). For the single cell in subfigure (a): ▶ denotes the local DoFs associated with the Dirichlet boundary conditions collected in $u_D^{(i)}$; ● denotes the local DoFs whose associated nodes lie on the internal skeleton, collected in $u^{(i)}$; ■ denotes the local internal DoFs collected in $u_I^{(i)}$. Regarding the entire structure in subfigure (b): ■ denotes the global internal DoFs collected in $\tilde{u}_I$; ○ denotes the non-redundant skeleton DoFs collected in u; ● denotes the redundant global skeleton DoFs collected in $\tilde{u}$. The relationship between $\tilde{u}$ and u, visualized by dashed lines in (b), is enforced by the equation $\tilde{u} = \tilde{\mathcal{R}}_U u$.

denoted by $\tilde{u}_I$ and $\tilde{u}$, respectively, as follows:

$$\tilde{u}_I = \text{vstack}\left(u_I^{(1)}, \dots, u_I^{(n_c)}\right), \quad \text{and} \quad \tilde{u} = \text{vstack}\left(u^{(1)}, \dots, u^{(n_c)}\right). \tag{24}$$

It is worth recalling that n_c denotes the number of cells and coincides with the number of subdomains in the decomposition. The dimensions of these vectors are given by $\tilde{n}_I = \sum_{i=1}^{n_c} n_I^{(i)}$ and $\tilde{n}_U = \sum_{i=1}^{n_c} n_U^{(i)}$. While the local internal DoFs $u_I^{(i)}$ vectors are mutually independent, the skeleton DoFs are not. Consequently, the vector $\tilde{u}$ contains redundant DoFs. In contrast, we denote by u the vector of independent non-redundant skeleton DoFs, and by n_U its dimension. The relationship between $\tilde{n}_U$ and n_U depends on the specific geometry of the mesh. After introducing a global numbering of the non-redundant skeleton DoFs, we introduce the extension/restriction operators $\mathcal{R}_U^{(i)} \in \mathbb{R}^{n_U^{(i)} \times n_U}$ associated to the relation

$$u^{(i)} = \mathcal{R}_U^{(i)} u, \tag{25}$$

that extract the vector of local skeleton DoFs from its global non-redundant version. $\mathcal{R}_U^{(i)}$ is therefore composed of zeros and ones accordingly to the global and local DoFs orderings. The corresponding global expression reads

$$\tilde{u} = \tilde{\mathcal{R}}_U u, \tag{26}$$

with $\tilde{\mathcal{R}}_U \in \mathbb{R}^{\tilde{n}_U \times n_U}$

$$\tilde{\mathcal{R}}_U = \text{vstack}\left(\mathcal{R}_U^{(1)}, \dots, \mathcal{R}_U^{(n_c)}\right). \tag{27}$$

3.2 Assembly of the stiffness matrix

The standard assembly of the bilinear forms introduced in Section 2.2, for the i-th cell, yields a local stiffness matrix that can be partitioned into submatrices denoted by $K_{AB}^{(i)}$. Here, the subscripts A and B take values in $\{I, U, D\}$, depending on whether the corresponding rows and columns are associated with $u_I^{(i)}$, $u^{(i)}$, and

$u_D^{(i)}$, respectively. Similarly, the assembly of the linear forms in Section 2.2 yields the local external force vector, which is partitioned into $f_I^{*(i)}$ and $f_U^{*(i)}$. These local force vectors are further adjusted by condensing the Dirichlet DoFs as

$$f_I^{(i)} = f_I^{*(i)} - K_{ID}^{(i)} u_D^{(i)} \,, \tag{28}$$

$$f_U^{(i)} = f_U^{*(i)} - K_{UD}^{(i)} u_D^{(i)} \,. \tag{29}$$

Assembling the local stiffness matrices and force vectors over the entire structure, while preserving the separation between internal and skeleton DoFs in the global ordering, yields the following linear system:

$$\begin{bmatrix} \tilde{K}_{II} & \tilde{K}_{IU}\tilde{\mathcal{R}}_U \\ \tilde{\mathcal{R}}_U^\top \tilde{K}_{IU} & \tilde{\mathcal{R}}_U^\top \tilde{K}_{UU} \tilde{\mathcal{R}}_U \end{bmatrix} \begin{bmatrix} \tilde{u}_I \\ u \end{bmatrix} = \begin{bmatrix} \tilde{f}_I \\ \tilde{\mathcal{R}}_U^\top \tilde{f}_U \end{bmatrix} \tag{30}$$

Here,

$$\tilde{K}_{AB} = \text{blockdiag}\left(K_{AB}^{(1)}, \ldots, K_{AB}^{(n_c)}\right) \,, \quad \text{and} \quad \tilde{f}_A = \text{vstack}\left(f_A^{(1)}, \ldots, f_A^{(n_c)}\right) \,, \tag{31}$$

where, the subscripts A and B take values in $\{I, U\}$. Then, condensing the internal DoFs within each cell leads to the local Schur complement $S^{(i)}$ and the corresponding condensed force vector $f^{(i)}$, defined by

$$S^{(i)} = K_{UU}^{(i)} - K_{UI}^{(i)} K_{II}^{(i)-1} K_{IU}^{(i)} \,,$$

$$f^{(i)} = f_U^{(i)} - K_{UI}^{(i)} K_{II}^{(i)-1} f_I^{(i)} \,.$$

Note that $K_{II}^{(i)}$ is invertible, since neither skeleton nor external boundary DoFs are included in its definition. In this work, the local Schur complements are assembled explicitly, although the inverse $K_{II}^{(i)-1}$ is never formed. Instead, the action of $K_{II}^{(i)-1}$ on $K_{IU}^{(i)}$ is computed by solving linear systems of the form $K_{II}^{(i)} X = K_{IU}^{(i)}$, i.e., by applying a local solve to each column of $K_{IU}^{(i)}$. The resulting global system reads

$$\tilde{\mathcal{R}}_U^\top \tilde{S} \tilde{\mathcal{R}}_U \, u = \tilde{\mathcal{R}}_U^\top \tilde{f}$$

where

$$\tilde{S} = \text{blockdiag}\left(S^{(1)}, \ldots, S^{(n_c)}\right) \,, \quad \text{and} \quad \tilde{f} = \text{vstack}\left(f^{(1)}, \ldots, f^{(n_c)}\right) \,.$$

Equivalently,

$$Su = f \tag{32}$$

with $S = \tilde{\mathcal{R}}_U^\top \tilde{S} \tilde{\mathcal{R}}_U$, and $f = \tilde{\mathcal{R}}_U^\top \tilde{f}$, where S is the global assembled Schur complement.

3.3 The BDDC method

In the BDDC method [60, 67], the system in Equation (32) is solved using an iterative method, here the preconditioned conjugate gradient (PCG) method, where the construction of the preconditioner S_{BDDC}^{-1} is described in the following subsection. Although the details are omitted here for brevity, the PCG algorithm requires the multiplication of search directions by S and of residuals by S_{BDDC}^{-1}.

In this contribution, neither of these quantities are explicitly assembled. Only local Schur complements $S^{(i)}$ are assembled and stored. Therefore, the action of S on a vector is performed through sequences of matrix-vector multiplications that can be efficiently distributed in parallel. In contrast, the application of S_{BDDC}^{-1} also involves local solves, which are likewise readily parallelizable, with only one coarse global solve performed using a parallel Cholesky factorization (although alternative solvers for the coarse correction could also be employed).

The BDDC preconditioner consists of two components: a coarse correction, which addresses errors associated with global low-frequency modes affecting the entire structure, and a fine correction, which targets errors in local high-frequency behaviors. To properly separate these two contributions, a set of coarse DoFs is selected.

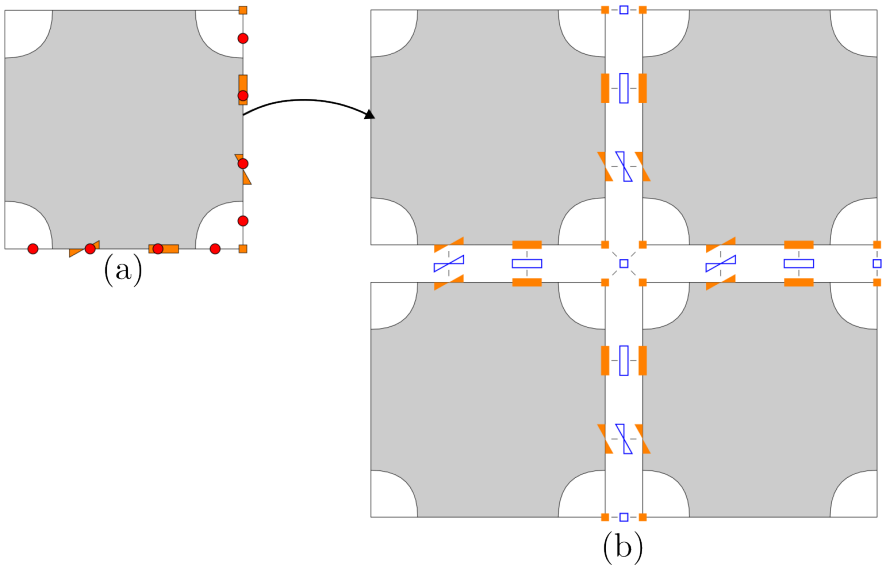


Figure 7: Classification of the coarse DoFs associated with a single cell (a) and with a two-by-two cells structure (b). For the single cell in subfigure (a): ● local DoFs of the internal skeleton used to compute edge averages and moments; ■ local DoFs at crosspoints that are also coarse DoFs; ▬ local averages of DoFs on edges lying on the internal skeleton; ◢ local moments of DoFs on edges lying on the internal skeleton. In particular, these last three categories are collected in the vector of local coarse DoFs $c^{(i)}$. Regarding the entire structure in subfigure (b): ■ global redundant DoFs at crosspoints that are also coarse DoFs; ▬ global redundant averages of DoFs on internal skeleton edges; ◢ global redundant moments of DoFs on internal skeleton edges. These last three categories are collected in the global vector of redundant coarse DoFs $\tilde{c}$; □ global non-redundant DoFs at crosspoints that are also coarse DoFs; ▭ global non-redundant averages of DoFs on internal skeleton edges; ◿ global non-redundant moments of DoFs on internal skeleton edges. These last three categories are collected in the global vector of non-redundant coarse DoFs c. The relationship between $\tilde{c}$ and c, visualized by dashed lines in (b), is enforced by the equation $\tilde{c} = \tilde{\mathcal{R}}_C c$.

3.3.1 Coarse degrees of freedom

Coarse DoFs are selected skeleton DoFs, or combinations thereof, used to capture the global behavior of the structure. In this work, three types of coarse DoFs are considered: cross-point values, edge averages, and edge moments (see Figure 7). This choice is inspired by [68, 69], although not identical. Cross points are nodes located at the intersection of mesh lines. The coarse DoFs associated with the i_x-th cross point are defined as

$$c_{i_x}^{u_1} = u_1|_{\boldsymbol{x}_{i_x}}\,, \tag{33a}$$

$$c_{i_x}^{u_2} = u_2|_{\boldsymbol{x}_{i_x}}\,, \tag{33b}$$

where u_1 and u_2 denote the components of the displacement vector, and $\boldsymbol{x}_{i_x}$ are the coordinates of the i_x-th cross point. The remaining coarse DoFs are associated to the i_e-th edge. Two average coarse DoFs are introduced, one for each displacement component:

$$c_{i_e}^{a_1} = \int_{\partial\Omega_{i_e}} u_1 \,\mathrm{d}\partial\Omega\,, \tag{34a}$$

$$c_{i_e}^{a_2} = \int_{\partial\Omega_{i_e}} u_2 \,\mathrm{d}\partial\Omega\,, \tag{34b}$$

where $\partial\Omega_{i_e}$ denotes the domain of the i_e-th edge. In addition, one moment-based coarse DoF is defined as

$$c_{i_e}^{m} = \int_{\partial\Omega_{i_e}} \boldsymbol{f}\cdot\boldsymbol{u} \,\mathrm{d}\partial\Omega\,, \tag{35}$$

where $\boldsymbol{f} = \begin{pmatrix} x_2 - \bar{x}_2 \\ \bar{x}_1 - x_1 \end{pmatrix}$ and $(\bar{x}_1, \bar{x}_2)$ are the coordinates of the center of the edge. Collecting all coarse DoFs into the vector c of size n_C, the following relationship holds:

$$c = Qu \,, \tag{36}$$

where $Q \in \mathbb{R}^{n_C \times n_U}$ is the matrix that enforces the above definitions through adequate integration and evaluation of basis functions. Figure 7 shows the different types of coarse DoFs for a single cell and for a two-by-two cells structure. As for the skeleton DoFs, the coarse DoFs vector on the i-th cell, denoted by $c^{(i)}$ and of size $n_C^{(i)}$, is obtained through an extension/restriction operator $\mathcal{R}_C^{(i)} \in \mathbb{R}^{n_C^{(i)} \times n_C}$,

$$c^{(i)} = \mathcal{R}_C^{(i)} c. \tag{37}$$

Similarly to $\mathcal{R}_U^{(i)}$, the matrix $\mathcal{R}_C^{(i)}$ consists of properly placed zeros and ones according to the global and local orderings of the coarse DoFs. By stacking the local coarse DoFs, the vector of redundant coarse DoFs $\tilde{c}$ of size $\tilde{n}_C = \sum_{i=1}^{n_c} n_C^{(i)}$ is defined as

$$\tilde{c} = \text{vstack}\left(c^{(1)}, \ldots, c^{(n_c)}\right) \,, \tag{38}$$

and the associated matrix $\tilde{\mathcal{R}}_C \in \mathbb{R}^{\tilde{n}_C \times n_C}$

$$\tilde{\mathcal{R}}_C = \text{vstack}\left(\mathcal{R}_C^{(1)}, \ldots, \mathcal{R}_C^{(n_c)}\right)$$

performs the operation

$$\tilde{c} = \tilde{\mathcal{R}}_C \, c \,.$$

Additionally, the constraint matrices $C^{(i)} \in \mathbb{R}^{n_C^{(i)} \times n_U^{(i)}}$ are introduced to extract the coarse DoFs of cell i from its skeleton DoFs:

$$C^{(i)} = \mathcal{R}_C^{(i)} Q \mathcal{R}_U^{(i)\top} \,,$$

associated to the relation

$$c^{(i)} = C^{(i)} u^{(i)} \,.$$

The global version of the precedent relation is expressed as:

$$\tilde{c} = \tilde{C} \tilde{u} \,,$$

where matrix $\tilde{C} \in \mathbb{R}^{\tilde{n}_C \times \tilde{n}_U}$ is defined as

$$\tilde{C} = \text{blockdiag}\left(C^{(1)}, \ldots, C^{(n_c)}\right) \,.$$

3.3.2 Coarse basis functions

Associated with the coarse DoFs c, the matrix $\tilde{\Psi}$ of size $\tilde{n}_U \times n_C$ is introduced, whose columns represent a coarse basis for the global correction. The matrix $\tilde{\Psi}$ is defined through

$$\begin{bmatrix} \tilde{S} & \tilde{C}^\top \\ \tilde{C} & 0 \end{bmatrix} \begin{bmatrix} \tilde{\Psi} \\ \tilde{\Lambda} \end{bmatrix} = \begin{bmatrix} 0 \\ \tilde{\mathcal{R}}_C \end{bmatrix} \,,$$

which implies that the i-th column of the matrix $\tilde{\Psi}$ is obtained as the solution associated with the i-th column of the matrix $\tilde{\mathcal{R}}_C$. The matrix $\tilde{\Lambda}$ contains the Lagrange multipliers associated with the coarse DoF constraints. Additionally, due to the block structures of $\tilde{S}$, $\tilde{C}$, and $\tilde{\mathcal{R}}_C$, the previous problem can be decomposed, and therefore parallelized, into local problems as

$$\begin{bmatrix} S^{(i)} & C^{(i)\top} \\ C^{(i)} & 0 \end{bmatrix} \begin{bmatrix} \Psi^{(i)} \\ \Lambda^{(i)} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathcal{R}_C^{(i)} \end{bmatrix} \,, \tag{39}$$

where $\Lambda^{(i)}$ contains the Lagrange multipliers associated to coarse DoFs at the cell level. Furthermore, it should be noted that while $\tilde{\mathcal{R}}_C$ has at least two nonzero elements in every column, this is not necessarily the

case for the local matrix $\mathcal{R}_C^{(i)}$ in cell i. Therefore, only a subset of the problems (39) needs to be solved for each cell. The coarse basis functions $\tilde{\Psi}$ are then assembled as

$$\tilde{\Psi} = \text{vstack}\left(\Psi^{(1)}, \ldots, \Psi^{(n_c)}\right) .$$

From a physical point of view, the i-th basis function (i.e., the i-th column of $\tilde{\Psi}$) represents the displacement field obtained by solving in each cell a local Dirichlet problem where, in addition to homogeneous Dirichlet conditions on the external boundary DoFs $u_D^{(i)}$, one coarse DoF is set equal to one while all the others are set to zero.

3.3.3 Discrete partition of unity matrix

In agreement with the PCG scheme, the input to the BDDC preconditioner is the residual force vector on the skeleton. As shown in the next subsection, the first step of the preconditioner is to distribute this vector to each cell by means of the extension/restriction operator $\tilde{\mathcal{R}}_U$. In doing so, it is important that the sum of the extended contributions equals one and that the forces are distributed proportionally to the local stiffness associated with their DoFs. For this purpose, the diagonal matrix $\tilde{D}$ is introduced, constructed as

$$\tilde{D} = \text{blockdiag}\left(D^{(1)}, \ldots, D^{(n_c)}\right), \tag{40}$$

where, for the i-th cell, the $j\,j$-th entry of the diagonal matrix $D^{(i)}$, corresponding to the node X_l, is defined as

$$D^{(i)}_{[j\,j]} = \frac{K^{(i)}_{[j\,j]}}{K_l^{tot}}, \tag{41}$$

where K_l^{tot} denotes the sum of all stiffness contributions associated with node X_l from the cells sharing that node.

3.3.4 The preconditioner

The BDDC preconditioner, denoted by S^{-1}_{BDDC}, provides an approximation of the inverse of the global Schur complement. Its action decomposes the correction of the residual into two contributions acting on distinct $\tilde{S}$-orthogonal subspaces. The first contribution is a *fine correction*: independent local Neumann-type problems are solved in each subdomain. Because such local Neumann problems are in general not invertible, the correction is computed in the subspace of functions that vanish at the coarse DoFs (i.e., $\ker(C^{(i)})$), which is enforced through local saddle-point systems (see Algorithm 2). The second contribution is a *coarse correction*: the residual is projected onto the subspace spanned by the coarse basis functions $\tilde{\Psi}$, and a global coarse system involving $S_C = \tilde{\Psi}^\top \tilde{S} \tilde{\Psi}^\top$ is solved. This coarse solve balances the local fine corrections, hence the name of the method. The constraints thus serve a dual purpose: they ensure the invertibility of local problems and define a coarse basis that allows the solver to scale effectively as the problem size grows. While the exact solution is globally continuous, the approximate correction obtained from the direct sum of these two subspaces is only guaranteed to be continuous at the coarse DoFs. To restore global continuity across the entire interface skeleton, the partially discontinuous correction is projected back onto the continuous space using the restriction operator $\tilde{\mathcal{R}}_U$ and the diagonal scaling matrix $\tilde{D}$. This weighting is essential for ensuring the robustness and convergence of the method.

The formal expression of the preconditioner reads

$$S^{-1}_{BDDC} = \tilde{\mathcal{R}}_U^\top \tilde{D}^\top \tilde{\Psi} S_C^{-1} \tilde{\Psi}^\top \tilde{D} \tilde{\mathcal{R}}_U + \sum_{i=1}^{n_c} \begin{bmatrix} D^{(i)} \mathcal{R}_U^{(i)} \\ 0 \end{bmatrix}^\top \begin{bmatrix} S^{(i)} & C^{(i)\top} \\ C^{(i)} & 0 \end{bmatrix}^{-1} \begin{bmatrix} D^{(i)} \mathcal{R}_U^{(i)} \\ 0 \end{bmatrix} . \tag{42}$$

In this expression, the first term corresponds to the coarse correction and the second term to the fine correction. The application of S^{-1}_{BDDC} is carried out through a sequence of matrix-vector multiplications and local solves, as detailed in Algorithm 1. While the local problems are fully decoupled and can be solved in parallel, the coarse problem requires the solution of a global system. Since S_C is explicitly assembled, the corresponding linear systems are solved using a direct method based on a parallel Cholesky factorization (specifically, the MUMPS solver [70], accessed through PETSc).

4 Solver accelerations

The BDDC method described in the previous section requires assembling the local stiffness matrix for each cell. However, as demonstrated in Section 5, this assembly step is relatively slow and represents the main computational bottleneck of the solver. This section outlines the strategies adopted in this work to accelerate the assembly process.

First, in Section 4.1 we employ the fast assembly technique introduced in [71]. This approach separates the contribution of the cell mapping, which determines the integrands of Equations (14), from the contribution arising from the implicit description of the trimmed domain, which affects only the quadrature rule. As a result, the integrand functions obtained no longer depend on the geometric map. Second, in Section 4.2 a reduced order model (ROM) is constructed for these integrand functions to avoid their computationally expensive evaluation. This process yields a surrogate model that, while specific to a given level-set, is valid for any geometric mapping for which the pulled-back coefficients satisfy the smoothness conditions required by the fast assembly technique. The final assembly process for the stiffness matrix is then summarized in Algorithm 4.

4.1 Fast assembly

Equation (20), introduces the terms that combine the constitutive equation and the external forces with the differential geometry quantities associated to the map. Such quantities, namely, the equivalent constitutive tensor $\hat{C}_{i_1 j_1 i_2 j_2}$, the equivalent body force $\hat{\boldsymbol{b}}$, and the equivalent traction $\hat{\boldsymbol{t}}$ are smooth on each cell, provided that the geometric map is regular (non-vanishing Jacobian) on the closure of the cell and that the material coefficients and loads are smooth cell-wise. Under these conditions, they can be interpolated at the cell level with high accuracy, resulting in

$$\hat{C}_{i_1 j_1 i_2 j_2}(\xi_1, \xi_2) \approx \mathscr{L}_k(\xi_1, \xi_2) \mathrm{C}_{i_1 j_1 i_2 j_2 k} \ , \tag{43a}$$

$$\hat{b}_i(\xi_1, \xi_2) \approx \mathscr{L}_k(\xi_1, \xi_2) \mathrm{B}_{ik} \ , \tag{43b}$$

$$\hat{t}_i(\xi_1, \xi_2) \approx \mathscr{L}_k(\xi_1, \xi_2) \mathrm{T}_{ik} \ , \tag{43c}$$

being $\hat{b}_i$ and $\hat{t}_i$ the components of $\hat{\boldsymbol{b}}$ and $\hat{\boldsymbol{t}}$, respectively, $\mathscr{L}_k(\xi_1, \xi_2)$ the Lagrangian polynomials of degree q associated with the interpolation nodes, selected also here as the Gauss-Lobatto-Legendre (GLL) tensor-product points, and $\mathrm{C}_{i_1 j_1 i_2 j_2 k}$, B_{ik}, and T_{ik} are the values of $\hat{C}_{i_1 j_1 i_2 j_2}$, $\hat{b}_i$, and $\hat{t}_i$, respectively, at the interpolation nodes. The components of the test (v_{i_1}) and trial (u_{i_2}) functions are first approximated using the basis functions $\mathscr{B}_k$:

$$v_{i_1}(\xi_1, \xi_2) = \mathscr{B}_{k_1}(\xi_1, \xi_2) \mathrm{V}_{i_1 k_1} \tag{44a}$$

$$u_{i_2}(\xi_1, \xi_2) = \mathscr{B}_{k_2}(\xi_1, \xi_2) \mathrm{U}_{i_2 k_2} \tag{44b}$$

where V_{ik} and U_{ik} are the degrees of freedom for the test and trial functions, respectively. Substituting this approximation into Equation (19), the expressions for the virtual work of the internal and external forces at the cell level are modified as follows:

$$\mathfrak{L}_{int}^{(i_c)} \approx \mathrm{V}_{i_1 k_1} \left[\left(\int_{\hat{\Omega}^{(c)}} \frac{\partial \mathscr{B}_{k_1}}{\partial \xi_{j_1}} \frac{\partial \mathscr{B}_{k_2}}{\partial \xi_{j_2}} \mathscr{L}_{k_3} \, \mathrm{d}\hat{\Omega} \right) \mathrm{C}_{i_1 j_1 i_2 j_2 k_3} \right] \mathrm{U}_{i_2 k_2} \ , \tag{45a}$$

$$\mathfrak{L}_{ext}^{(i_c)} \approx \mathrm{V}_{i_1 k_1} \left[\left(\int_{\hat{\Omega}^{(i_c)}} \mathscr{B}_{k_1} \mathscr{L}_{k_3} \, \mathrm{d}\hat{\Omega} \right) \mathrm{B}_{i_1 k_3} + \left(\int_{\partial \hat{\Omega}^{(i_c)}} \mathscr{B}_{k_1} \mathscr{L}_{k_3} \, \mathrm{d}\hat{\Omega} \right) \mathrm{T}_{i_1 k_3} \right] \ . \tag{45b}$$

The quantities in the square brackets are the entries of the stiffness matrix and the external force vector associated to the cell. However, only the quantities inside the integrals actually depend on the curvilinear coordinates ξ_1 and ξ_2. If the value of the integral were known, the assembly procedure would be equivalent to computing the tensorial contraction along j_1, j_2, and k_3, which is computationally cheap.

In a nutshell, if the constitutive relationship, the external forces, and the differential geometry, can be combined and accurately approximated as polynomials, then the integration could be precomputed with some efficient technique and the actual stiffness matrix and external forces vectors obtained by a fast tensorial

product with the coefficients of such polynomials. This procedure, introduced in [37], is referred to as the fast assembly technique.

It should be noted that the interpolation in Equations (43b) and (43c) is not always applicable. The fast assembly technique requires the geometric map to be regular, i.e., with non-vanishing Jacobian, on the closure of each cell, and the material coefficients, the body forces, and the tractions to be smooth on each cell. Therefore, discontinuities aligned with cell boundaries are admissible. Under these conditions, and in particular for the spline maps and constant material coefficients employed in this work, the pulled-back quantities $\hat{C}_{i_1 j_1 i_2 j_2}$, $\hat{\boldsymbol{b}}$, and $\hat{\boldsymbol{t}}$ are analytic within each cell, and their polynomial interpolation of degree q converges exponentially. Furthermore, the interpolation error can be evaluated a priori, by sampling the difference between the exact and interpolated quantities, so that q can be selected adaptively to meet a prescribed tolerance; the resulting consistency error is controlled through Strang's first lemma [37, 72].

When these regularity requirements are violated, for instance, when the geometric map approaches degeneracy or in the presence of discontinuous material fields or non-smooth loads, the interpolation error ceases to decrease at high order, and a full quadrature of the affected term is required. We remark, however, that the impact of non-smooth data is not limited to the fast assembly step. In fact, spectral elements themselves lose their high-order approximation properties when the solution lacks sufficient regularity.

Regarding the stabilization term in Equation (23), it can also be assembled using a fast assembly procedure. In particular,

$$\mathfrak{L}_{sta} \approx \mathrm{V}_{i_1 k_1} \left[\left(\int_{\hat{\Omega}_{\Pi}^{(i_c)}} \frac{\partial \mathscr{B}_{k_1}}{\partial \xi_{j_1}} \frac{\partial \mathscr{B}_{k_2}}{\partial \xi_{j_2}} \mathscr{L}_{k_3} \, \mathrm{d}\hat{\Omega} \right) \rho \mathrm{C}_{i_1 j_1 i_2 j_2 k_3} \right] \mathrm{U}_{i_2 k_2} \, . \tag{46}$$

Exploiting the identity

$$\int_{\hat{\Omega}_{\Pi}^{(i_c)}} \left(\bullet \mathrm{d}\hat{\Omega} \right) = \int_{\hat{\Pi}^{(i_c)}} \left(\bullet \mathrm{d}\hat{\Omega} \right) - \int_{\hat{\Omega}^{(i_c)}} \left(\bullet \mathrm{d}\hat{\Omega} \right) , \tag{47}$$

the above expression can be rewritten as

$$\mathfrak{L}_{sta} \approx \mathrm{V}_{i_1 k_1} \left[\left(\int_{\hat{\Pi}^{(i_c)}} \frac{\partial \mathscr{B}_{k_1}}{\partial \xi_{j_1}} \frac{\partial \mathscr{B}_{k_2}}{\partial \xi_{j_2}} \mathscr{L}_{k_3} \, \mathrm{d}\hat{\Omega} - \int_{\hat{\Omega}^{(i_c)}} \frac{\partial \mathscr{B}_{k_1}}{\partial \xi_{j_1}} \frac{\partial \mathscr{B}_{k_2}}{\partial \xi_{j_2}} \mathscr{L}_{k_3} \, \mathrm{d}\hat{\Omega} \right) \rho \mathrm{C}_{i_1 j_1 i_2 j_2 k_3} \right] \mathrm{U}_{i_2 k_2} \, . \tag{48}$$

The first integral in this expression can be precomputed and stored, as it does not depend on the trimming configuration. The second integral is formally equivalent to the one appearing in Equation (45a), which can therefore be reused here.

4.2 Reduced order modeling

In this section, we describe how to construct the ROM in order to avoid the need for full numerical quadrature. In this regard, to make it adaptable to different mappings, the models are constructed on the fast assembly integrals introduced in Section 4.1. The procedure is presented here for the tensor associated with the stiffness term, as in Equation (45a), but it is adopted also for the integrals related to the external forces as in Equation (45b). Let us denote as $\mathcal{I}$ the tensor whose components are

$$\mathcal{I}_{j_1 j_2}^{k_1 k_2 k_3} = \frac{\partial \mathscr{B}_{k_1}}{\partial \xi_{j_1}} \frac{\partial \mathscr{B}_{k_2}}{\partial \xi_{j_2}} \mathscr{L}_{k_3} . \tag{49}$$

The vector containing the integrals of the components of $\mathcal{I}$ is denoted as:

$$I(\boldsymbol{\mu}) = \int_{\hat{\Omega}(\boldsymbol{\mu})} \mathrm{vec}(\mathcal{I}) \, \mathrm{d}\hat{\Omega} \tag{50}$$

where vec(•) stands for the vectorization operation of the tensor •. Here, the dependence of I from the level-set threshold parameters $\boldsymbol{\mu} = \{\mu_1, \mu_2, \mu_3, \mu_4\}$, arises from the variation of the trimmed domain $\hat{\Omega}$. This section revolves around the strategy to avoid explicitly computing the integrals in Equation (50) by means of an offline built reduced order model.

In particular, the goal is to construct an approximation of the form:

$$I(\boldsymbol{\mu}) \approx U_r I_r(\boldsymbol{\mu}) \, , \tag{51}$$

where $U_r \in \mathbb{R}^{n_i \times n_r}$, and $I_r \in \mathbb{R}^{n_r}$. The columns of U_r constitute the basis vectors of the reduced order model, and the reduced order vector I_r contains the components of I with respect to such basis. The length of the vector I is denoted as n_i, while the number of basis vectors is denoted as n_r.

4.2.1 Clustering of the threshold parameters space

The construction of the ROM approximation begins with the definition of the threshold parameter space, $\boldsymbol{\mu} \in M$. This space is defined as the hypercube $M = [\mu_{\min}, \mu_{\max}]^d$, where $\mu_{\min}$ and $\mu_{\max}$ are the respective lower and upper bounds, and d the number of parameters, four in this case. This domain is then partitioned into clusters by subdividing each interval into n_k subintervals, resulting in a total of $n_K = (n_k)^4$ cluster domains denoted as M_{i_k}. The approximation described in Equation (51) is then performed independently within each cluster. In other words, a distinct ROM is constructed for each cluster, following the procedure outlined below.

To keep the notation concise, no additional index is introduced to explicitly indicate the cluster to which each quantity belongs, as this is not expected to generate ambiguity. It should be noted, however, that when evaluating functions of $\boldsymbol{\mu}$, the first step consists in identifying the cluster to which the corresponding threshold parameter vector belongs.

4.2.2 Reduced order basis

The basis U_r is constructed in an offline phase. First, a set of n_s representative level-set threshold parameter vectors, $\boldsymbol{\mu}^{i_s}$, is generated within the associated cluster domain. To ensure a representative sample with a limited number of elements, these parameter sets are selected using the Latin hypercube sampling method [73]. For each of these samples, the corresponding snapshot integrand, $I(\boldsymbol{\mu}^{i_s})$, is then computed exactly using the accurate quadrature procedure described in Section 2.2.3. These snapshots are then collected in the matrix

$$\mathfrak{I} = \left[I(\boldsymbol{\mu}^{\{1\}}),\ I(\boldsymbol{\mu}^{\{2\}}),\ \cdots,\ I(\boldsymbol{\mu}^{\{n_s\}}) \right] \in \mathbb{R}^{n_i \times n_s}. \tag{52}$$

A randomized singular values decomposition is then applied on this matrix and truncated to n_v singular values, with $n_v > n_r$ to account for the loss of accuracy on the trailing singular vectors, leading to the following approximated expression

$$\mathfrak{I} \approx U_v \Sigma_v V_v^{\intercal}, \tag{53}$$

where $U_v \in \mathbb{R}^{n_i \times n_v}$ is the matrix containing as columns the left singular vectors, $\Sigma_v \in \mathbb{R}^{n_v \times n_v}$ contains on the main diagonal the singular values, and $V_v \in \mathbb{R}^{n_s \times n_v}$ contains as columns the right singular vectors. The vectors of the basis U_r in Equation (51) are selected as the first n_r columns of U_v.

4.2.3 Coefficients of the reduced model

With the basis for the reduced-order model constructed, the non-linear function $I_r(\boldsymbol{\mu})$ is then approximated using the Matrix Discrete Empirical Interpolation Method (MDEIM) [47].

Firstly, the so-called *magic points* are selected [74]. The number of magic points is equal to the size of the reduced basis n_r, and correspond to positions within the basis vectors. As such, they are integers in the set $m_{i_r} \in \{1, \dots, n_i\}$, with $i_r = 1, \dots, n_r$. How the magic points are computed is detailed in Algorithm 3. In a nutshell, the algorithm works by iterating over the basis vectors; at each iteration, the entry that is most orthogonal to the previous vectors is selected.

The magic points are used to compute the reduced vector I_r from the full vector I. In particular, I_r contains coefficients chosen so that the reduced approximation matches the entries of I exactly at the magic points. This condition is expressed as

$$U_{r\,i}^{[m_j]}\, I_r^{[j]} = I^{[m_j]}, \qquad i, j = 1, \dots, n_r, \tag{54}$$

which corresponds to solving an $n_r \times n_r$ linear system, where $U_{r\,i}^{[m_j]}$ denote the m_j-th component of the i-th vector of the reduced-order basis, and $I_r^{[j]}$ and $I^{[m_j]}$ are the j-th and the m_j-th components of I_r and I, respectively.

However, computing I_r exactly requires access to the magic-points entries of the non-reduced vector I, which in turn would require a full integration process, which the proposed approach aims to avoid in the online phase. To overcome this issue, the vector $I_r(\boldsymbol{\mu})$ is computed exactly in an offline phase for a set of interpolatory snapshots. This sampling, which does not coincide with that used for the singular value decomposition, has a tensor-product structure within the associated cluster domain M_{i_k}, and follows the distribution of GLL points.

The function $I_r(\boldsymbol{\mu})$ is then obtained by interpolating these values, using Lagrangian polynomial of degree r. In this work, the interpolation is based on standard Lagrange polynomials defined over GLL nodes, which are preferred over radial basis functions [75]. This choice is motivated by the aim of minimizing the impact of the interpolation error on the overall accuracy of the MDEIM model. For problems with a larger number of parameters, Lagrangian interpolation may become too expensive in terms of both memory and computational cost. Alternative approaches, such as sparse grids [76], might be used in such cases.

4.3 Offline cost and speedup

This section quantifies the offline and online costs of the ROM-based acceleration strategies introduced above. The offline phase involves training a surrogate model for each level-set family. The online phase consists in evaluating the surrogate model to assemble each cell stiffness matrix, replacing the expensive numerical quadrature.

Offline training: The training is performed independently for each level-set geometry. The first step is to define the interval within which the threshold parameters are allowed to vary. This interval is chosen to exclude geometrically degenerate configurations, such as disconnected cells. For the Schwarz Diamond, two models are trained over the parameter spaces $\mathbb{P}_1^{\mathrm{SD}} = [0.1, 0.9]^4$ and $\mathbb{P}_2^{\mathrm{SD}} = [0.1, 1.0]^4$. For the Schoen IWP, the two corresponding spaces are $\mathbb{P}_1^{\mathrm{IWP}} = [-2.5, 2.5]^4$ and $\mathbb{P}_2^{\mathrm{IWP}} = [-2.5, 3.0]^4$. Where, for both geometries, the latter space is defined to include the fully solid cell, which however comes at the cost of a slight reduction in accuracy (see Section 5.1.2.

Each model is built for a polynomial degree $p = 8$ and a fast assembly interpolation degree $q = 2$. The parameter space is partitioned into $n_K = n_k^4 = 16$ clusters, $n_k = 2$ for each parametric direction. Within each cluster, $n_s = 100$ fully integrated snapshots are generated via Latin hypercube sampling, and a randomized SVD is performed, retaining $n_v = 50$ singular vectors of which the first $n_r = 40$ form the reduced basis.

The MDEIM coefficient fitting requires, per cluster, $(r+1)^4 = 7^4 = 2{,}401$ additional fully integrated evaluations on a tensor-product GLL grid, for a total of $n_K \times 7^4 = 38{,}416$ evaluations across all clusters. These parameter values were tuned based on the numerical results presented in Section 5. For the ROM corresponding to $\mathbb{P}_1^{\mathrm{IWP}}$, the total training time is approximately 9 hours (using 30 cores on a workstation with 4 Intel Xeon Gold 6148 processors and 1 TB of memory), and the total disk-storage footprint of the trained ROM data is 2.16 GB. Comparable training times were observed for the other geometries considered in this work. These figures include the model for the stiffness, the model for the mass matrix (not utilized in this contribute), and the model used to impose Neumann boundary conditions. The difference in the speedup factor arises from the higher geometric complexity of the Schoen IWP-based geometries, resulting in a larger number of quadrature points.

Table 2: Online assembly cost per cell, with and without the ROM, for two level-set families. Timings are averaged over 1000 cells.

Quantity	**Schwarz Diamond**	**Schoen IWP**
Full quadrature assembly	499 ms	870 ms
Assembly time with ROM	2.3 ms	2.5 ms
Speedup factor	218×	342×

Online assembly and amortization threshold: Once the ROM is available, the assembly cost per cell is reduced to a Lagrangian interpolation of the MDEIM reduced vector, followed by a multiplication with the reduced basis. The assembly times, with and without the ROM, for two level-set families and the settings described in Section 5.1.1, are reported in Table 2.

The amortization threshold is defined as the total number of cell assembly evaluations required for the cumulative online savings to offset the offline training cost. Denoting the training time as T_{off}, the per-cell assembly time without ROM as T_{full}, and with ROM as T_{rom}, the threshold is

$$n_{\text{amm}} = \left\lceil \frac{T_{\text{off}}}{T_{\text{full}} - T_{\text{rom}}} \right\rceil . \tag{55}$$

As an example, for the ROM model $\mathbb{P}_1^{\text{IWP}}$, the break-even point is reached after approximately 38000 cell evaluations, beyond which the offline training cost is amortized. This threshold is largely independent of the geometry, as the offline training cost is primarily determined by the fixed number of fully integrated cells used to construct the model.

On this regard, it is worth noting that the number of TPMS geometries used as lattice micro-structures is limited. The training therefore represents a one-time community investment rather than a per-user burden. Consistent with this philosophy, the precomputed ROM models used in this work are publicly available on Zenodo [77]. These models are directly reusable in any simulation involving the same level-set families: indeed, they were computed once and then reused, without any retraining, in all the analyses of Section 5.

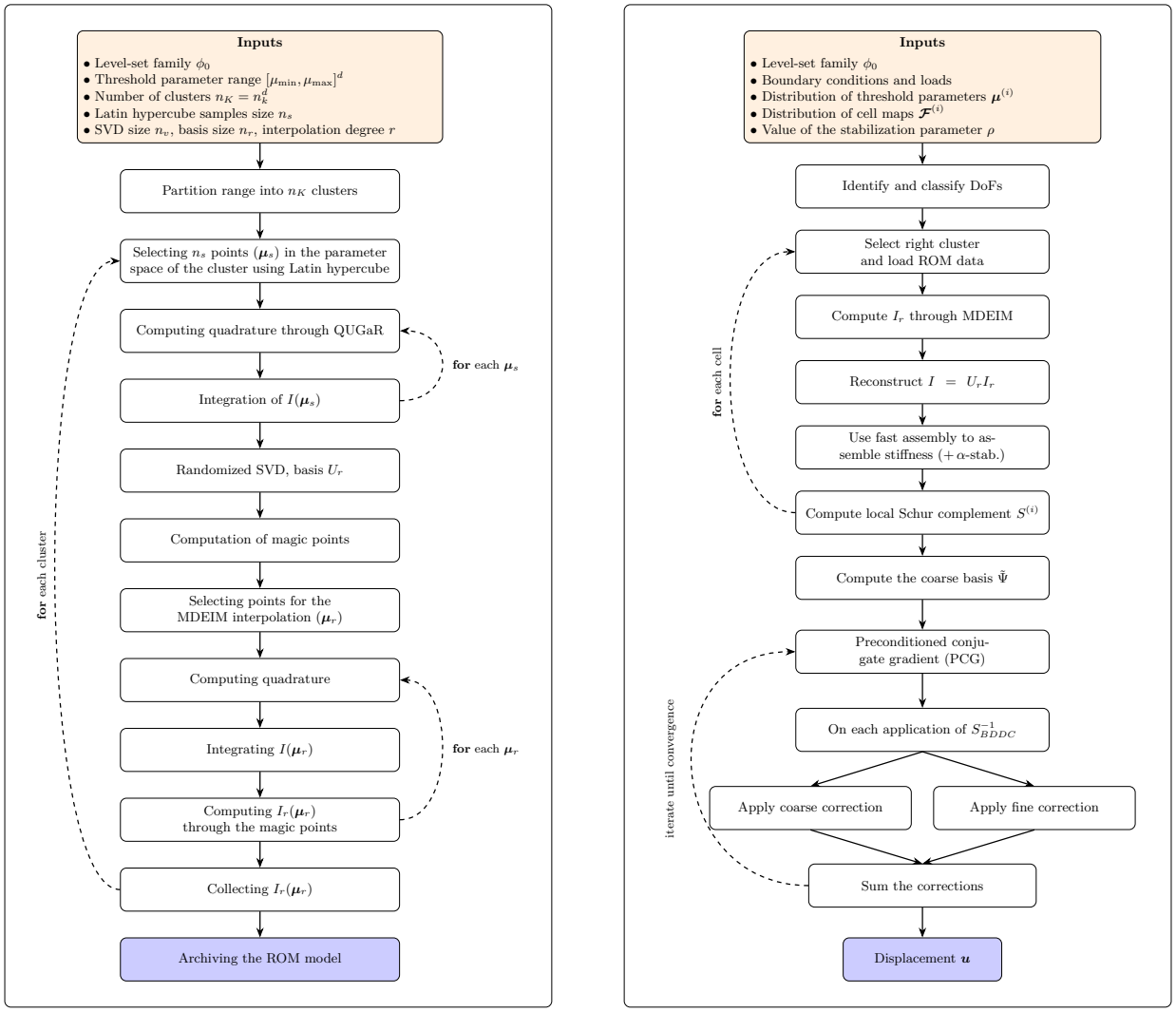
4.4 Summary of the computational workflow

Having introduced the individual ingredients and quantified their offline and online cost, we now summarize how they combine into a single offline–online pipeline, illustrated in Figure 8. The two panels show the respective workflows from the input box to the output box: Figure 8a traces the one-time offline training of the ROM, while Figure 8b traces the online assembly and the subsequent BDDC solve. The two stages are deliberately decoupled, and trimmed quadrature is confined entirely to the offline stage.

Offline phase: Following Figure 8a, the range of the threshold parameters is partitioned into the $n_K = (n_k)^4$ clusters, and every subsequent step is repeated independently for each cluster. Within a cluster, snapshots of the fast-assembly tensor $I(\boldsymbol{\mu})$ are computed by full trimmed quadrature (Section 2.2.3) at the n_s parameter samples $\boldsymbol{\mu}_s$ obtained by Latin hypercube sampling, and a truncated randomized SVD of these snapshots yields the reduced basis U_r. The MDEIM magic points are then extracted (Algorithm 3), and the interpolation coefficients are fitted from a second, tensor-product set of samples $\boldsymbol{\mu}_r$, which likewise require full quadrature. Once all clusters have been processed, the trained model is archived for reuse (Section 4.3). We note once again that the resulting trimming-based ROM is geometry-specific but mapping-independent.

Online phase: The online stage corresponds to Figure 8b. After the degrees of freedom have been identified and classified as either Dirichlet, internal, or skeleton DoFs, the solver loops over the cells: given the threshold parameters $\boldsymbol{\mu}^{(i)}$ of a cell, the relevant cluster is selected and the reduced vector I_r is obtained by MDEIM interpolation, from which the fast-assembly tensor is reconstructed as $I \approx U_r I_r$ (Algorithm 5). This tensor is contracted with the mapping coefficients corresponding to $\boldsymbol{\mathcal{F}}^{(i)}$ to form the local stiffness matrix, to which the α-stabilization is added (Algorithm 4), the local Schur complement $S^{(i)}$ is then formed. Once every cell has been processed, it is computed the coarse basis $\tilde{\Psi}$ and the preconditioned conjugate gradient is initiated. When requested by the iterative solver scheme, application of preconditioner S_{BDDC}^{-1} is carried out by superimposition of a coarse and a fine correction (Algorithm 1), the first relying on a global direct solve, and the latter relying on the local saddle-point solves of Algorithm 2, and the iteration proceeds until the prescribed tolerance is met, returning the displacement field $\boldsymbol{u}$.

Read together, the two panels make explicit the migration of the dominant cost. In the unaccelerated baseline, the per-cell trimmed quadrature dominates the setup phase (Figure 17b); the acceleration layers move this cost to the one-time offline training of Figure 8a, so that the online assembly of Figure 8b becomes negligible and the BDDC setup and solve times become comparable (Figure 16). The ingredients remain simultaneously effective provided that the pulled-back integrands are smooth, as required by the fast assembly (Section 4.1); that the threshold parameters lie within the admissible range used to train the ROM and to exclude degenerate cells; and that the α-stabilization keeps the conditioning of strongly trimmed cells under control, which in turn supports the robustness of the BDDC coarse space.



(a) (b)

Figure 8: Computational workflow of the ROM-based BDDC solver, read top to bottom from the input to the output box. (a) Offline ROM-training pipeline, built once per level-set family. (b) Online per-cell assembly and BDDC solve, performed for every analysis. The individual steps, with the corresponding algorithms, are detailed in the text.

5 Numerical Results

In this section, several numerical examples are presented to assess the performance of the proposed computational framework. The section is divided into two parts. Initially, tests are conducted on simple geometries to evaluate the impact of the various approximations introduced by the method, namely the α-stabilization, the fast assembly, and the reduced-order modeling, and to guide the selection of the relevant parameters. The performance is also assessed in terms of iteration count and total analysis time. In the second part, the proposed approach is applied to more realistic geometries, namely a sandwich wing and a lattice wrench.

It is further recalled that, within the presented BDDC framework, there is a one-to-one correspondence between cells and subdomains. Accordingly, the two terms are used interchangeably throughout the section.

The method presented in this paper is implemented in FLAS$_h$, an open-source Python library [78]. In particular, the repository includes dedicated example scripts that reproduce each figure and timing table presented in this section, along with the corresponding input configurations and solver settings. Table 3 lists, for each figure of this section, the script(s) of the FLAS$_h$ repository reproducing it, all referring to version v0.2.0. FLAS$_h$ relies on PETSc4py [79, 80] for the parallel solver, NumPy and SciPy for numerical computations, and QUGaR [61] for the unfitted discretizations. Precomputed ROM models used in this work are publicly available on Zenodo [77] and are automatically downloaded during the library installation. All the numerical results presented in this section are fully reproducible using FLAS$_h$. The reported tests were performed on a Apple MacBook Pro with an Apple M4 Pro chip (14-core CPU: 10 performance + 4 efficiency cores; 48 GB unified memory). The simulations were run in parallel using 8 performance cores.

Table 3: Scripts of the FLAS$_h$ repository (v0.2.0) reproducing the figures of this section.

Figure	**Section**	**Script(s)**
9	5.1.1	`test_convergence.py`
10	5.1.1	`test_stabilization_1.py`
11	5.1.1	`test_stabilization_2.py` (a), `test_stabilization_3.py` (b)
13	5.1.2	`test_fast_assembly_accuracy.py`
14	5.1.2	`test_rom_basis.py`
15	5.1.2	`test_rom_accuracy.py`
16	5.1.3	`test_solver_comparison.py`
17	5.1.3	`test_acceleration_efficiency.py`
18	5.1.3	`test_scalability.py`
19	5.2.1	`example_wing.py`
20	5.2.2	`example_wrench_coarse.py` (a,b), `wrench_refined.py` (c,d)
21	5.2.2	`wrench_refined.py`, `wrench_validation.py`, `wrench_paper_summary.py`, `plot_wrench_h_study.py`

5.1 Method validation and benchmarking

This section justifies the choice of discretization strategy, evaluates the accuracy of each approximation component (fast assembly, ROM, and stabilization), and assesses the computational performance of the overall method. Throughout this section, the term *baseline* refers to the BDDC method assembled with full integration without applying any acceleration technique or stabilization. Additionally, all errors reported in this section are normalized with respect to the corresponding reference quantities.

5.1.1 Accuracy assessment on a single-cell structure

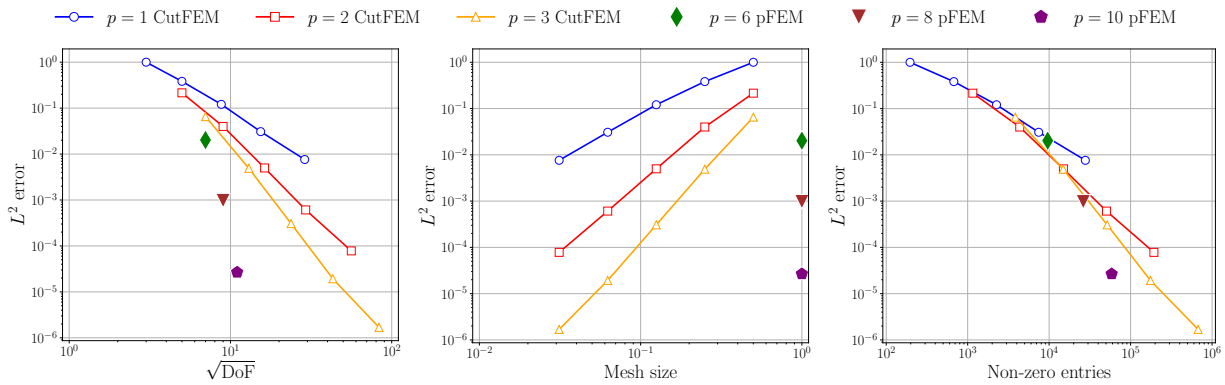
As a starting point, a simple single-cell geometry is considered. The threshold parameter $\mu(\boldsymbol{\xi})$ is set to be uniform, with value specified in the tests. . The geometry is subsequently transformed through the map.

$$\boldsymbol{\mathcal{F}} = \begin{bmatrix} \frac{1}{2} + \left(\xi_1 - \frac{1}{2}\right)\left(1 + (a-1)(2\xi_2 - \xi_2^2)\right) \\ \xi_2 \end{bmatrix},$$

where a is a parameter controlling the severity of the geometric distortion. The body forces, prescribed displacements, and boundary tractions for this test are derived from the reference manufactured solution:

$$\boldsymbol{u}_{ex} = \begin{bmatrix} u_0(2x_2 - 1)\sin^2(\pi x_1)\sin^2(\pi x_2) \\ u_0(1 - 2x_1)\sin^2(\pi x_1)\sin^2(\pi x_2) \end{bmatrix}. \tag{56}$$

It is important to note that, here, Neumann boundary conditions are also applied on the trimmed boundary $\partial\Omega_t$ to enable a consistent comparison with the exact solution, as permitted by the manufactured nature of the problem settings. The Lamé constants of the adopted material are both set unitary. Here, no stabilization or acceleration techniques are employed in the assembly of the stiffness matrices.



(a) (b) (c)

Figure 9: Comparison between unfitted p-FEM element method and CutFEM for the manufactured problem described in Section 5.1.1. In the left picture the L^2 error is plotted against the square root of the number of DoFs. In the center, the L^2 error is plotted against the mesh size. Here p-FEM data are vertically aligned since the mesh size in all such cases is unitary. On the right, the L^2 error is reported as a function of non-zero entries of the stiffness matrix.

Accuracy of the p-FEM discretization In this test, the adopted level-set is the Schoen FRD shown in Figure 2c. The threshold parameter is set to zero, and the transformation is taken as the identity ($a = 1$). No stabilization is adopted in this test. Two different discretizations are taken into account. Specifically, either a single unfitted p-FEM element is adopted, or a grid of rectangular elements is employed within a CutFEM framework [81, 45]. Figure 9 shows the convergence curves of the L^2 error in the displacement field for both discretizations, with respect to the square root of the degrees of freedom, the mesh size, and the number of non-zero entries of the stiffness matrix. From these results, it is evident that a single high-degree p-FEM element can achieve the same level of accuracy with a significantly lower number of degrees of freedom. This is particularly apparent for $p = 10$, which outperforms the CutFEM refinement of degree $p = 3$ by requiring more than an order of magnitude fewer degrees of freedom. This advantage becomes less pronounced when considering the number of nonzero entries of the stiffness matrix. In this case, the p-FEM with $p = 10$ requires fewer than four times the number of nonzero entries compared to the corresponding CutFEM curve with $p = 3$. This comparison becomes even more relevant when examining the $p = 6$ and $p = 8$ points for the p-FEM, where CutFEM refinements of order $p = 2$ and $p = 3$ can be more efficient.

However, when adopting a CutFEM approach, shape functions may switch between active and inactive states, which can introduce difficulties in the construction of the ROM. This issue does not arise in the

p-FEM case. Consequently, the stiffness matrix to be approximated within the ROM framework depends continuously on the threshold parameters, therefore making the surrogate model more accurate. For this reason, in this work p-FEM elements are adopted to approximate the solution within each cell. Additionally, unless otherwise specified, the polynomial degree is set to $p = 8$, as it provides a suitable compromise between accuracy and computational cost.

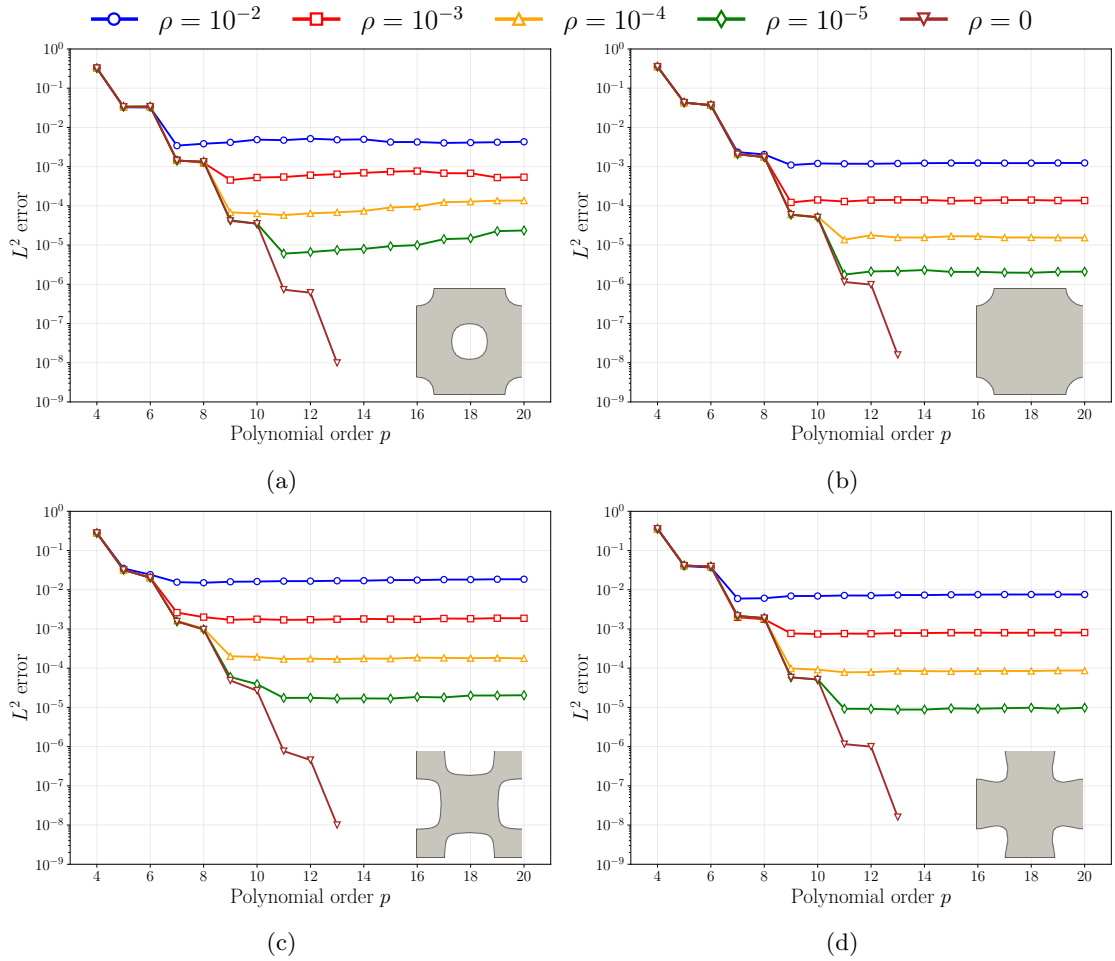
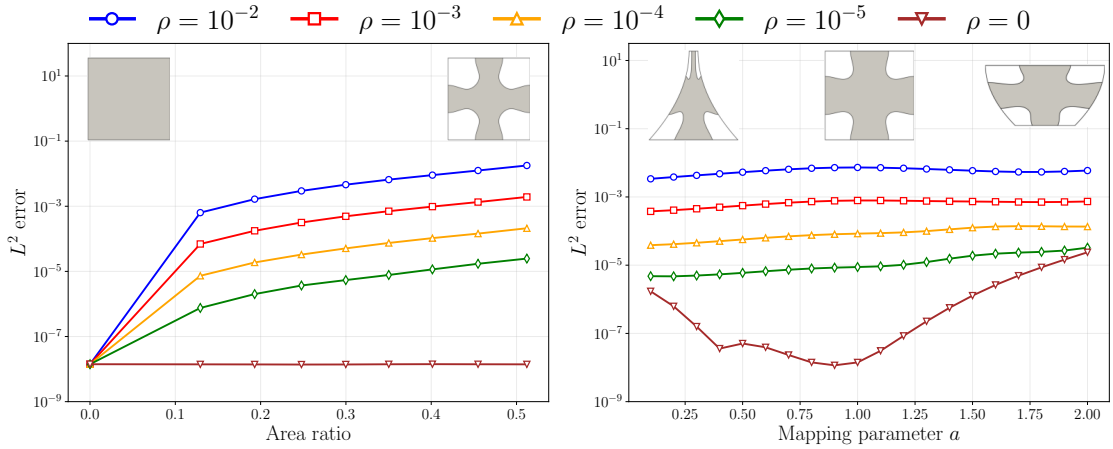


Figure 10: L^2 error as a function of the approximation degree p for different values of the stabilization parameter ρ, computed on a single cell with the manufactured solution described in Section 5.1.1. Results are shown for the four TPMS geometries introduced in Figure 2: (a) Schwarz Diamond, (b) Schwarz Primitive, (c) Schoen FRD, and (d) Schoen IWP.

Accuracy of the stabilization The accuracy impact of the stabilization parameter ρ is assessed using the single-cell manufactured solution described above, considering five values $\rho \in \{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 0\}$. Three aspects are investigated: the effect on p-convergence, the sensitivity to trimming severity, and the robustness with respect to mapping distortion.

Figure 10 reports the L^2 error as a function of the polynomial degree p for all four TPMS geometries, with the identity map ($a = 1$) and a uniform threshold parameter (0.1 for the Schwarz Diamond and 0 for the remaining cases). Without stabilization ($\rho = 0$), spectral convergence is achieved in all cases. The staircase pattern for $\rho = 0$ results from the higher ability of odd-order polynomials to capture the reference solution during p-convergence. As ρ increases, the error eventually plateaus at a level approximately equal to the stabilization parameter itself, with minor differences among geometries. This confirms that the stabilization introduces a consistency error whose magnitude scales linearly with ρ, limiting high-order accuracy for large stabilization values.



(a) (b)

Figure 11: L^2 error for the Schoen IWP geometry at fixed polynomial degree $p = 14$, for different values of the stabilization parameter ρ. (a) Error as a function of the trimmed area ratio, controlled by a uniform additive shift of the threshold parameter, ranging from no trimming to increasingly trimmed configurations. (b) Error as a function of the mapping distortion parameter a, where $a = 1$ corresponds to the identity map.

Figure 11a shows the L^2 error at fixed $p = 14$ as a function of the trimmed area ratio. Such a high polynomial degree, compared to the degree $p = 8$ employed in the rest of this work, is deliberately chosen to drive the approximation error well below the error floor induced by the stabilization, thereby isolating the consistency error introduced by ρ from the approximation error. Without stabilization, the error remains at the level dictated by the element's approximation accuracy, regardless of how severely the cell is trimmed, demonstrating that the base method is inherently robust to trimming. With stabilization, the error increases monotonically with the trimmed area, since the penalty term acts over an increasingly large fictitious extension of the domain. Larger values of ρ lead to proportionally higher errors across all trimming levels.

Figure 11b reports the L^2 error at fixed $p = 14$ as a function of the mapping distortion parameter a, for a homogeneously zero threshold parameter. For $\rho = 0$, the error is minimized near $a = 1$ (identity map), corresponding to a trimmed area ratio of 0.67, and grows for strongly compressed ($a = 0.1$) or stretched ($a = 2$) configurations, reflecting the reduced approximation accuracy of the polynomial basis defined in the parametric domain. With stabilization, the error remains bounded across the full range of mapping parameters, with the floor determined by ρ.

These results characterize how the interaction between p, ρ, μ, and $\boldsymbol{\mathcal{F}}$ affects the accuracy of the analysis. Extending this characterization to a broader range of multi-cell geometries and loading conditions is a natural continuation of this study.

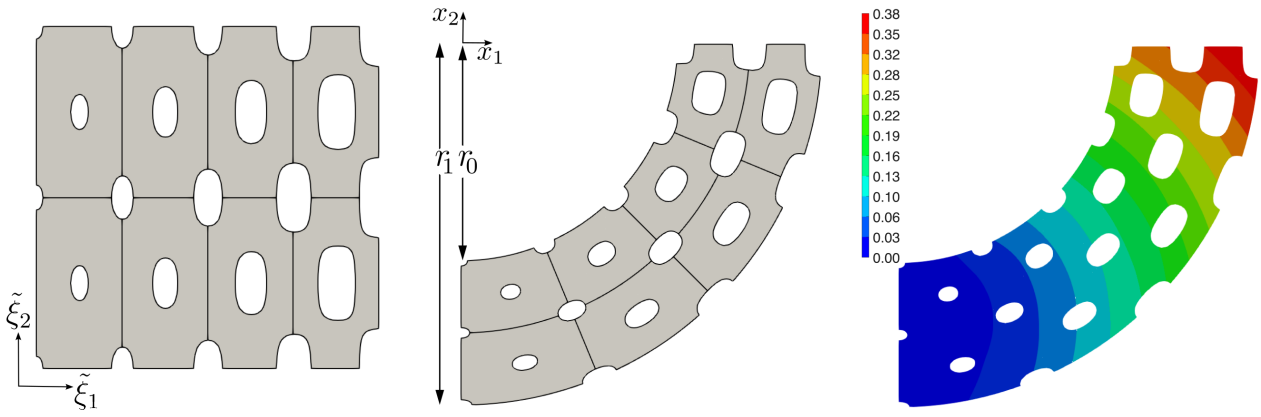
5.1.2 Accuracy assessment on a multi-cells structure

The remaining examples in this subsection share a common benchmark geometry, namely the lattice structure depicted in Figure 12. The geometry under consideration is constructed by adopting an intermediate step with respect to the procedure described in Section 2.1. Here, it is introduced an auxiliary parametric domain $\tilde{\Pi} = [0, 1]^2$, with associated curvilinear coordinates $\tilde{\xi}_1$ and $\tilde{\xi}_2$. The untrimmed physical domain Π is then defined through the auxiliary mapping

$$\tilde{\mathcal{F}}(\tilde{\xi}_1, \tilde{\xi}_2) = \begin{bmatrix} +(r_0 + (r_1 - r_0)\tilde{\xi}_2)\sin(\frac{\pi}{2}\tilde{\xi}_1) \\ -(r_0 + (r_1 - r_0)\tilde{\xi}_2)\cos(\frac{\pi}{2}\tilde{\xi}_1) \end{bmatrix}, \tag{57}$$

where $r_0 = 0.6$ m and $r_1 = 1$ m, corresponding to a quarter annulus.

The untrimmed cells are then constructed with the aid of $\tilde{\Pi}$ by partitioning it into a tensor-product rectangular mesh. The number of cells is chosen such that the number of cells in the $\tilde{\xi}_1$ direction is twice



(a) (b) (c)

Figure 12: Benchmark geometry for the tests in Sections 5.1.2 and 5.1.3. (a) Auxiliary parametric domain after trimming operation. (b) Physical domain resulting from the map given in Equation (57). (c) Contour of the magnitude of the displacements expressed in m.

that in the $\tilde{\xi}_2$ direction. This mesh naturally induces a partitioning of Π into curvilinear quadrilateral cells. Starting from this geometric construction, it is straightforward to define, for each cell, a mapping from its local parametric domain $\hat{\Pi}^{(i_c)} = [0,1]^2$ to the corresponding physical untrimmed domain in the assembled structure, thereby recovering the procedure described in Section 2.1.1. Such an intermediate auxiliary parameter domain $\tilde{\Pi}$, although not strictly necessary, is particularly useful when the arrangement of the lattice cells follows a tensor-product structure, allowing the mapping to be reduced to a common global one.

With regard to the trimming procedure described in Section 2.1.2, the level-set function selected here is the Schwarz diamond, as defined in Table 1. The threshold parameters are chosen according to the position of the corresponding node in the auxiliary parametric domain $\tilde{\Pi}$. In particular, the threshold parameters are defined as

$$\mu_i(\tilde{\boldsymbol{\xi}}) = 0.9 - 0.8\,\tilde{\xi}_1\,.$$

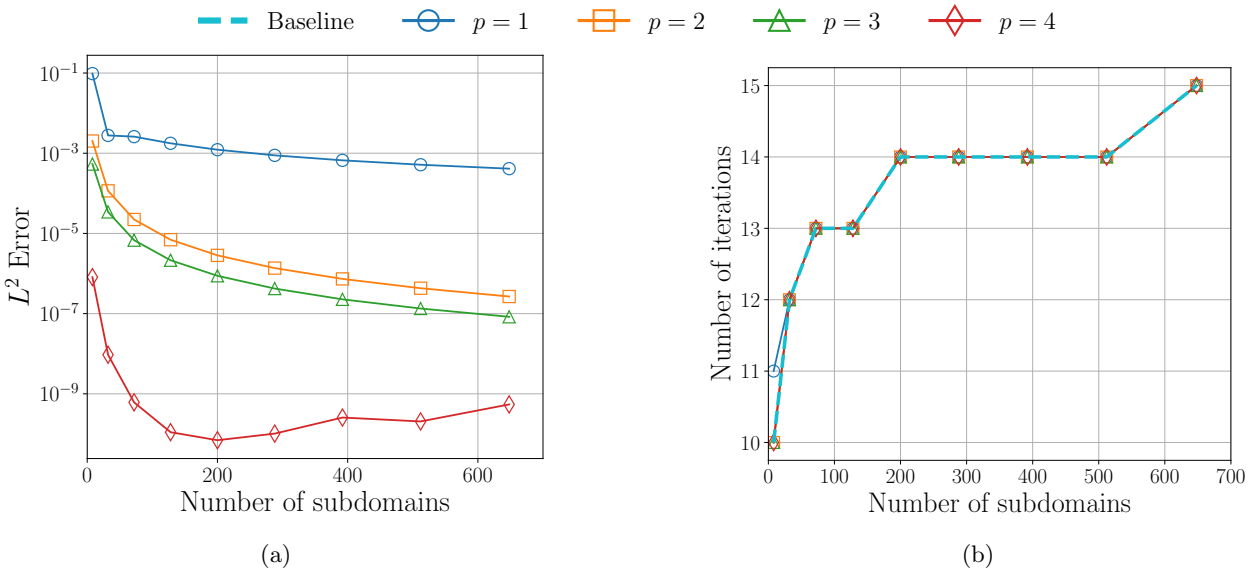
It is important to remark that the introduction of the auxiliary parametric domain $\tilde{\Pi}$ is particularly convenient for lattice structures whose cell distribution follows a tensor-product pattern, as in the case of Figure 12. However, this assumption is not strictly required, and structures that do not follow such a cells organization can also be analyzed within the proposed framework, as will be demonstrated in Section 5.2.

The material properties are defined by a Young's modulus $E = 5$ N/m and a Poisson's ratio $\nu = 0.25$. No body forces are applied within the domain. The boundary corresponding to $\tilde{\xi}_1 = 0$ is clamped, while a traction $f = 0.1$ N/m, directed downward in the physical space, is applied on the boundary corresponding to $\tilde{\xi}_1 = 1$. The deformation magnitude is depicted in Figure 12c.

Throughout the remainder of this subsection, the parameters under investigation are the stabilization parameter ρ, the degree q of the fast assembly technique, and the number of basis vectors n_r and clusters n_k for the ROM, as described in Sections 2.2.3, 4.1, and 4.2, respectively. A polynomial basis of degree 8 is employed in all tests. However, when not explicitly varied, or differently specified, the parameters are set to $\rho = 5 \cdot 10^{-4}$, $q = 2$.

Accuracy of the fast assembly technique We begin by assessing the capability of the fast assembly technique described in Section 4.1 to accurately interpolate the terms that combine the constitutive parameters and external forces with the differential geometry contributions associated with the mapping. Here, the stabilization parameter ρ is set to zero and no ROM acceleration is applied.

To evaluate the accuracy of the interpolation in Equation (43), Figure 13a shows the L^2 error on the displacement vector obtained with the fast assembly technique compared with a standard integration. It can be observed that increasing the interpolation degree reduces the error as expected. Furthermore, the



(a) (b)

Figure 13: (a) Accuracy of the fast assembly interpolation measured in the L^2 norm of the displacements error. (b) Number of iterations required by the solver when the fast assembly technique is adopted.

interpolation becomes increasingly accurate as the number of subdomains increases, since the mapping for a small cell approaches a simpler bilinear one.

Additionally, Figure 13b reports the number of iteration required by the solver, varying the degree q of the fast assembly interpolation and the number of subdomains. The plots demonstrate that the interpolation degree has no significant influence on the number of iterations. In the remainder of the tests, the interpolation degree for the fast assembly technique is set to two, as it provides a good compromise between accuracy and memory consumption. In fact, a higher interpolation degree leads to larger tensors I and, consequently, a larger ROM. It should be noticed that for $q = 2$, even considering a single subdomain, the error in Figure 13a is still of the order of magnitude of 10^{-3}.

Accuracy of the reduced order model Next, we study how the performance of the method depends on the number of basis functions n_r and the number of clusters n_k in the ROM, seeking a suitable compromise between accuracy and efficiency. Four level-set geometries are considered, namely the Schwarz diamond, the Schoen IWP, the Schoen FRD, and the Schwarz Primitive (see Figure 2). For each geometry, the parameter domain over which the threshold parameters are allowed to vary is specified for the construction of the ROM. In particular, two options are considered for each case, as defined in Section 4.3. For the Schwarz diamond, the threshold parameters are taken within either $\mathbb{P}_1^{\text{SD}} = [0.1, 0.9]^4$ or $\mathbb{P}_2^{\text{SD}} = [0.1, 1.0]^4$; for the Schoen IWP within either $\mathbb{P}_1^{\text{IWP}} = [-2.5, 2.5]^4$ or $\mathbb{P}_2^{\text{IWP}} = [-2.5, 3.0]^4$; for the Schoen FRD, within either $\mathbb{P}_1^{\text{FRD}} = [-6.5, 0.5]^4$ or $\mathbb{P}_2^{\text{FRD}} = [-7.0, 0.5]^4$; and for the Schwarz Primitive, within either $\mathbb{P}_1^{\text{SP}} = [-0.5, 0.8]^4$ or $\mathbb{P}_2^{\text{SP}} = [-0.5, 1.0]^4$. The use of two different parameter spaces for each geometry is intended to illustrate the performance of the ROM when the parameter domain used for its construction is progressively enlarged. Specifically, the sets $\mathbb{P}_2^{\text{SD}}$, $\mathbb{P}_2^{\text{IWP}}$, $\mathbb{P}_2^{\text{FRD}}$, and $\mathbb{P}_2^{\text{SP}}$ contain the combination of the threshold parameters that reproduces a full cell. We recall that the ROM models are built on the fast assembly tensor I as described in Sections 4.1 and 4.2. The ROM coefficients are computed via Lagrangian interpolation (see Section 4.2.3), with interpolation order 6 used throughout the results presented below.

Figure 14 reports the results of this study. Figure 14a refers to the Schwarz diamond level-set, Figure 14b to the Schoen IWP, Figure 14c to the Schoen FRD, and Figure 14d to the Schwarz Primitive. The vertical axis represents a measure of the ROM error, computed as follows. For 50 different randomly generated snapshots (for each cluster) that are not included in the singular value decomposition training set, the L_∞ norm of the difference between the vectorized tensor I obtained through full integration and its projection in the reduced-order basis is evaluated and normalized by the L_∞ norm of I. The final ROM error is then

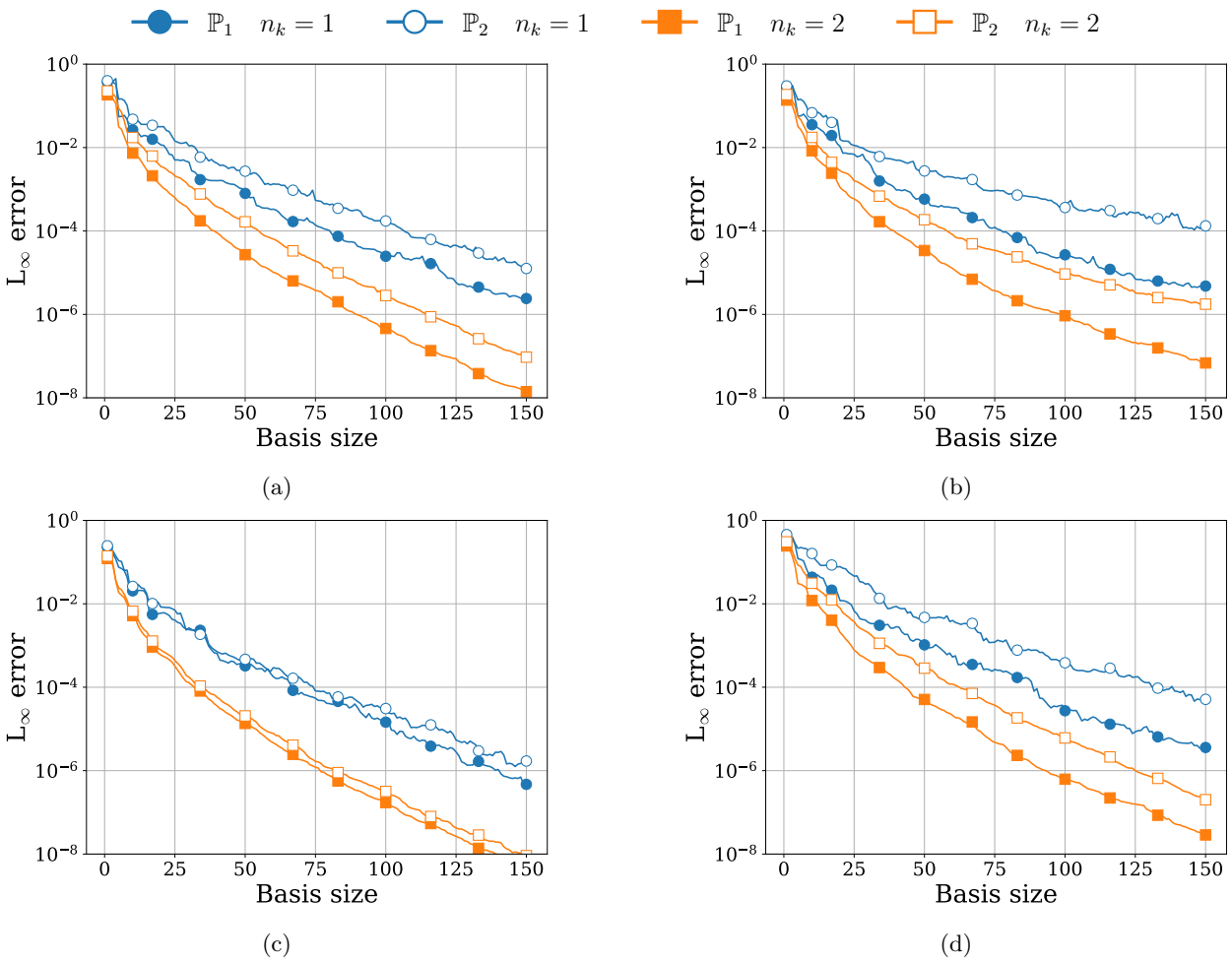


Figure 14: Error of the fast assembly tensor depending on the basis size n_r and for different number of clusters n_k. (a) Schwarz diamond, with space of the threshold parameters $\mathbb{P}_1^{\mathrm{SD}}$ and $\mathbb{P}_2^{\mathrm{SD}}$, (b) Schoen IWP, with threshold-parameter space $\mathbb{P}_1^{\mathrm{IWP}}$ and $\mathbb{P}_2^{\mathrm{IWP}}$, (c) Schoen FRD, with space of the threshold parameters $\mathbb{P}_1^{\mathrm{FRD}}$ and $\mathbb{P}_2^{\mathrm{FRD}}$, (d) Schwarz Primitive, with space of the threshold parameters $\mathbb{P}_1^{\mathrm{SP}}$ and $\mathbb{P}_2^{\mathrm{SP}}$

obtained as the arithmetic average of these values.

As expected, the error decreases as both the number of basis vectors and the number of clusters increase, reflecting the improved accuracy of the ROM. Furthermore, as the parameter domain expands from $\mathbb{P}_1$ to $\mathbb{P}_2$, a deterioration in accuracy is observed for all the tested geometries. Similar trends are evident in all panels of Figure 14; however, the error level is geometry-dependent, with higher values observed for the Schoen IWP and a milder but still visible deterioration also for the newly included Schoen FRD and Schwarz Primitive cases, indicating that the ROM reliability is sensitive to both the geometry family and the extent of the threshold-parameter space.

For the remainder of this paper, unless otherwise specified, the ROM is constructed, as described in Section 4.3, using two clusters per threshold parameter (resulting in a total of 16 clusters) and 40 basis vectors per cluster. Furthermore, the threshold parameters are taken in the sets $\mathbb{P}_1$. This choice ensures an accuracy comparable to that of the fast assembly procedure, so that the different components of the method exhibit a balanced level of approximation.

We remark that, since in the online phase the surrogate is only queried inside the threshold-parameter box sampled during training, the study of Figure 14 constitutes a practical acceptance test that can be carried out entirely at the ROM training stage, at a marginal cost compared to the training itself (50 additional full-quadrature evaluations per cluster): the user selects the basis size and the number of clusters so that the

indicator meets a prescribed tolerance, or otherwise rejects the model and retrains it with more clusters or a smaller parameter range. The reported projection error relates to the actual interpolation error through a computable amplification factor associated with the magic-point rows of the basis [46]. Its transfer to the solution follows a first-order perturbation argument: the entries of the cell operators depend linearly on the tensor I, so a relative error ε on I induces an operator perturbation of the same order, and the ensuing relative solution error is bounded, to first order, by ε times a constant depending on the conditioning of the stabilized problem. For energy-type quantities of interest, such as the compliance, the first-order error is bounded by the energy-relative operator perturbation, with no conditioning amplification. These transfer constants are moderate in practice: as will be shown in the wrench example of Section 5.2.2, replacing the ROM by full integration changes the monitored quantities of interest by $1.4 \cdot 10^{-4}$, of the same order as the offline indicator for the settings employed here. A practical guideline is thus to select, at the ROM training stage, the ROM configuration such that the offline indicator lies below the target discretization error, as done in this work.

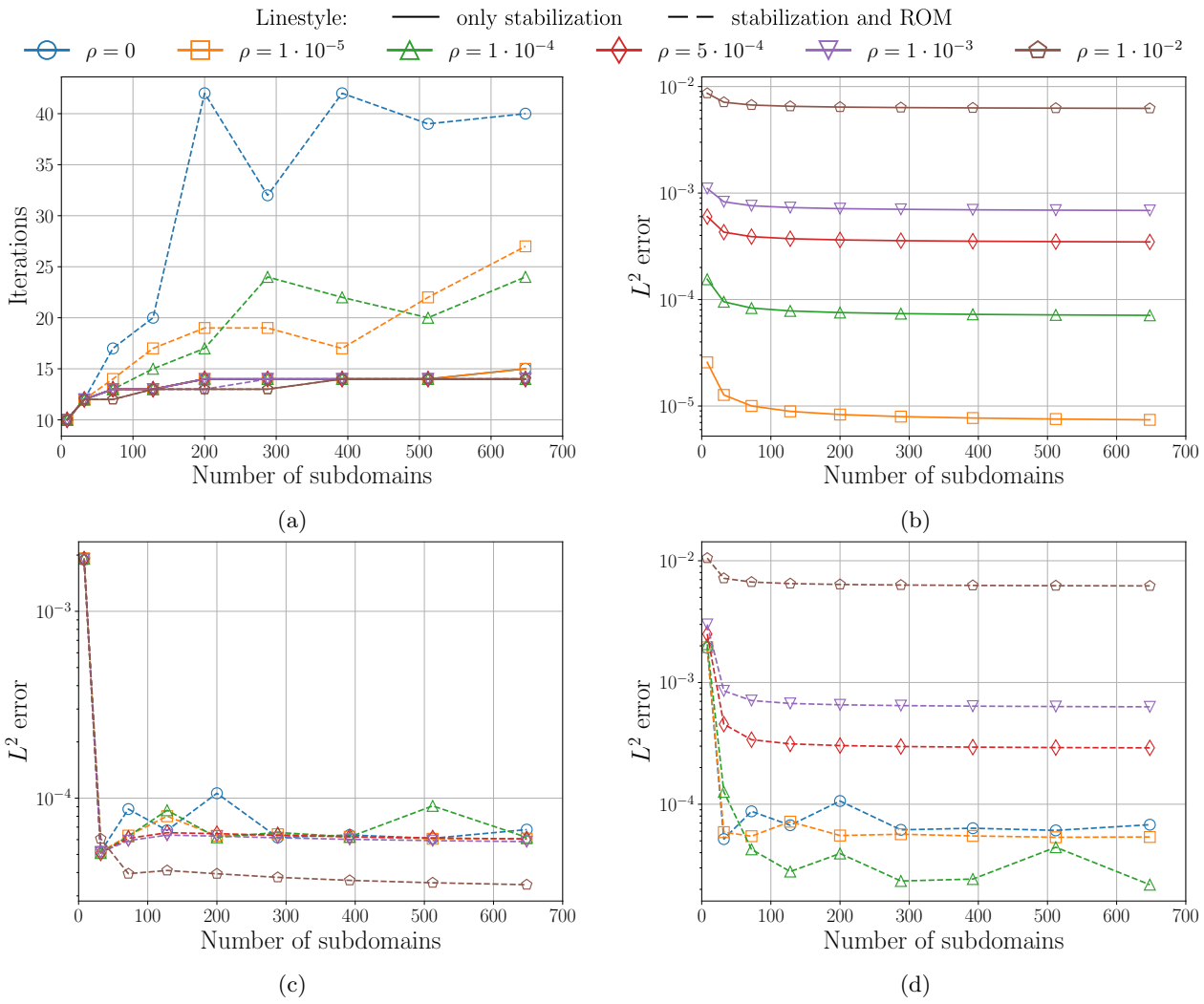


Figure 15: Effect of the stabilization on the solver iterations and the solution error. (a) Iterations with and without ROM for different stabilizations. (b) Error between the original solution and stabilized solution without using ROM. (c) Error between the solution using ROM and without ROM (both with the same stabilization). (d) Error between the original solution and the solution using ROM + stabilization

Effect of the stabilization The following analysis is aimed at demonstrating the effect of stabilization on the different components of the proposed method. Figure 15 summarizes the results and illustrates its interaction with the ROM. It is worth noting that, since the ROM is constructed from the fast assembly tensors, its use is inherently coupled with the fast assembly technique. In contrast, when the ROM is not employed, standard numerical integration is used for matrix assembly.

Figure 15a shows the number of iterations required by the BDDC preconditioner when employing both the ROM and stabilization, and when using stabilization alone. It can be observed that stabilization has a negligible effect on the number of iterations when standard integration is adopted (solid lines). However, when the matrices are assembled through the ROM (dashed lines), increasing the stabilization parameter helps control the number of iterations, bringing it closer to the baseline case, while reducing it can cause the iterations to grow unbounded. This highlights that the stability of the solver can be delicate. Although not reported here, for different microgeometries, the solver performed well when stabilization was applied. However, for some geometries, it did not scale properly in the absence of stabilization, even when no ROM was used.

Figure 15b assesses the consistency error associated with the stabilization by measuring the L^2 norm of the difference between the solutions obtained with and without stabilization. As expected, increasing the stabilization parameter also increases the error. Moreover, for sufficiently large numbers of cells, the error no longer varies as a function of this quantity. In addition, the error levels are approximately proportional to ρ, consistently with the single-cell study of Figure 10: the $O(\rho)$ scaling of the consistency error carries over from the single cell to multi-cell structures.

Figure 15c compares the solution obtained with both ROM and stabilization to the solution obtained with stabilization alone, in order to isolate the contribution of the ROM. It can be observed that, for values below 10^{-3}, this error becomes independent of the stabilization, indicating that it is primarily due to the ROM approximation, whose accuracy is independent from ρ.

Finally, Figure 15d depicts the error obtained by comparing the solution without ROM or stabilization to the solution computed with both ROM and stabilization. It can be observed that, initially, the accuracy is primarily governed by the stabilization, and decreasing ρ leads to a reduction in the error. However, once the accuracy limit of the ROM is reached, the error saturates at approximately the same value, and further reduction of ρ does not yield any additional improvement.

The value chosen for the stabilization parameter in the remainder of this section is $\rho = 5 \cdot 10^{-4}$, which as demonstrated by these results and those of the related paragraph of Section 5.1.1, is sufficient to control the number of solver iterations while maintaining a satisfactory level of accuracy.

More generally, a practical guideline is that ρ should be chosen small enough that the consistency error $O(\rho)$ lies below the desired accuracy at the working polynomial degree p, and large enough to ensure bounded BDDC iterations when the ROM is active. Typical accuracy requirements in structural analysis are of order 10^{-2} or better, so $\rho = 5 \times 10^{-4}$ is comfortably below this threshold while still controlling the iteration count, as suggested by the results in Figure 15a. A sharp analytical criterion, relating ρ explicitly to p, the Jacobian of the geometric map, the trimmed area fraction, or the loading, remains an open question.

5.1.3 Computational performance

This section assesses the computational performance of the proposed solver through three complementary studies: a comparison against representative direct and iterative solvers, the benefit of the proposed acceleration layers against an unaccelerated BDDC baseline, and a scalability study up to 17,000 subdomains. The geometry, materials, loads, and discretization are the same as in Section 5.1.2.

Comparison with other solvers We first compare a basic version of the BDDC method with three widely used solvers for linear systems: the Cholesky factorization, representing a direct method, and the conjugate gradient method combined with either the Successive Over-Relaxation (SOR) preconditioner [82] or an algebraic multigrid (AMG) preconditioner [83], in its smoothed-aggregation variant (GAMG) as implemented in PETSc, representing iterative approaches. All the solvers in this comparison rely on their PETSc implementations, accessed through petsc4py [79, 80], so that the results are not biased by implementation differences. In this comparison, the BDDC method is used without any of the acceleration techniques introduced in Section 4. Furthermore, the stabilization parameter ρ is also set to zero. The performance is evaluated for an

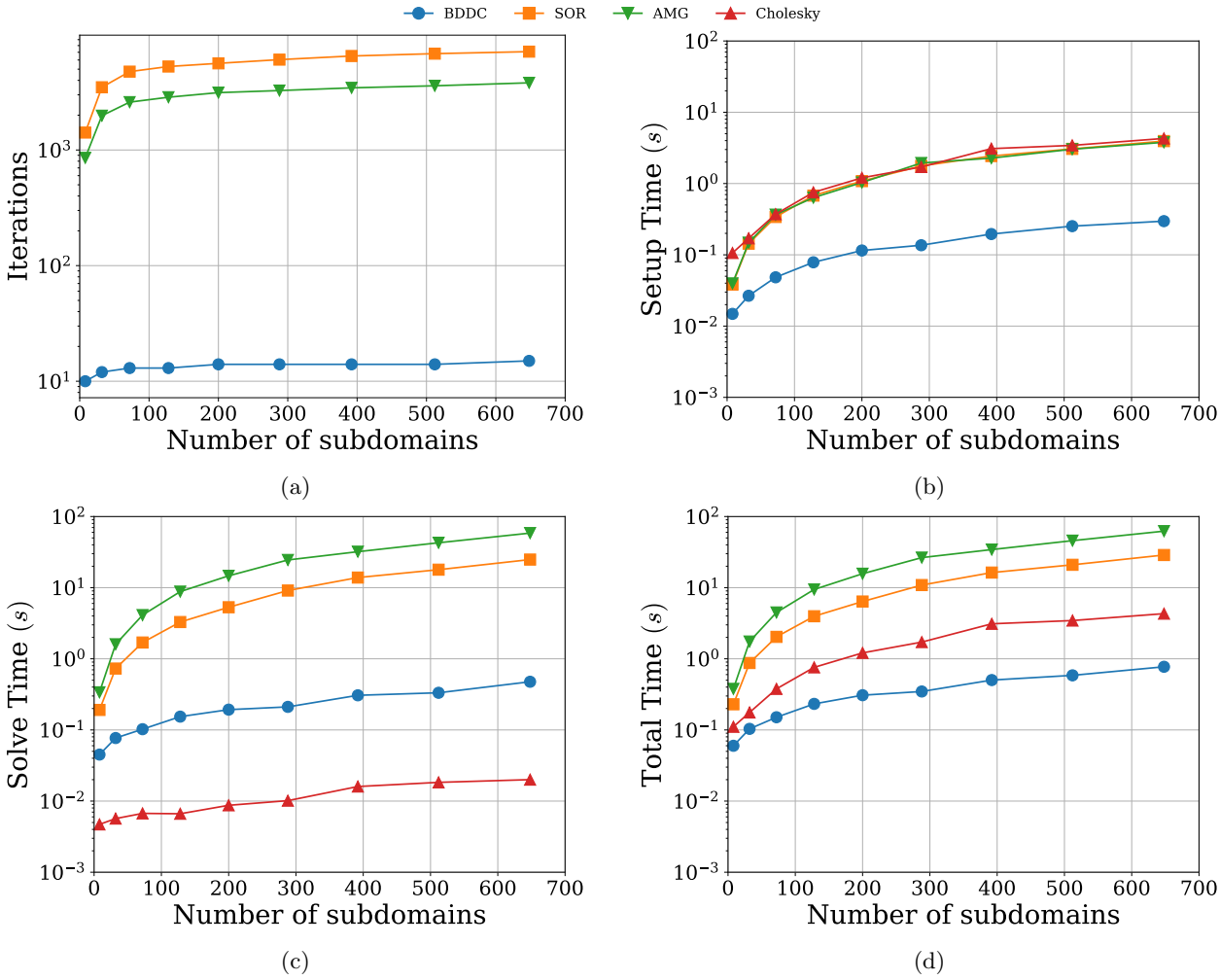


Figure 16: Performances of four different solvers are compared: BDDC, Cholesky decomposition, SOR, and AMG. (a) Number of iterations required by the iterative solvers. (b), (c) Time required for the setup and solve phases, respectively. (d) Total computational time required by the solvers.

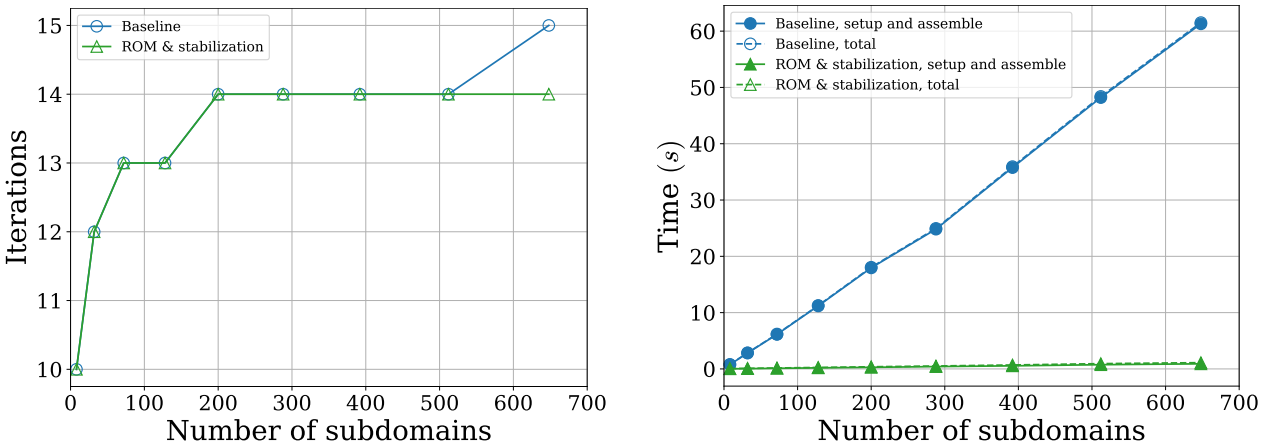
increasing number of cells. The trends for the four solvers are reported in Figure 16. In particular, Figure 16a shows the number of iterations required by the iterative methods, highlighting that BDDC is more efficient than both SOR and AMG by more than two orders of magnitude. It should be noted, however, that the iterations in these approaches are performed on different systems. The SOR and AMG methods operate on the full system given in Equation (30), whereas BDDC is applied only to the condensed system defined in Equation (32).

Regarding computational time, Figures 16(b-d) illustrate the setup time, solve time, and total time for the four solvers. More precisely, the setup time does not include the assembly of the local stiffness matrices, as this cost is common to all methods. Although it represents the main portion of the computational effort, as shown in the remainder of this section, local assembly time can be reduced to a level comparable with setup and solve time by employing the acceleration techniques discussed in Section 4. Therefore, the setup time includes:

- BDDC: construction of the local operators (e.g., Schur complements, restriction/extension matrices) and preparation of the coarse problem solver (assembly of the coarse correction operator S_C and its Cholesky factorization);
- Cholesky: assembly of the global matrix and parallel computation of its Cholesky factorization;

- AMG: assembly of the global matrix and construction of the multigrid hierarchy (coarse levels, transfer operators, and smoothers);
- SOR: assembly of the global matrix and standard preprocessing required by the SOR method.

The most striking feature of these results is that BDDC outperforms all the alternative methods in terms of total computational time, being approximately one order of magnitude faster than the Cholesky decomposition and about two orders of magnitude faster than SOR and AMG. More specifically, for the Cholesky decomposition, the majority of the computational cost is associated with the setup phase, while the solution phase is comparatively fast, as expected for a direct method. In contrast, for SOR and AMG, most of the computational effort is concentrated in the solution phase due to the large number of iterations required. In this regard, it is worth noting that, although AMG requires fewer iterations than SOR, each of its iterations is more expensive, as it involves the application of a full multigrid cycle, resulting in an overall longer solve time. Moreover, the relatively poor performance of AMG is not unexpected, since standard algebraic multigrid methods are known to lose efficiency for high-order discretizations, and the ill-conditioning introduced by trimming further degrades the effectiveness of the coarsening and smoothing procedures. For BDDC, the setup and solution times are of comparable magnitude, leading to a more balanced computational cost distribution.



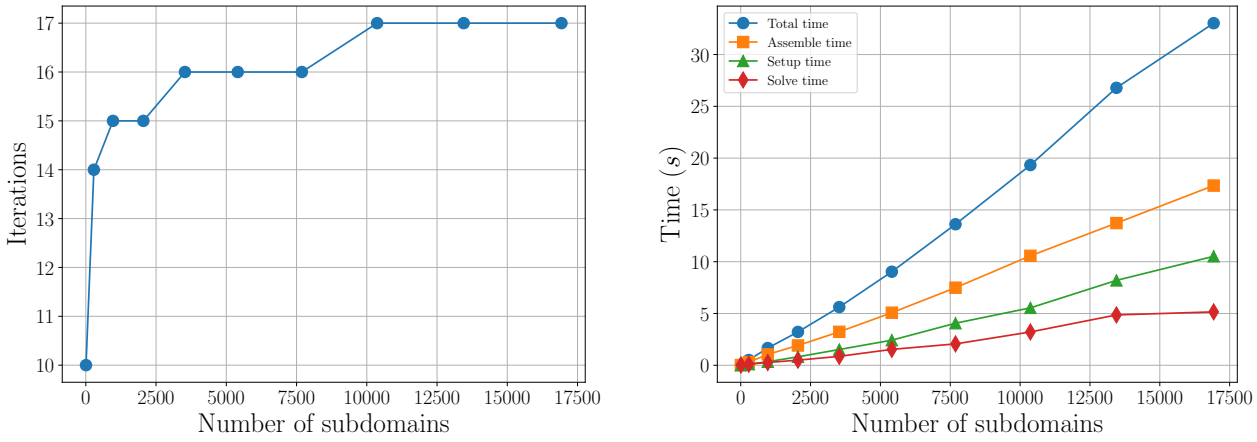
(a) (b)

Figure 17: Iterations (a) and computational time (b) associated with the solution of the problem using a baseline BDDC approach and using the BDDC approach proposed here.

Efficiency of the accelerations We now turn to the efficiency of the proposed acceleration techniques with respect to the full integration approach. The performance is evaluated in terms of the number of iterations, setup time, and total computational time, as functions of the number of subdomains.

Figure 17a shows the number of iterations for both the baseline case and the proposed method. Here, the term baseline refers to the use of full integration, i.e., no fast assembly technique and no ROM are employed. Additionally, stabilization is not included in the baseline configuration. It can be observed that there is no significant difference in the number of iterations between the baseline and the proposed method, except for the last data point, where the baseline requires one additional iteration.

In contrast, a substantial difference emerges when considering the setup and total computational times, as shown in Figure 17b. In the baseline case, the majority of the computational cost is associated with the setup phase (that here includes also the assemble time), which is expected since full numerical integration must be performed. These results confirm the superiority of the proposed techniques with respect to a standard approach in terms of computational time.



(a) (b)

Figure 18: Iterations (a) and computational time (b) as a function of number of cells. In (b), the computational time is decomposed into assembly time, setup time, solve time, and total time (approximately equal to the sum of the former components). The test is performed up to a large number of cells, specifically 17,000.

Scalability of the method To examine scalability, the method is evaluated for increasingly large numbers of cells in the geometry, reaching up to 17,000 subdomains. From Figure 18a, it can be seen that the number of iterations slightly increases with the number of cells, however reaching a constant value of 17.

Figure 18b reports the different components of the computational time, namely: the assembly time that includes the construction of the local matrices, the setup time associated with the computation of all quantities required by the BDDC method, the solve time corresponding to the iterative phase, and the total time, which is approximately the sum of the previous contributions. From this figure, it can be observed that the solve phase is the fastest component, being approximately twice as fast as the setup phase, which in turn is about twice as fast as the assembly phase. The overall computational complexity of the different components with respect to the number of cells appears to exhibit a linear trend. This is expected for problems that are not dominated by the size of the coarse correction. Remarkably, the most demanding analysis including 17,000 cells was still performed in approximately 30 seconds in a standard laptop (online phase only; see Section 4.3 for the offline training cost).

For completeness, the problem size and statistics for the largest test are reported here. The total number of non-redundant degrees of freedom is $n_I + n_U = 2{,}169{,}728$, of which $n_U = 510{,}784$ belong to the skeleton and $n_I = 1{,}658{,}944$ are internal. The coarse problem has $n_C = 136{,}344$ degrees of freedom and is solved with a direct Cholesky factorization; its diagonally scaled condition number is $\kappa(S_C) = 6.0 \times 10^6$, which induces a relative error of at most $\kappa(S_C)\,\varepsilon \approx 10^{-9}$ in the coarse solve (ε being the double-precision machine epsilon), well below the solver tolerance of 10^{-8}, and therefore does not affect the robustness of the preconditioner. The peak resident memory is 30 GB (aggregate over 8 processes). It should be noted that this scalability test is performed on a structured quarter-annulus geometry, which is more regular than the application examples of Section 5.2; nevertheless, the latter exhibit a comparable iteration count. More challenging configurations, such as highly graded geometries or severe trimming, may increase the BDDC iteration count; their systematic assessment is left for future work.

Parallel efficiency and cost distribution So far the scalability has been assessed with respect to the number of subdomains at a fixed number of cores; we now turn to the parallel behaviour. To this end, a weak-scaling study, which is the natural measure of scalability for a domain-decomposition method, was carried out on a shared-memory server equipped with four Intel Xeon Gold 6148 sockets (20 cores per socket, 80 physical cores, four NUMA nodes, 1 TB of RAM). Each MPI rank is pinned to a distinct physical core with node-local memory, using a single computational thread per rank, and the reported times are the fastest of three repetitions

Table 4 reports the weak scaling at a fixed load of approximately 2,048 subdomains per core, up to

131,072 subdomains and 1.68×10^7 degrees of freedom on 64 cores, over which the coarse problem grows proportionally from 16,704 to 1,051,136 degrees of freedom. Despite this 64× growth, the BDDC iteration count stays between 15 and 18, the peak memory per rank remains 4.4–5.4 GB, and the direct coarse solve stays below 1 s, so the coarse problem does not become the bottleneck; this confirms the scalability of the coarse space. The parallel efficiency nonetheless decreases from 88% on 2 cores to 19% on 64 cores, and the distribution of costs reveals the origin. The coarse solve ($\leq$ 1 s) and the inter-process communication ($\leq$ 17 s) remain small throughout; the growth is concentrated in the solve phase and is dominated by the vector operations of the interface conjugate-gradient iteration, which reach about 160 s of the 196 s solve on 64 cores. In the present implementation these operations are carried out redundantly on every process over the full interface vector, so that their cost scales with the global problem size; a distributed Krylov solver for the interface problem would remove them. A secondary contribution to the efficiency loss is the saturation of the per-socket memory bandwidth in the setup and assembly phases once more than about four ranks share a socket.

We emphasise that the parallel implementation is a research prototype and that its optimisation lies beyond the scope of this work, which concerns the overall solution strategy rather than parallel-implementation tuning. The principal route to improved efficiency at large core counts is a distributed Krylov solver for the interface problem, which would eliminate the redundant per-process vector operations identified above; at substantially larger scale a parallel or multilevel coarse solve would also become relevant. We finally observe that, in three dimensions, the per-cell workload grows substantially (the local problems scale as $(p+1)^3$ rather than $(p+1)^2$), so that each subdomain carries considerably more computation; this higher granularity is expected to improve the parallel efficiency of the dominant per-subdomain phases.

Table 4: Weak scaling at a fixed load of $\approx$ 2,048 subdomains per core on the four-socket Xeon server: number of subdomains, total (non-redundant) and coarse degrees of freedom, BDDC iteration count, the setup/assembly/solve decomposition of the wall time with the coarse-solve and communication contributions (both contained in the solve phase) listed separately, and the weak-scaling efficiency $E_w(p) = T(1)/T(p)$, expressed as a percentage.

Cores	Subdom.	DOFs	Coarse DOFs	Iters	Setup [s]	Asm. [s]	Solve [s]	Coarse [s]	Comm. [s]	Total [s]	Eff.
1	2,048	263,168	16,704	15	37.1	17.1	10.6	0.09	0.01	64.8	100%
2	4,232	543,168	34,316	16	40.7	19.4	13.4	0.14	0.82	73.5	88%
4	8,192	1,050,624	66,176	16	42.8	19.4	15.0	0.17	1.20	77.2	84%
8	16,200	2,076,480	130,500	17	44.8	20.5	21.3	0.23	1.53	86.6	75%
16	32,768	4,198,400	263,424	17	48.1	24.2	38.8	0.30	3.12	111.1	58%
32	66,248	8,485,568	531,804	17	56.2	30.4	86.9	0.52	5.95	173.4	37%
64	131,072	16,785,408	1,051,136	18	94.5	46.7	195.6	0.99	17.03	336.8	19%

5.2 Application examples

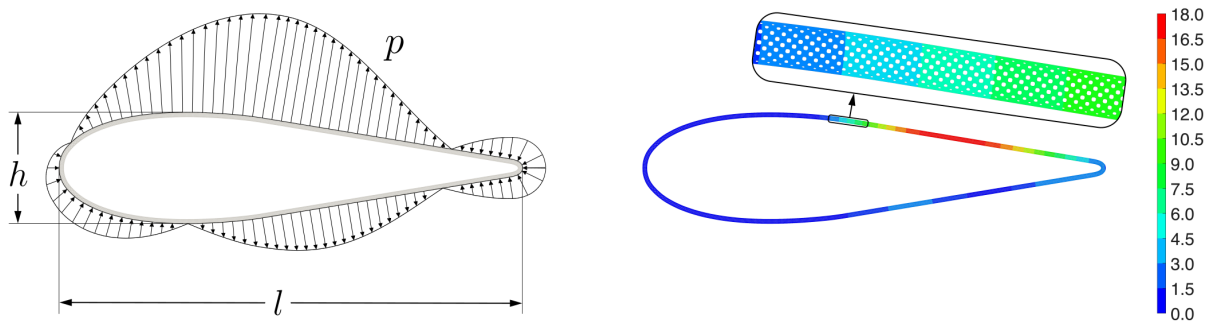
We now demonstrate the effectiveness of the proposed method on application-oriented examples. In particular, we consider two scenarios: a sandwich wing profile with a lattice core, and a wrench featuring a lattice structure.

5.2.1 Sandwich wing

The first application considers the analysis of a 2D wing profile featuring a sandwich structure, which is composed of a continuous external layer and a porous inner core. The geometry of the wing is depicted in Figure 19a, and it is characterized by a chord length of $l = 10$ m, a maximum height of $h = 220$ mm, and a profile thickness of 10 mm.

The load applied to the structure, also illustrated in Figure 19a, simulates a typical aerodynamic relative pressure distribution over a wing profile with a maximum value of $p_{\max} = 4.5$ N/mm. The structure is clamped at the nodes where the internal part of the wing section would be in contact with the main spar, representing a realistic constraint for a wing structure.

The domain is obtained by combining a total of 4,000 cells. The internal lattice structure is generated using a Schwarz Diamond level-set, whose threshold parameters are kept constant along the chord direction



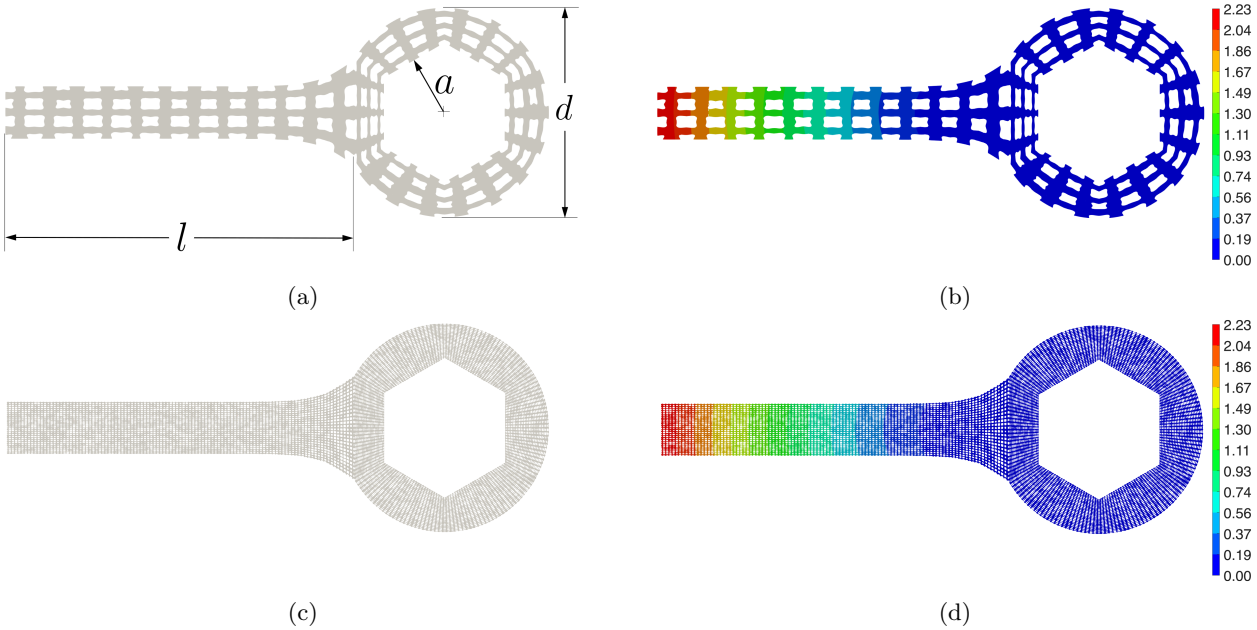
(a) (b)

Figure 19: (a) Geometry of the sandwich wing example in Section 5.2.1. (b) Contour of the magnitude of the displacement field with a zoom on the wing profile. Displacements are expressed here in mm.

but are varied through the thickness within the interval $\mathbb{P}_1^{SD} = [0.1, 1]$. This specific selection of parameters ensures the creation of a continuous, solid external boundary, which is a critical requirement for aerodynamic performance. On the other hand, it produces a porous internal core, which is a characteristic of sandwich structures designed for lightweight applications.

The stabilization parameter for this analysis was chosen as $\rho = 1{\cdot}10^{-4}$. It is worth mentioning that despite the geometric complexity of the problem, the BDDC solver demonstrated excellent performance, reaching convergence in just 18 iterations. Figure 19b presents a colormap of the magnitude of the displacement field superimposed, illustrating the structural response under the applied aerodynamic load. The internal structure of the sandwich wing is further illustrated in the zoomed view within the same figure.

5.2.2 Lattice wrench



(a) (b)

(c) (d)

Figure 20: Geometry of lattice structure representing a wrench as described in Section 5.2.2 for the coarse (a) and the dense (c) lattice structures. Contour of the magnitude of the displacement field expressed in mm for the coarse (b) and the dense (d) lattice structures.

In this second example, we consider a geometry that does not derive from a standard tensor-product-

based disposition of cells. Specifically, we consider the wrench depicted in Figure 20(a), partitioned into 90 subdomains. The global geometric features of the wrench are the handle length of $l = 50$ mm, the apothem of the hexagonal hole of $a = 8.7$ mm, and the external diameter of $d = 60$ mm. The lattice micro-structure within each subdomain is defined by a Schoen IWP level-set, with threshold parameters chosen within the interval $[-2.5, 2.5]^4$. The threshold parameters are drawn randomly, independently for each cell, from a uniform distribution over this interval; the random seed is fixed in the corresponding scripts (see Table 3), so that the lattice geometries and the reported results are exactly reproducible.

The mechanical behavior of the structure is modeled with a Young's modulus $E = 5$ N/mm and a Poisson's ratio $\nu = 0.25$. To simulate the action of the wrench, a homogeneous Dirichlet boundary condition is imposed on the interior hexagonal boundary of the wrench head, representing the fixed grip on a bolt. A downward traction force $\mathbf{f} = -1$ N/mm $\boldsymbol{e}_2$ is applied as a Neumann boundary condition on the vertical left side of the handle, while homogeneous Neumann boundary conditions are enforced on the remaining portions of the boundary.

The stabilization parameter is chosen as $\rho = 5 \cdot 10^{-4}$. Despite the non-trivial layout of the subdomains and the variability of the random lattice parameters, the proposed BDDC solver proves to be highly robust, achieving convergence in only 15 iterations. The resulting displacement field u_h is illustrated in Figure 20(b), exhibiting the expected bending behavior of the loaded wrench.

To assess the accuracy of the computed solution, a reference solution is built by discretizing every cell of the wrench with a grid of $k \times k$ rectangular elements of degree $p = 3$ within a CutFEM framework [81, 45], with k up to 6. The level-set of each element describes the corresponding portion of the cell level-set, so that the lattice geometry is exactly the same for every k. These reference solutions employ full integration, no ROM acceleration, and a direct solver. Two quantities of interest are monitored: the compliance and the mean displacement of the loaded boundary. Since the traction is interpolated at the degrees of freedom of the loaded boundary, the resultant of the discrete load varies slightly with the discretization. For this reason, the compliance is normalized by the square of the resultant of the discrete load, and the mean displacement by the resultant itself.

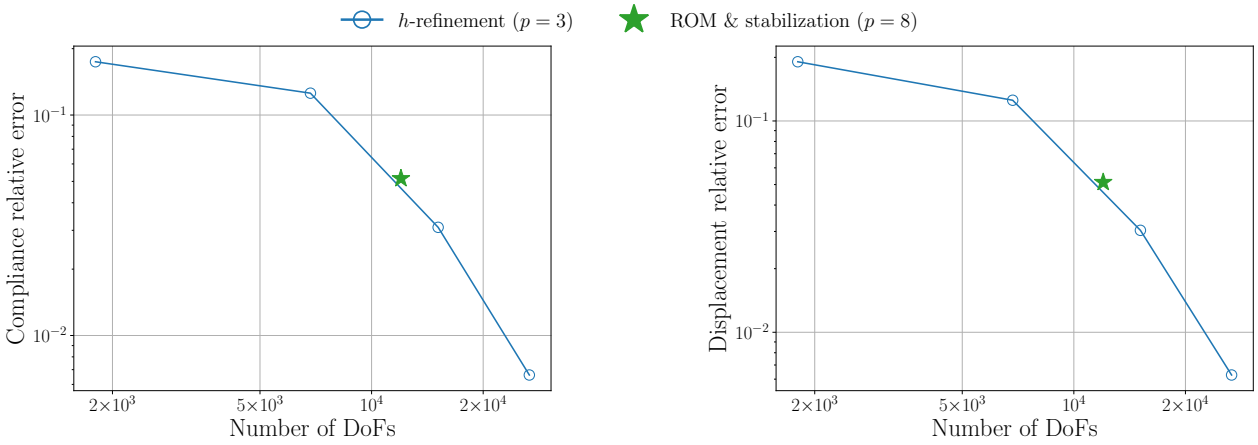


Figure 21: Relative error of the normalized compliance (a) and of the normalized mean displacement of the loaded boundary (b) for the wrench example. In each plot, the blue curve corresponds to the h-refined solutions, obtained with a grid of $k \times k$ rectangular CutFEM elements of degree $p = 3$ per cell, with $k = 1, 2, 3, 4$. The star corresponds to the proposed spectral method, with a single element of degree $p = 8$ per cell, combined with the ROM acceleration, the α-stabilization, and the BDDC solver. The errors are measured against the finest h-refined solution ($k = 6$, corresponding to 59,400 DoFs).

Figure 21 reports the relative error of both quantities with respect to the finest reference solution, which counts 59,400 DoFs. The reference solutions converge monotonically as k increases, with errors decreasing from about 17% for $k = 1$ down to 0.7% for $k = 4$. The result of the proposed method, which combines the ROM acceleration, the α-stabilization, and the BDDC solver with degree $p = 8$, lies on the same error curve,

with an error close to 5% for 12,000 DoFs. The impact of the individual ingredients of the method on both quantities is small compared to the discretization error. Replacing the ROM by full integration changes both quantities by $1.4 \cdot 10^{-4}$; removing the α-stabilization changes them by $5.5 \cdot 10^{-3}$; and replacing the BDDC solver by a direct factorization changes them by $2.5 \cdot 10^{-10}$.

To further confirm the scalability observed in Section 5.1.3, the similar wrench geometry shown in Figure 20(c) is considered, this time obtained as a lattice structure consisting of 7,290 subdomains, while all other properties remain unchanged. The BDDC solver remains robust and converges in a comparable number of iterations (18). The resulting displacement field is shown in Figure 20(d).

6 Conclusions

In this work, we presented a novel domain decomposition method for fast simulation of large two-dimensional lattice structures geometrically described through implicit functions, without relying on homogenization or multiscale approaches. Individual lattice cells, described through level-set functions that may change from cell to cell, are mapped using arbitrary order mappings (in a finite element spirit), allowing the creation of complex graded designs.

In order to solve such designs efficiently and with low memory requirements, without the need for homogenization or multiscale techniques, which rely on assumptions such as scale separation and periodicity, we use domain decomposition methods. Specifically, the proposed framework integrates a Balanced Domain Decomposition by Constraints (BDDC) solver with high-order unfitted discretizations, which naturally handle level-set geometric descriptions. This strategy exploits the geometric similarities between lattice cells by building a reduced order model (ROM) surrogate that assembles the cell stiffness matrices in a fraction of the time and without the need for expensive quadrature rules. The different hypotheses and approximations introduced in the solver have been assessed through numerical experiments to validate the proposed framework.

This methodology is complemented with a α-stabilization term, similar to the one applied in finite cell methods, required by the unfitted nature of the discretization. When using ROM techniques, such stabilization is required to obtain a scalable solver with respect to the number of subdomains. However, it breaks the consistency of the problem, introducing an error in the solution which remains small for moderate stabilization parameters. As reported in the numerical experiments, the number of iterations remains asymptotically bounded as the ratio H/h (subdomain versus mesh size) is kept constant, in agreement with the scalability properties of BDDC methods. We demonstrated its performance on a 2D problem with 17,000 varying cell geometries, solving it in approximately 30 seconds on an off-the-shelf laptop. A dedicated weak-scaling study further confirms this behaviour, the BDDC iteration count remaining bounded (between 15 and 18) as the problem is scaled up to 131,072 subdomains and 1.68×10^7 degrees of freedom on 64 cores.

Such a solver could constitute a solid foundation for use in conjunction with optimization algorithms: it allows forward problems to be solved quickly, and the ROM surrogate for stiffness matrix assembly makes it straightforward to compute the gradient of the solution with respect to the design parameters.

We will also investigate alternative ROM surrogates for 3D problems. The MDEIM coefficient interpolation algorithm proposed in this work relies on one parameter coefficient per interpolation mesh vertex, which in hexahedral meshes amounts to at least 8 coefficients per cell. In view of the additional complexity associated with a three-dimensional extension of the present framework, two possible research directions can be considered: the use of neural network based surrogates that do not suffer the curse of dimensionality; and the discretization of the porosity field on tetrahedral meshes, which have fewer vertices per cell and thus lead to lower-dimensional parameter spaces. Beyond the construction of the surrogates themselves, we plan to explore their use as approximate subdomain inverses for building efficient preconditioners. In this setting, the ROM-generated subdomain stiffness matrices would be applied directly inside a Krylov solver without the need to invert them, which would also reduce the need for the stabilization introduced in this work. Finally, we plan to extend this framework to support more complex boundary conditions, to the case of hyperelastic materials under large deformations, and to the simulation of plates.

Acknowledgments

The authors acknowledge the financial support of the Swiss National Science Foundation through the project FLAS$_h$ (200021_214987). The authors would like to acknowledge the support of Raul Rubio for his valuable scientific contributions, particularly in the area of domain decomposition.

Declaration of generative AI and AI-assisted technologies in the writing process: During the preparation of this work, the authors used ChatGPT, Gemini, and Claude in order to improve language and readability. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

A Algorithms

In this appendix, some of the algorithms employed in the method are presented to provide additional clarity and to assist the reader in navigating its various aspects.

Algorithm 1 Application of operator S_{BDDC}^{-1} to v (parallelizable over subdomains i).

Input: vector $v \in \mathbb{R}^{n_U}$
Output: vector $w \in \mathbb{R}^{n_U}$
$w \leftarrow 0 \in \mathbb{R}^{n_U}$
for all $i \in [1, \dots, n_c]$ **do** ▷ Executed in parallel
    Extract subvector $v^{(i)}$ from v
    Solve $\begin{bmatrix} S^{(i)} & C^{(i)^\top} \\ C^{(i)} & 0 \end{bmatrix} \begin{bmatrix} x^{(i)} \\ \lambda^{(i)} \end{bmatrix} = \begin{bmatrix} D^{(i)} v^{(i)} \\ 0 \end{bmatrix}$ ▷ Using Algorithm 2
    Prolong $D^{(i)} x^{(i)}$ to x
    Compute $w = w + x$
end for
$z \leftarrow 0 \in \mathbb{R}^{n_c}$
for all $i \in [1, \dots, n_c]$ **do** ▷ Executed in parallel
    Extract subvector $v^{(i)}$ from v
    Prolong $\Psi^{(i)\top} D^{(i)\top} v^{(i)}$ to y
    Compute $z = z + y$
end for
Solve $S_C g = z$ ▷ Executed with a parallel sparse Cholesky solver
for all $i \in [1, \dots, n_c]$ **do** ▷ Executed in parallel
    Extract subvector $g^{(i)}$ from g
    Prolong $D^{(i)} \Psi^{(i)} g^{(i)}$ to x
    Compute $w = w + x$
end for
return w

Algorithm 2 Local solve of the problem $\begin{bmatrix} S^{(i)} & C^{(i)^\top} \\ C^{(i)} & 0 \end{bmatrix} \begin{bmatrix} x^{(i)} \\ \lambda^{(i)} \end{bmatrix} = \begin{bmatrix} f^{(i)} \\ g^{(i)} \end{bmatrix}$

Input: vector $f^{(i)} \in \mathbb{R}^{n_U^{(i)}}$, vector $g^{(i)} \in \mathbb{R}^{n_C^{(i)}}$
Output: vector $x^{(i)} \in \mathbb{R}^{n_U^{(i)}}$, vector $\lambda^{(i)} \in \mathbb{R}^{n_C^{(i)}}$.
Remove the rows and columns associated with cross-point coarse DoFs in $C^{(i)}$, $x^{(i)}$, $\lambda^{(i)}$, $f^{(i)}$, $g^{(i)}$, and $S^{(i)}$, yielding the system

$$\begin{bmatrix} \hat{S}^{(i)} & \hat{C}^{(i)^\top} \\ \hat{C}^{(i)} & 0 \end{bmatrix} \begin{bmatrix} \hat{x}^{(i)} \\ \hat{\lambda}^{(i)} \end{bmatrix} = \begin{bmatrix} \hat{f}^{(i)} \\ \hat{g}^{(i)} \end{bmatrix}.$$

Compute $\hat{S}^{(i)-1}\hat{C}^{(i)^\top}$ ▷ Through solve of $\hat{S}^{(i)}X = \hat{C}^{(i)^\top}$
Compute $\hat{t}^{(i)} = \hat{C}^{(i)}\hat{S}^{(i)-1}\hat{f}^{(i)} - \hat{g}^{(i)}$ ▷ Through solve of $\hat{S}^{(i)}x = \hat{f}^{(i)}$
Solve $\hat{T}^{(i)}\hat{\lambda}^{(i)} = \hat{t}^{(i)}$
Compute $\hat{h}^{(i)} = \hat{f}^{(i)} - \hat{C}^{(i)^\top}\hat{\lambda}^{(i)}$
Compute $\hat{x}^{(i)} = \hat{S}^{(i)-1}\hat{h}^{(i)}$ ▷ Through solve of $\hat{S}^{(i)}x = \hat{h}^{(i)}$
Assemble $x^{(i)}$ and $\lambda^{(i)}$ by reversing step 5
return $x^{(i)}$, $\lambda^{(i)}$

Algorithm 3 Select magic points

Input: matrix $U_r \in \mathbb{R}^{n_i \times n_r}$
Output: set of indices $\{m_1, \ldots, m_{n_r}\} \in \mathbb{N}$
for $j = 1$ to n_r **do**
  if $j = 1$ **then**
    $m_1 \leftarrow \arg\max_i \left| U_{r\,1}^{[i]} \right|$
  else
    Compute projection of $U_{r\,j}$ onto $\text{span}\{U_{r\,1}, \ldots, U_{r\,j-1}\} : U_{p\,j} = \sum_{k=1}^{j-1} U_{r\,k} U_{r\,k}^\top U_{r\,j}$
    $m_j \leftarrow \arg\max_i \left| U_{r\,j}^{[i]} - U_{p\,j}^{[i]} \right|$
  end if
end for
return $m_1, m_2, \ldots, m_{n_r}$

Algorithm 4 Assembly of the local stiffness matrix

Input: tensor function $\hat{C}_{i_1 j_1 i_2 j_2}(\boldsymbol{\xi})$, threshold vector $\boldsymbol{\mu}_0$, stabilization parameter ρ
Output: stiffness tensor $K_{i_1 i_2 k_1 k_2}$
Compute $\mathrm{C}_{i_1 j_1 i_2 j_2 k_3}$ by interpolating $\hat{C}_{i_1 j_1 i_2 j_2}(\boldsymbol{\xi})$
Obtain $I(\boldsymbol{\mu}_0)$ ▷ Using Algorithm 5
Obtain $\mathcal{I}_{j_1 j_2}^{k_1 k_2 k_3} \leftarrow \text{Reshape}(I)$
Obtain $\tilde{\mathcal{I}}_{j_1 j_2}^{k_1 k_2 k_3} \leftarrow \int_{\hat{\Pi}} \frac{\partial \mathscr{B}_{k_1}}{\partial \xi_{j_1}} \frac{\partial \mathscr{B}_{k_2}}{\partial \xi_{j_2}} \mathscr{L}_{k_3} \, \mathrm{d}\hat{\Omega} - \mathcal{I}_{j_1 j_2}^{k_1 k_2 k_3}$ ▷ $\int_{\hat{\Pi}} \frac{\partial \mathscr{B}_{k_1}}{\partial \xi_{j_1}} \frac{\partial \mathscr{B}_{k_2}}{\partial \xi_{j_2}} \mathscr{L}_{k_3} \, \mathrm{d}\hat{\Omega}$ is precomputed
Obtain $K_{i_1 i_2 k_1 k_2} \leftarrow \left(\mathcal{I}_{j_1 j_2}^{k_1 k_2 k_3} + \rho \cdot \tilde{\mathcal{I}}_{j_1 j_2}^{k_1 k_2 k_3} \right) \mathrm{C}_{i_1 j_1 i_2 j_2 k_3}$
return $K_{i_1 i_2 k_1 k_2}$

Algorithm 5 MDEIM

Input: list(tensor $U_r \in \mathbb{R}^{n_i \times n_r}$), list(reduced-order functions $I_r(\boldsymbol{\mu}) : \mathbb{R}^4 \to \mathbb{R}^r$), threshold vector $\boldsymbol{\mu}_0$
Output: vector $I \in \mathbb{R}^{n_i}$
$i_k \leftarrow \text{FindCluster}(\boldsymbol{\mu}_0)$
$I \leftarrow U_r^{i_k} I_r^{i_k}(\boldsymbol{\mu}_0)$
return I

References

[1] T. D. Ngo, A. Kashani, G. Imbalzano, K. T. Nguyen, D. Hui, Additive Manufacturing (3D Printing): A Review of Materials, Methods, Applications and Challenges, Composites Part B: Engineering 143 (2018) 172–196.

[2] A. Vyatskikh, S. Delalande, A. Kudo, X. Zhang, C. M. Portela, J. R. Greer, Additive Manufacturing of 3D Nano-Architected Metals, Nature communications 9 (1) (2018) 1–8.

[3] C. Pan, Y. Han, J. Lu, Design and Optimization of Lattice Structures: A Review, Applied Sciences 10 (18) (2020) 6374.

[4] X. Zheng, H. Lee, T. H. Weisgraber, M. Shusteff, J. DeOtte, E. B. Duoss, J. D. Kuntz, M. M. Biener, Q. Ge, J. A. Jackson, et al., Ultralight, Ultrastiff Mechanical Metamaterials, Science 344 (6190) (2014) 1373–1377.

[5] J. Bauer, S. Hengsbach, I. Tesari, R. Schwaiger, O. Kraft, High-Strength Cellular Ceramic Composites with 3D Microarchitecture, Proceedings of the National Academy of Sciences 111 (7) (2014) 2453–2458.

[6] S. C. Han, J. W. Lee, K. Kang, A New Type of Low Density Material: Shellular, Advanced Materials 27 (37) (2015) 5506–5511.

[7] L. R. Meza, A. J. Zelhofer, N. Clarke, A. J. Mateos, D. M. Kochmann, J. R. Greer, Resilient 3D Hierarchical Architected Metamaterials, Proceedings of the National Academy of Sciences 112 (37) (2015) 11502–11507.

[8] A. J. D. Shaikeea, H. Cui, M. O'Masta, X. R. Zheng, V. S. Deshpande, The Toughness of Mechanical Metamaterials, Nature Materials 21 (3) (2022) 297–304.

[9] X. Ren, R. Das, P. Tran, T. D. Ngo, Y. M. Xie, Auxetic Metamaterials and Structures: A Review, Smart materials and structures 27 (2) (2018) 023001.

[10] T. Tancogne-Dejean, A. B. Spierings, D. Mohr, Additively-Manufactured Metallic Micro-Lattice Materials for High Specific Energy Absorption under Static and Dynamic Loading, Acta Materialia 116 (2016) 14–28.

[11] S. Shan, S. H. Kang, J. R. Raney, P. Wang, L. Fang, F. Candido, J. A. Lewis, K. Bertoldi, Multistable Architected Materials for Trapping Elastic Strain Energy, Advanced Materials 27 (29) (2015) 4296–4301.

[12] J. U. Surjadi, L. Gao, H. Du, X. Li, X. Xiong, N. X. Fang, Y. Lu, Mechanical Metamaterials and Their Engineering Applications, Advanced Engineering Materials 21 (3) (2019) 1800864.

[13] D. M. Kochmann, J. B. Hopkins, L. Valdevit, Multiscale Modeling and Optimization of the Mechanics of Hierarchical Metamaterials, MRS Bulletin 44 (10) (2019) 773–781.

[14] M. Buck, O. Iliev, H. Andrä, Multiscale Finite Element Coarse Spaces for the Application to Linear Elasticity, Open Mathematics 11 (4) (2013) 680–701.

[15] T. Y. Hou, X.-H. Wu, A Multiscale Finite Element Method for Elliptic Problems in Composite Materials and Porous Media, Journal of computational physics 134 (1) (1997) 169–189.

[16] N. Castelletto, H. Hajibeygi, H. A. Tchelepi, Multiscale Finite-Element Method for Linear Elastic Geomechanics, Journal of Computational Physics 331 (2017) 337–356.

[17] M. Doškář, J. Zeman, P. Krysl, J. Novák, Microstructure-Informed Reduced Modes Synthesized with Wang Tiles and the Generalized Finite Element Method, Computational Mechanics 68 (2) (2021) 233–253.

[18] D. Savvas, G. Stefanou, M. Papadrakakis, G. Deodatis, Homogenization of Random Heterogeneous Media with Inclusions of Arbitrary Shape Modeled by XFEM, Computational mechanics 54 (5) (2014) 1221–1235.

[19] T. Strouboulis, I. Babuška, K. Copps, The Design and Analysis of the Generalized Finite Element Method, Computer methods in applied mechanics and engineering 181 (1-3) (2000) 43–69.

[20] F. Feyel, A Multilevel Finite Element Method (FE2) to Describe the Response of Highly Non-Linear Structures Using Generalized Continua, Computer Methods in applied Mechanics and engineering 192 (28-30) (2003) 3233–3244.

[21] J. Schröder, A Numerical Two-Scale Homogenization Scheme: the FE 2-Method, in: Plasticity and beyond, Springer, 2014, pp. 1–64.

[22] N. Charalambakis, Homogenization Techniques and Micromechanics: A Survey and Perspectives, Applied Mechanics Reviews 63 (3) (2010).

[23] H. Moulinec, P. Suquet, A FFT-Based Numerical Method for Computing the Mechanical Properties of Composites from Images of Their Microstructures, in: IUTAM symposium on microstructure-property interactions in composite materials, Springer, 1995, pp. 235–246.

[24] J. Zeman, J. Vondřejc, J. Novák, I. Marek, Accelerating a FFT-Based Solver for Numerical Homogenization of Periodic Media by Conjugate Gradients, Journal of Computational Physics 229 (21) (2010) 8065–8071.

[25] V. Müller, M. Kabel, H. Andrä, T. Böhlke, Homogenization of Linear Elastic Properties of Short-Fiber Reinforced Composites—A Comparison of Mean Field and Voxel-Based Methods, International Journal of Solids and Structures 67 (2015) 56–70.

[26] R. N. Glaesener, E. A. Träff, B. Telgen, R. M. Canonica, D. M. Kochmann, Continuum Representation of Nonlinear Three-Dimensional Periodic Truss Networks by On-The-Fly Homogenization, International Journal of Solids and Structures 206 (2020) 101–113.

[27] P. Onck, E. Andrews, L. Gibson, Size Effects in Ductile Cellular Solids. Part I: Modeling, International Journal of Mechanical Sciences 43 (3) (2001) 681–699.

[28] M. Yoder, L. Thompson, J. Summers, Size Effects in Lattice Structures and a Comparison to Micropolar Elasticity, International Journal of Solids and Structures 143 (2018) 245–261.

[29] C. M. Portela, J. R. Greer, D. M. Kochmann, Impact of Node Geometry on the Effective Stiffness of Non-Slender Three-Dimensional Truss Lattice Architectures, Extreme Mechanics Letters 22 (2018) 138–148.

[30] M. Jamshidian, N. Boddeti, D. W. Rosen, O. Weeger, Multiscale Modelling of Soft Lattice Metamaterials: Micromechanical Nonlinear Buckling Analysis, Experimental Verification, and Macroscale Constitutive Behaviour, International Journal of Mechanical Sciences 188 (2020) 105956.

[31] O. Weeger, Isogeometric sizing and shape optimization of 3D beams and lattice structures at large deformations, Structural and Multidisciplinary Optimization 65 (2) (2022) 1–22.

[32] C. Bonatti, D. Mohr, Large deformation response of additively-manufactured FCC metamaterials: From octet truss lattices towards continuous shell mesostructures, International Journal of Plasticity 92 (2017) 122–147.

[33] K. D. Nguyen, T. V. Duong, J. Lee, Y. Bazilevs, H. Nguyen-Xuan, An Efficient NURBS-Based Reconstruction Approach to Isogeometric Analysis of Triply Periodic Minimal Surface Structures, Engineering with Computers 41 (6) (2025) 4479–4509.

[34] N. Korshunova, G. Alaimo, S. B. Hosseini, M. Carraturo, A. Reali, J. Niiranen, F. Auricchio, E. Rank, S. Kollmannsberger, Image-Based Numerical Characterization and Experimental Validation of Tensile Behavior of Octet-Truss Lattice Structures, Additive Manufacturing 41 (2021) 101949.

[35] N. Korshunova, G. Alaimo, S. Hosseini, M. Carraturo, A. Reali, J. Niiranen, F. Auricchio, E. Rank, S. Kollmannsberger, Bending Behavior of Octet-Truss Lattice Structures: Modelling Options, Numerical Characterization and Experimental Validation, Materials & Design 205 (2021) 109693.

[36] A. Düster, J. Parvizian, Z. Yang, E. Rank, The Finite Cell Method for Three-Dimensional Problems of Solid Mechanics, Computer methods in applied mechanics and engineering 197 (45-48) (2008) 3768–3782.

[37] T. Hirschler, P. Antolin, A. Buffa, Fast and Multiscale Formation of Isogeometric Matrices of Microstructured Geometric Models, Computational Mechanics 69 (2) (2022) 439–466.

[38] T. Hirschler, R. Bouclier, P. Antolin, A. Buffa, Reduced Order Modeling Based Inexact FETI-DP Solver for Lattice Structures, International Journal for Numerical Methods in Engineering 125 (8) (2024) e7419.

[39] C. Guillet, T. Hirschler, P. Jolivet, P. Antolin, R. Bouclier, Efficient Fine-Scale Simulation of Nonlinear Hyperelastic Lattice Structures, arXiv preprint arXiv:2603.10741 (2026).

[40] C. Guillet, T. Hirschler, P. Jolivet, R. Bouclier, Multilevel Matrix-Free Method for High-Performance Isogeometric Analysis of Lattice Structures, Journal of Computational Physics 537 (2025) 114136.

[41] S. Kumar, S. Tan, L. Zheng, D. M. Kochmann, Inverse-Designed Spinodoid Metamaterials, npj Computational Materials 6 (1) (2020) 73.

[42] J. Feng, J. Fu, X. Yao, Y. He, Triply Periodic Minimal Surface (TPMS) Porous Structures: from Multi-Scale Design, Precise Additive Manufacturing to Multidisciplinary Applications, International Journal of Extreme Manufacturing 4 (2) (2022) 022001.

[43] S. Badia, F. Verdugo, A. F. Martín, The Aggregated Unfitted Finite Element Method for Elliptic Problems, Computer Methods in Applied Mechanics and Engineering 336 (2018) 533–553.

[44] A. Main, G. Scovazzi, The Shifted Boundary Method for Embedded Domain Computations. Part I: Poisson and Stokes Problems, Journal of Computational Physics 372 (2018) 972–995.

[45] E. Burman, P. Hansbo, M. G. Larson, S. Zahedi, Cut Finite Element Methods, Acta Numerica 34 (2025) 1–121.

[46] S. Chaturantabut, D. C. Sorensen, Nonlinear Model Reduction via Discrete Empirical Interpolation, SIAM Journal on Scientific Computing 32 (5) (2010) 2737–2764.

[47] F. Negri, A. Manzoni, D. Amsallem, Efficient Model Reduction of Parametrized Systems by Matrix Discrete Empirical Interpolation, Journal of Computational Physics 303 (2015) 431–454.

[48] E. N. Karatzas, F. Ballarin, G. Rozza, Projection-Based Reduced Order Models for a Cut Finite Element Method in Parametrized Domains, Computers & Mathematics with Applications 79 (3) (2020) 833–851.

[49] M. Chasapi, P. Antolin, A. Buffa, A Localized Reduced Basis Approach for Unfitted Domain Methods on Parameterized Geometries, Computer Methods in Applied Mechanics and Engineering 410 (2023) 115997.

[50] N. Mueller, S. Badia, Y. Zhao, Reduced Basis Solvers for Unfitted Methods on Parameterized Domains, Computer Methods in Applied Mechanics and Engineering 451 (2026) 118610.

[51] J. Parvizian, A. Düster, E. Rank, Finite Cell Method: h- and p-Extension for Embedded Domain Problems in Solid Mechanics, Computational Mechanics 41 (1) (2007) 121–133.

[52] C. Lehrenfeld, A. Reusken, Analysis of a High-Order Unfitted Finite Element Method for Elliptic Interface Problems, IMA Journal of Numerical Analysis 38 (3) (2018) 1351–1387.

[53] P. A. Martorell, S. Badia, High Order Unfitted Finite Element Discretizations for Explicit Boundary Representations, Journal of Computational Physics 511 (2024) 113127.

[54] A. Idesman, M. Mobin, J. Bishop, 10-th order of accuracy for numerical solution of 3-D elasticity equations for heterogeneous materials on unfitted Cartesian meshes, Computational Mechanics 76 (2025) 1085–1115.

[55] C. R. Dohrmann, A preconditioner for substructuring based on constrained energy minimization, SIAM Journal on Scientific Computing 25 (1) (2003) 246–258.

[56] J. Mandel, C. R. Dohrmann, Convergence of a balancing domain decomposition by constraints and energy minimization, Numerical Linear Algebra with Applications 10 (7) (2003) 639–659.

[57] J. Li, O. B. Widlund, FETI-DP, BDDC, and block Cholesky methods, International Journal for Numerical Methods in Engineering 66 (2) (2006) 250–271.

[58] L. Beirão da Veiga, D. Cho, L. F. Pavarino, S. Scacchi, BDDC preconditioners for isogeometric analysis, Mathematical Models and Methods in Applied Sciences 23 (6) (2013) 1099–1142.

[59] L. Beirão da Veiga, L. F. Pavarino, S. Scacchi, O. B. Widlund, S. Zampini, Isogeometric BDDC preconditioners with deluxe scaling, SIAM Journal on Scientific Computing 36 (3) (2014) A1118–A1139.

[60] S. Badia, F. Verdugo, Robust and Scalable Domain Decomposition Solvers for Unfitted Finite Element Methods, Journal of Computational and Applied Mathematics 344 (2018) 740–759.

[61] P. Antolin, QUGaR: Quadratures for Unfitted GeometRies, https://github.com/FELIGN/qugar, GitHub repository (2025).

[62] I. A. Baratta, J. P. Dean, J. S. Dokken, M. Habera, J. S. Hale, C. N. Richardson, M. E. Rognes, M. W. Scroggs, N. Sime, G. N. Wells, DOLFINx: The next generation FEniCS problem solving environment (Dec. 2023).

[63] R. I. Saye, Algoim: Algorithms for Implicit Domains, https://github.com/algoim/algoim, GitHub repository (2025).

[64] R. I. Saye, High-Order Quadrature Methods for Implicitly Defined Surfaces and Volumes in Hyperrectangles, SIAM Journal on Scientific Computing 37 (2) (2015) A993–A1019.

[65] R. I. Saye, High-Order Quadrature on Multi-Component Domains Implicitly Defined by Multivariate Polynomials, Journal of Computational Physics 448 (2022) 110720.

[66] M. Dauge, A. Düster, E. Rank, Theoretical and Numerical Investigation of the Finite Cell Method, Journal of Scientific Computing 65 (3) (2015) 1039–1064.

[67] T. P. A. Mathew, Domain Decomposition Methods for the Numerical Solution of Partial Differential Equations, Springer, 2008.

[68] A. Heinlein, A. Klawonn, M. Lanser, J. Weber, A Frugal FETI-DP and BDDC Coarse Space for Heterogeneous Problems, Tech. rep., TR series, Center for Data and Simulation Science, University of Cologne (2019).

[69] J. Mandel, C. R. Dohrmann, R. Tezaur, An algebraic theory for primal and dual substructuring methods by constraints, Applied Numerical Mathematics 54 (2) (2005) 167–193.

[70] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, J. Koster, A fully asynchronous multifrontal solver using distributed dynamic scheduling, SIAM Journal on Matrix Analysis and Applications 23 (1) (2001) 15–41.

[71] T. Hirschler, P. Antolin, A. Buffa, Fast and Multiscale Formation of Isogeometric Matrices of Microstructured Geometric Models, Computational Mechanics 69 (2) (2022) 439–466.

[72] A. Mantzaflaris, B. Jüttler, Integration by interpolation and look-up for Galerkin-based isogeometric analysis, Computer Methods in Applied Mechanics and Engineering 284 (2015) 373–400.

[73] M. D. Mckay, R. J. Beckman, W. J. Conover, A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code, Technometrics 42 (1) (2000) 55–61.

[74] M. Barrault, Y. Maday, N. C. Nguyen, A. T. Patera, An 'Empirical Interpolation' Method: Application to Efficient Reduced-Basis Discretization of Partial Differential Equations, Comptes Rendus Mathematique 339 (9) (2004) 667–672.

[75] M. J. Powell, The Theory of Radial Basis Function Approximation in 1990, Advances in numerical analysis (1992) 105–210.

[76] H.-J. Bungartz, M. Griebel, Sparse Grids, Acta Numerica, Cambridge University Press, 2004, pp. 147–270.

[77] P. Antolin, G. Bonilla Moreno, Reduced-Order Model data for FLAS$_h$: Fast simulation tools for Lattice Structures, https://doi.org/10.5281/zenodo.19254389, zenodo dataset (Mar. 2026). doi:10.5281/zenodo.19254389.

[78] G. Bonilla Moreno, FLAS$_h$: Fast simulation tools for Lattice Structures, https://github.com/Gonzalobm99/FLASh, GitHub repository (2025).

[79] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, E. M. Constantinescu, L. Dalcin, S. Benson, A. Dener, V. Eijkhout, J. Faibussowitsch, W. D. Gropp, V. Hapla, T. Isaac, P. Jolivet, D. Karpeev, D. Kaushik, M. G. Knepley, F. Kong, S. Kruger, D. A. May, L. C. McInnes, R. T. Mills, L. Mitchell, T. Munson, J. E. Roman, K. Rupp, P. Sanan, J. Sarich, B. F. Smith, H. Suh, S. Zampini, H. Zhang, J. Zhang, PETSc/TAO users manual revision 3.24, Tech. Rep. ANL-21/39 - Revision 3.24, Argonne National Laboratory (2025).

[80] L. D. Dalcin, R. R. Paz, P. A. Kler, A. Cosimo, Parallel distributed computing using Python, Advances in Water Resources 34 (9) (2011) 1124–1139.

[81] E. Burman, S. Claus, P. Hansbo, M. G. Larson, A. Massing, CutFEM: Discretizing Geometry and Partial Differential Equations, International Journal for Numerical Methods in Engineering 104 (7) (2015) 472–501.

[82] D. M. Young, Iterative Solution of Large Linear Systems, Elsevier, 2014.

[83] J. W. Ruge, K. Stüben, Algebraic Multigrid, in: S. F. McCormick (Ed.), Multigrid Methods, SIAM, 1987, pp. 73–130.