École polytechnique fédérale de Lausanne

Semester project spring 2023

Master in Computational Science and Engineering

# Fast parallel simulation of architected materials using domain decomposition techniques

*Author:*
Pau Romeu Llordella

*Supervisor:*
Pablo Antolin Sanchez

# Contents

# 1 Introduction

The development of architected metamaterials has marked a significant milestone in the field of materials science and engineering [3]. These uniquely designed materials, characterized by their intricate and regular microstructures, exhibit extraordinary mechanical properties that are not typically found in conventional materials. The lattice-like structures of these metamaterials shown in Figure 1 have opened up a huge ammount of applications, ranging from the creation of lightweight structures and energy absorption systems to the development of advanced acoustic, thermal, and vibration insulation systems. Their ultralight yet ultrastiff characteristics make them ideal for applications where weight is a critical factor, such as in the aerospace and automotive industries. This unique combination of properties, coming from their carefully designed microarchitecture, makes architected metamaterials a fascinating subject of study. However, the complex geometry and large-scale nature of these materials present substantial computational challenges when it comes to predicting their behavior under various loading conditions.

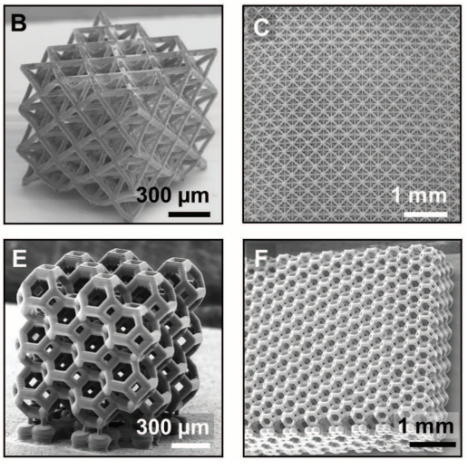



Figure 1: Layout of different architected metamaterials. Obtained from [3].

To address this challenge, this semester project aims to develop and implement the Finite Element Tearing and Interconnecting - Dual Primal (FETI-DP) method, a domain decomposition method known for its efficiency in solving large-scale problems in computational mechanics. The FETI-DP method is particularly suited for parallel computing environments, as it decomposes a large problem into smaller, more manageable subproblems that can be solved independently [2]. This approach is ideal for the regular, lattice-like structures of architected metamaterials, where each unit cell in the lattice can be treated as a subdomain.

The primary goal of this project is to create a custom implementation of the FETI-DP method that exploits the unique characteristics of architected metamaterials. The project involves the development of the FETI-DP algorithm, its implementation in a parallel computing environment, and its application to various structural problems involving architected metamaterials based on the work presented in [2].

# 2 Problem formulation

## 2.1 FETI methods

The main objective of this paper is to study the domain decomposition (DD) methods, specifically focusing on the Finite Element Tearing and Interconnecting (FETI) method and its advanced version, the FETI-Dual Primal (FETI-DP) method. The FETI methods are a family of algorithms for the fast parallel iterative solution of large systems of equations arising from the Finite element discretization of partial differential equations [2].

Domain decomposition problems involve breaking down a large problem into smaller subproblems, each defined over a subset of the original domain. The coarse problem, with one or more unknowns pertaining to each subdomain, is used to coordinate the solution globally among the subdomains. Because the coarse problem is much smaller than the original, it leads to a faster computation of the system.

FETI methods are iterative techniques that aim to satisfy the equilibrium in each substructure at each iteration. FETI methods can be understood as a Conjugate Gradient algorithm where in each iteration, two main steps are performed. First, the problem is solved individually in each of the subdomains by applying Dirichlet boundary conditions. Typically, this step is performed as part of the preconditioner of the most efficient version of the CG the Preconditioned Conjugate Gradient. Next, the global problem is solved where Neumann boundary conditions are applied to compute the residuals.

The original FETI method employs Lagrange multipliers to enforce the continuity of the displacement field at the subdomain interfaces. This continuity, however, is fully achieved only at convergence. The FETI-2 method, an extension of the original FETI, introduces corrective Lagrange multipliers to ensure the exact continuity of the displacement field at the subdomain corners at each (preconditioned) conjugate gradient iteration.

The FETI-DP method, on the other hand, presents a dual-primal formulation. This formulation considers the displacement degrees of freedom at the subdomain corners as part of the fundamental unknowns, thereby eliminating the need for corrective Lagrange multipliers used in the FETI-2 method. This ensures the continuity of some or all components of the displacement field at the corner nodes at each CG iteration.

A huge advantage of the FETI-DP method is that the problem matrices associated with floating subdomains are never singular. This implies that the FETI-DP solver does not have to compute any subdomain rigid body mode, enhancing the robustness of the method and reducing code maintenance.

In terms of computational efficiency, the FETI-DP method outperforms the original FETI and FETI-2 methods. While most FETI methods require solving at least two coarse problems at each CG iteration, the FETI-DP method has to solve only one coarse problem at each CG iteration. This reduction contributes to the improved efficiency of the FETI-DP method.

## 2.2 Problem decomposition

This section aims to describe how a global general problem that is to be solved can be decomposed in subdomains. In the FETI-DP framework, we divide the original problem defined by

$$K\mathbf{u} = \mathbf{f} \tag{1}$$

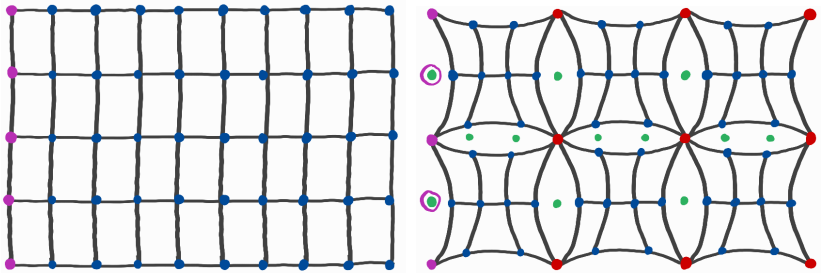
into smaller sub-problems associated with subdomains. This is the classical formulation for a structural problem where $K$ is the stiffness matrix, $\mathbf{f}$ is the source vector and $\mathbf{u}$ represent the unknown. An example of this decomposition is illustrated in Figure 2a, where the original geometry is decomposed into subdomains as shown in Figure 2b.

Three distinct types of nodes are defined within this decomposition:

1. Primal nodes $P$ (red): These nodes connect different subdomains and are shared by them, they are located in the corners of the subdomain. This nodes will be part of the coarse problem to be solved in the dual-primal formulation of the FETI-DP method. We have $N_P$ primal nodes. This number is always way lower than the total number of original nodes $N$ and therefore the resulting problem is significantly smaller.

2. Remainder nodes $R$ (blue): These nodes are located within each subdomain and belong exclusively to that subdomain. In this paper, all subdomains are assumed to be identical. The total number of remainder nodes, $N_R$, is the product of the number of remainder nodes in each subdomain, $N_r$, and the number of subdomains, $N_S$.

3. Dirichlet nodes $D$ (purple): These nodes have imposed solutions as boundary conditions. They are also located in the corners of the subdomain but they are not treated as part of the primal nodes $P$.



(a) Original geometry (b) Subdomains-decomposed geometry

Figure 2: Mesh of the problem. In purple, Dirichelt nodes $D$. In blue, remainder nodes $R$. In red, primal nodes $P$. In green, lagrange multipliers nodes $\lambda$

This way, we can express the unknown $\mathbf{u}$ as

$$\mathbf{u} = \begin{pmatrix} \mathbf{u}_P \\ \mathbf{u}_D \\ \mathbf{u}_R \end{pmatrix}. \tag{2}$$

where each component is the array of unknowns referring to the primals, Dirichlet and remainder nodes respectively. Note that, by definition, $\mathbf{u_D}$ is already known.

To account for the overlap between subdomains, Lagrange multipliers $\lambda$ are introduced at interfaces. The role of these multipliers is to ensure the continuity of the solution across subdomains. This includes remainder nodes representing the same physical point in different subdomains (green), as well as nodes associated with Dirichlet boundary conditions (green with purple circle). Therefore, the total number of Lagrange multipliers $N_\lambda$ is $N_\lambda = N_{\lambda u} + N_{\lambda D}$, accounting for remainder unknown nodes and Dirichlet conditions respectively.

## 2.3 Lagrange multipliers in interfaces

In the FETI-DP method, ensuring the continuity of the solution across subdomains is crucial. This requirement is particularly significant at the interfaces between subdomains, where individual nodes from the original problem are shared. When the problem is split into subdomains, the same physical point may be represented as a node in multiple subdomains. Remind that this is the case only in the $R$ set of nodes. To guarantee the coherence of the solution, these corresponding nodes should have the same displacement in the final result $\mathbf{u}_R$.

This alignment is achieved through the introduction of Lagrange multipliers, which impose equality conditions on corresponding nodes across subdomains. These conditions are represented by the matrix $B_R$ and the vector $\mathbf{d}$, which form a system of equations to enforce the displacement continuity:

$$B_R \mathbf{u}_R = \mathbf{d} \tag{3}$$

Remind that we denote the number of remainder nodes as $N_R$. This value consists of the remainder nodes that are unknown $N_{Ru}$ plus the number of remainder nodes that are part of the boundary conditions, $N_{RD}$. Hence, we have $N_R = N_{Ru} + N_{RD}$. For the number of Lagrange multipliers, $N_\lambda$, we can also distinguish between those associated with the remainder unknowns and those linked to the Dirichlet remainder nodes, resulting in $N_\lambda = N_{\lambda u} + N_{\lambda D}$. This way, the vector $\mathbf{u}_R$ belongs to $\mathbb{R}^{N_R}$ and the vector $\mathbf{d}$ belongs to $\mathbb{R}^{N_\lambda}$ , whereas the matrix $B_R$ we are to construct belongs to $\mathbb{R}^{N_\lambda \times N_R}$.

Now, let's turn to assembling the matrix $B_R$ and the vector $\mathbf{d}$. We will use the first $N_{\lambda u}$ rows, corresponding to the unknown remainder nodes, to ensure the same nodes have final equal value. For instance, if $\boldsymbol{\lambda}[k]$ is the multiplier enforcing that remainder node $R_i$ equals remainder node $R_j$, we'll populate the matrix such that $B_R[k, i] = 1$ and $B_R[k, j] = -1$, and the vector with $\mathbf{d}[k] = 0$. Thus, the $k$-th row of the matrix and vector gives us the equation:

$$\mathbf{u}_R[i] - \mathbf{u}_R[j] = 0$$

The next block of $N_{\lambda D}$ rows corresponds to the remainder nodes where the value is already determined because they belong to the boundary condition. For the remainder node $R_h$, we enforce its value $\alpha_h$ using the multiplier $\boldsymbol{\lambda}[l]$ by setting $B_R[l, h] = 1$ and $\mathbf{d}[l] = \alpha_h$, so that the $h$-th row of the matrix gives us:

$$\mathbf{u}_R[h] = \alpha_h$$

## 2.4 Local-global transformation matrix

In the following demonstrations, we will work in both the global domain and the local domain (i.e, subdomain). To facilitate the transition from the local numeration of subdomains to the global context of the problem, we construct transformation matrices. These matrices are represented as $A$. Each subdomain has its unique transformation matrix, denoted as $A^{(s)}$. We establish matrices for the remainder nodes $R$ and primal nodes $P$, which we denote as $A^{(s)}_{Rr}$ and $A^{(s)}_{Pp}$ respectively.

The transformation matrix for the remainder nodes, denoted as $A^{(s)}_{Rr}$, is structured with $N_R$ rows and $N^s_r$ columns, where $N^s_r$ stands for the number of remainder nodes in the subdomain $s$. This matrix is primarily sparse, with all entries initialized to zero, adding a single '1' in each column.

The purpose of this matrix is to map the global remainder nodes to their corresponding local counterparts in each subdomain. For example, consider a global remainder node $R_i$ which is enumerated as $i$ in the global context. This node corresponds to a local remainder node $r_i$ in a subdomain, which is enumerated as $i_{loc}$ in the local context. In such a case, we populate the matrix as follows:

$$A^{(s)}_{Rr}[i, i_{loc}] = 1$$

The transformation matrix for the primal nodes is constructed the same way. In this case, $A^{(s)}_{Pp} \in \mathbb{R}^{N_P \times N^s_p}$ where $N^s_p$ stands for the number of local primal nodes in the specific subdomain $s$.

## 2.5 Algebra decomposition of the FETI-DP problem

By using the variables introduced in the previous sections, we managed to obtain a FETI-DP formulation of the original problem described in Equation 1. It is first rewritten in a generalized form using Equation 3 that encompasses all nodes.

$$\begin{pmatrix} 0 & B \\ B^T & K \end{pmatrix} \begin{pmatrix} \boldsymbol{\lambda} \\ \mathbf{u} \end{pmatrix} = \begin{pmatrix} \mathbf{d} \\ \mathbf{f} \end{pmatrix} \tag{4}$$

As discussed in Section 2.2, we dissect the nodes of the original problem into primal, Dirichlet, and remainder nodes, based on their specific roles, as shown below:

$$\mathbf{u} = \begin{pmatrix} \mathbf{u}_P \\ \mathbf{u}_D \\ \mathbf{u}_R \end{pmatrix}, \mathbf{f} = \begin{pmatrix} \mathbf{f}_P \\ \mathbf{f}_D \\ \mathbf{f}_R \end{pmatrix}, K = \begin{pmatrix} K_{PP} & K_{PD} & K_{PR} \\ K_{DP} & K_{DD} & K_{DR} \\ K_{RP} & K_{RD} & K_{RR} \end{pmatrix} \tag{5}$$

With this decomposition, we can revise Equation 4, employing the expressions from 5:

$$\begin{pmatrix} 0 & 0 & 0 & B_R \\ 0 & K_{PP} & K_{PD} & K_{PR} \\ 0 & K_{DP} & K_{DD} & K_{DR} \\ {B_R}^T & K_{RP} & K_{RD} & K_{RR} \end{pmatrix} \begin{pmatrix} \boldsymbol{\lambda} \\ \mathbf{u}_P \\ \mathbf{u}_D \\ \mathbf{u}_R \end{pmatrix} = \begin{pmatrix} \mathbf{d} \\ \mathbf{f}_P \\ \mathbf{f}_D \\ \mathbf{f}_R \end{pmatrix} \tag{6}$$

In the third row of Equation 6, the generated equation pertains to the Dirichlet nodes $\mathbf{u}_D$ which already have a known solution. Consequently, this row is removed. In the next step, we also eliminate the third column of the matrix, defining vectors $\tilde{\mathbf{f}_P}$ and $\tilde{\mathbf{f}_R}$, which carry the contribution of $\mathbf{u}_D$:

$$\begin{pmatrix} 0 & 0 & B_R \\ 0 & K_{PP} & K_{PR} \\ {B_R}^T & K_{RP} & K_{RR} \end{pmatrix} \begin{pmatrix} \boldsymbol{\lambda} \\ \mathbf{u}_P \\ \mathbf{u}_R \end{pmatrix} = \begin{pmatrix} \mathbf{d} \\ \tilde{\mathbf{f}_P} \\ \tilde{\mathbf{f}_R} \end{pmatrix} \tag{7}$$

$$\tilde{\mathbf{f}_P} = \mathbf{f}_P - K_{PD}\mathbf{u}_D, \qquad \tilde{\mathbf{f}_R} = \mathbf{f}_R - K_{RD}\mathbf{u}_D \tag{8}$$

Following this step, we end up with three equations:

$$B_R\mathbf{u}_R = \mathbf{d} \tag{9}$$

$$K_{PP}\mathbf{u}_P + K_{PR}\mathbf{u}_R = \tilde{\mathbf{f}_P} \tag{10}$$

$${B_R}^T\boldsymbol{\lambda} + K_{RP}\mathbf{u}_P + K_{RR}\mathbf{u}_R = \tilde{\mathbf{f}_R} \tag{11}$$

The main objective of the following steps is to algebraically modify these equations to express the unknowns $\mathbf{u}_P$ and $\mathbf{u}_R$ as functions of $\boldsymbol{\lambda}$. This way, the system we need to solve relies only on this vector. We start by rephrasing Equation 10.

$$K_{PP}\mathbf{u}_P = \tilde{\mathbf{f}}_P - K_{PR}\mathbf{u}_R \tag{12}$$

Next, Equation 11 is rewritten:

$$\mathbf{u}_R = {K_{RR}}^{-1}\left(\tilde{\mathbf{f}}_R - {B_R}^T\boldsymbol{\lambda} - K_{RP}\mathbf{u}_P\right) \tag{13}$$

We insert Equation 13 into Equation 12:

$$K_{PP}\mathbf{u}_P = \tilde{\mathbf{f}}_P - K_{PR}{K_{RR}}^{-1}\tilde{\mathbf{f}}_R + K_{PR}{K_{RR}}^{-1}{B_R}^T\boldsymbol{\lambda} + K_{PR}{K_{RR}}^{-1}K_{RP}\mathbf{u}_P \tag{14}$$

From this last expression, we define the term $S_{PP}$ as follows:

$$S_{PP} = K_{PP} - K_{PR}{K_{RR}}^{-1}K_{RP} \tag{15}$$

This matrix is defined as the primal Schur complement in terms of FETI-DP and is the one that describes the coarse problem. This is further discussed in Section 3.2

This allows us to obtain the expression for $\mathbf{u}_P$:

$$\mathbf{u}_P = {S_{PP}}^{-1}\left(\tilde{\mathbf{f}}_P - K_{PR}{K_{RR}}^{-1}\tilde{\mathbf{f}}_R + K_{PR}{K_{RR}}^{-1}{B_R}^T\boldsymbol{\lambda}\right) \tag{16}$$

We now use this result in Equation 13 and we obtain the expression for $\mathbf{u}_R$:

$$\mathbf{u}_R = {K_{RR}}^{-1}\left(I_R + K_{RP}{S_{PP}}^{-1}K_{PR}{K_{RR}}^{-1}\right)\tilde{\mathbf{f}}_R - {K_{RR}}^{-1}K_{RP}{S_{PP}}^{-1}\tilde{\mathbf{f}}_P - {K_{RR}}^{-1}\left(I_R + K_{RP}{S_{PP}}^{-1}K_{PR}{K_{RR}}^{-1}\right){B_R}^T\boldsymbol{\lambda} \tag{17}$$

Our goal is to find an expression that allows us to solve a system with $\boldsymbol{\lambda}$ as the only unknown. To this end, we use Equation 17 and introduce it into Equation 9.

$$F\boldsymbol{\lambda} = \tilde{\mathbf{d}} \tag{18}$$

$$F = -B_R{K_{RR}}^{-1}\left(I_R + K_{RP}{S_{PP}}^{-1}K_{PR}{K_{RR}}^{-1}\right){B_R}^T \tag{19}$$

$$\tilde{\mathbf{d}} = \mathbf{d} - B_R{K_{RR}}^{-1}\left(\left(I_R + K_{RP}{S_{PP}}^{-1}K_{PR}{K_{RR}}^{-1}\right)\tilde{\mathbf{f}}_R - K_{RP}{S_{PP}}^{-1}\tilde{\mathbf{f}}_P\right) \tag{20}$$

With this, we state the problem to solve. The next section details how to use this expression to solve the problem more efficiently than employing a standard solver for a system of linear equations.

# 3 Implementation

The aim of this section is to provide a framework that translates the outcomes of the preceding section into a programmable format. Initially, we introduce the conjugate gradient method, which is employed to solve the system that determines the unknown Lagrange multipliers $\boldsymbol{\lambda}$. The primary goal is to utilize the FETI-DP formulation to execute the matrix-vector multiplication, a process that can be computationally expensive if performed in a naive approach.

## 3.1 Conjugate Gradient

The Algorithm presented here (1) is the implementation of the Conjugate Gradient Method, an iterative method for solving systems of linear equations that are symmetric and positive-definite. The parameters required for this method are the matrix $F$, the target vector $\tilde{\mathbf{d}}$, an initial guess of the solution $\boldsymbol{\lambda}_0$, a tolerance for convergence *tol*, and a maximum number of iterations *max_iterations*. The goal is to find a vector $\boldsymbol{\lambda}$ such that $F\boldsymbol{\lambda} \approx \tilde{\mathbf{d}}$ to solve the problem stated in the previous section.

**Algorithm 1** Conjugate Gradient Method

**Require:** $F$, $\tilde{\mathbf{d}}$, $\boldsymbol{\lambda}_0$, tol, max_iterations
**Ensure:** $\boldsymbol{\lambda}$ such that $F\boldsymbol{\lambda} \approx \tilde{\mathbf{d}}$

1: $\mathbf{r}_0 \leftarrow \tilde{\mathbf{d}} - F\boldsymbol{\lambda}_0$ ▷ Matrix-vector multiplication
2: $\mathbf{p}_0 \leftarrow \mathbf{r}_0$
3: **for** $k = 0, 1, 2, \ldots,$ max_iterations $- 1$ **do**
4: $\quad \mathbf{fp}_k \leftarrow F\mathbf{p}_k$ ▷ Matrix-vector multiplication
5: $\quad \alpha_k \leftarrow \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{p}_k^T \mathbf{fp}_k}$
6: $\quad \boldsymbol{\lambda}_{k+1} \leftarrow \boldsymbol{\lambda}_k + \alpha_k \mathbf{p}_k$
7: $\quad \mathbf{r}_{k+1} \leftarrow \mathbf{r}_k - \alpha_k \mathbf{fp}_k$
8: $\quad$ **if** $\|\mathbf{r}_{k+1}\| <$ tol **then**
9: $\quad\quad$ **exit**
10: $\quad$ **end if**
11: $\quad \beta_k \leftarrow \frac{\mathbf{r}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{r}_k^T \mathbf{r}_k}$
12: $\quad \mathbf{p}_{k+1} \leftarrow \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$
13: **end for**

Looking at the pseudocode of the algorithm, the most computationally expensive operation is the matrix-vector multiplication, particularly in lines 1 and 4. In most cases, constructing and storing large matrices can be computationally intensive, and as it can be seen in the algorithm, we only need to multiply the matrix $F$ by a vector. Thus, to reduce the computational cost, instead of constructing the matrix $F$, the implementation computes the product directly. This kind of optimization where we only compute what is necessary is common in large-scale numerical computations. It can significantly reduce the time and memory requirements of the algorithm, leading to a much more efficient solution.

## 3.2 Matrix-vector multiplication using FETI-DP

We can use Expression 19 where $F$ is defined to write the multiplication of $F$ with a vector $\mathbf{z}$ as follows:

$$F\mathbf{z} = -B_R {K_{RR}}^{-1} {B_R}^T \mathbf{z} - B_R {K_{RR}}^{-1} K_{RP} {S_{PP}}^{-1} K_{PR} {K_{RR}}^{-1} {B_R}^T \mathbf{z} \tag{21}$$

The following is a strategy to compute the matrix-vector multiplication more efficiently by leveraging the structure of the matrix $F$, its subdomain property, and the power of parallel computing.

The matrix $F$ is structured as in Equation 21. Its computation involves a chain of matrix-vector multiplications and inversions. In order to avoid constructing and inverting large matrices, we can perform these operations step by step, from left to right, storing intermediate results. This is essentially what the strategy is about.

The first summation term of Equation 21 can be conveniently calculated utilizing the subsequent expression:

$$\mathbf{v} = B_R {K_{RR}}^{-1} {B_R}^T \mathbf{z} = \sum_{s=1}^{N_S} B_r^{(s)} {K_{rr}}^{-1} {B_r^{(s)}}^T \mathbf{z} \tag{22}$$

It is important to note that this step can be executed in parallel for each subdomain independently, as all terms are solely dependent on the specific subdomain $s$. In the FETI-DP framework, this initial step is typically interpreted as solving the Dirichlet problem within each subdomain. This step is commonly performed as a component of the preconditioner. However, for the purpose of simplicity in this project, it will be executed as an additional step within the standard CG.

We continue by computing $\mathbf{y}$ (as shown in Equation 23), which involves the multiplication of several matrices and the vector $\mathbf{z}$ for the second term of the sum. The computation of $\mathbf{y}$ can again be performed independently for each subdomain and therefore parallelized. This operation results in the sum of the contributions from all subdomains $N_s$. We make use of the the transformation matrix $A_{Pp}^{(s)}$ defined in Section 2.4.

$$\mathbf{y} = K_{PR}{K_{RR}}^{-1}{B_R}^T\mathbf{z} = \sum_{s=1}^{N_S} A_{Pp}^{(s)} K_{pr}^{(s)} {K_{rr}}^{-1} {B_r^{(s)}}^T \mathbf{z} \tag{23}$$

Next, we need to compute $\mathbf{x}$, which involves the multiplication of the inverse of primal Schur complement ${S_{PP}}^{-1}$ and $\mathbf{y}$. We assume that $S_{PP}$ has already been assembled using Expression 15. Note that this is relatively small matrix with dimensions equal to the number of primal nodes ($\mathbb{R}^{N_P \times N_P}$). This is the only one coarse problem that FETI-DP needs to solve. Since this equation involves the primal nodes pertaining to all subdomains, it is performed globally and cannot be parallelized. However, we still can optimize this operation by performing a Cholesky decomposition of $S_{PP}$ and solving the resulting triangular system instead of directly computing the matrix inversion.

$$S_{PP}\mathbf{x} = \mathbf{y} \tag{24}$$

Once, $\mathbf{x}$ has been computed we proceed to find the rest of the expression. This part, again can be computed in parallel.

$$\mathbf{w} = B_R{K_{RR}}^{-1}K_{RP}\mathbf{x} = \sum_{s=1}^{N_S} B_r^{(s)} {K_{rr}}^{-1} K_{rp}^{(s)} A_{Pp}^{(s)} \mathbf{x} \tag{25}$$

Finally, with the computations of $\mathbf{v}$ and $\mathbf{w}$ done, we can compute the multiplication of $F$ and $\mathbf{z}$ as

$$F\mathbf{z} = -(\mathbf{v} + \mathbf{w}) \tag{26}$$

Algorithm 2 describes this procedure and presents a function that can be used in lines 1 and 4 of Algorithm 1 to perform the matrix-vector multiplication of the CG.

**Algorithm 2** Matrix-Vector Multiplication Using Geometry of Subdomains

1: **function** SUBDOMAINS_MULTIPLY($\mathbf{z}$, $A_{Pp}$, $K_{pr}$, $K_{rr}$, $B_r$, $K_{rp}$, $S_{PP}$)
2: **for** $s = 1, \ldots, N_S$ **do**
3: Compute $\mathbf{v}^s \leftarrow B_r^s (K_{rr})^{-1} (B_r^s)^T \mathbf{z}$ ▷ Parallel operation in each subdomain
4: Compute $\mathbf{y}^s \leftarrow A_{Pp}^s K_{pr}^s (K_{rr})^{-1} (B_r^s)^T \mathbf{z}$ ▷ Parallel operation in each subdomain
5: **end for**
6: Sync ▷ Synchronize the parallel operations
7: $\mathbf{v} \leftarrow \sum_{s=1}^{N_S} \mathbf{v}^s$
8: $\mathbf{y} \leftarrow \sum_{s=1}^{N_S} \mathbf{y}^s$
9: Solve $S_{PP}\mathbf{x} = \mathbf{y}$ ▷ Global operation (Coarse problem)
10: **for** $s = 1, \ldots, N_S$ **do**
11: Compute $\mathbf{w}^s \leftarrow B_r^s (K_{rr})^{-1} K_{rp}^s A_{Pp}^s \mathbf{x}$ ▷ Parallel operation in each subdomain
12: **end for**
13: Sync ▷ Synchronize the parallel operations
14: $\mathbf{w} \leftarrow \sum_{s=1}^{N_S} \mathbf{w}^s$
15: $\mathbf{fp} \leftarrow -(\mathbf{v} + \mathbf{w})$
16: **return fp**
17: **end function**

# 4 Results

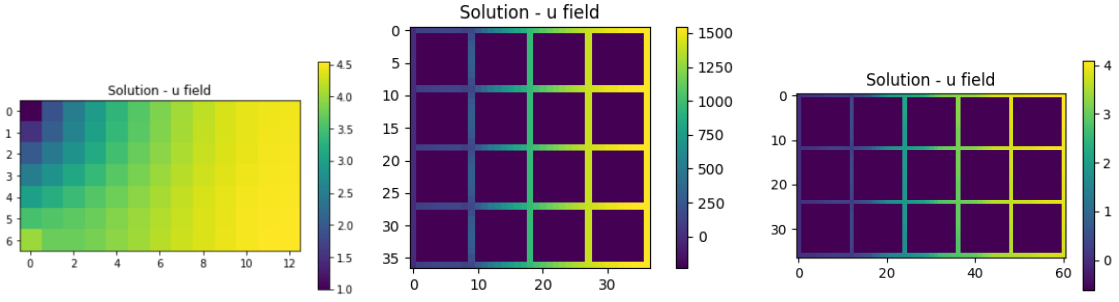
The aim of this section is to discuss the main results obtained with the parallel implementation of the FETI-DP method. We are interested in studying the effect of parallelization on the problem and how performance scales with the number of nodes used. We also aim to investigate the impact of varying the number of subdomains and the size these have. The key metrics to consider in this type of study are the problem-solving time and the number of iterations required for convergence. These values allow us to understand the efficiency of the method's resolution and how it scales for larger problems. To compare between different scenarios, we fix the value of the tolerance *tol* to determine the residual value considered acceptable ($tol = 1 \times 10^{-8}$).

The scheme followed for the code design is presented in Section 3.2. First, all necessary matrices to define the problem are constructed. We should take advantage of the fact that the vast majority of the defined matrices are sparse and use this fact to reduce the required memory. Subsequently, we have constructed the solution method by implementing the Conjugate Gradient using the decomposition of the matrix-vector product described. Unlike what is proposed in the paper [2], a preconditioner has not been used. The use of this would reduce the conditioning number of the matrix to be solved and therefore the number of iterations. However, for simplicity, it has been developed without it. Also, for the sake of simplicity, the problem is developed in 2D and with a single degree of freedom in each unknown. This way, it can be understood as solving the displacements in only one direction or as solving a heat conduction problem where the unknowns are values of temperature.

The code is designed in such a way that given the stiffness matrix $K_s$ and the load vector $\mathbf{f}_s$ of a subdomain, as many subdomains can be added in the horizontal and vertical direction as the user wishes. These subdomains must have equivalent nodes on the horizontal faces to be able to connect the left-right nodes and equivalent nodes on the vertical interfaces to connect the top-bottom subdomains. The interior geometry can be completely irregular. The design of the global problem has been carried out following the scheme in Figure 2, in such a way that only the left nodes of the left subdomains will have Dirichlet nodes.

The presented code provides the validated resolution of three problems:

- A very small structural problem, the solution of which is known, to validate the code (Figure 3a).
- A larger heat conduction problem with a perfectly regular internal structure. Their number of domains and the size of these can be decided by the user (Figure 3b).
- A structural problem of architected metamaterials, in which the borders of each subdomain are regular but the internal geometry is completely irregular (Figure 3c). Note that only the geometry of the subdomain borders is presented, as this is the only numbering available. How the internal nodes are distributed is unknown.




(a) Small structural problem. (b) Heat conduction problem. (c) Architected material problem.

Figure 3: Study problems.

The results shown below have been obtained using the heat conduction problem, due to the ease of scaling it both in subdomain size and in the number of these, though they are perfectly applicable to the architected materials problem.

The code has been created using Python and its parallelization module mpi4py, which is an implementation of the MPI library. The repository of the code is available clicking here. The results of this semester project have been obtained using a personal laptop. The computer model is a Lenovo Legion 5 AMD Ryzen 7 5800 with 8 nodes. The use of a more powerful computing infrastructure would have allowed us to obtain more representative results, but this was beyond the scope of this master semester project.

## 4.1 Strong scaling

The subdivision into subdomains allows us to derive a problem that can be solved quite easily in parallel. As demonstrated in the previous section, many of the operations performed to solve the linear system of equations - which lead to the final solution - can be carried out within each subdomain, and thus, independently.

In this section, we present the results of this parallelization. We solved a problem for which the analytical solution is known (so it can be validated), with $N_s = 100$ subdomains and a total of $N = 1200$ degrees of freedom. The tolerance used is $tol = 1 \times 10^{-8}$. Each node is assigned a similar number of subdomains that will be solved there independently.

Indeed, the computation time is reduced for each additional node used, achieving a substantial reduction in the computation time. In all cases, the problem converges in only 4 iterations. We present in Figure 4 the strong scaling curve of the parallelization. This figure informs us about the level of parallelization that the code can achieve. It represents the speedup in relation to the ideal speedup (with double the nodes, the time should ideally be halved). According to Amdahl's law [1], the performance improvement obtained through code parallelization is constrained by the fraction of the code that can effectively be parallelized. As observed in the previous section, part of the solution time is dedicated to solving the coarse problem. This step is not parallelizable. Thus, no matter how many nodes are added, the computation time can never be less than the time it takes to solve this synchronous problem.

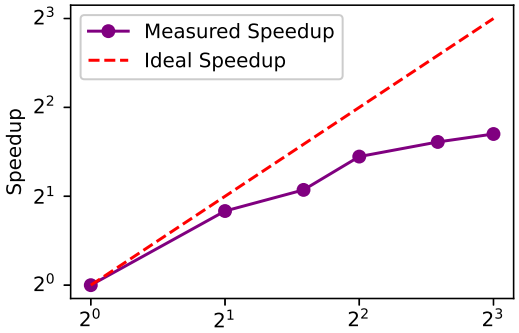
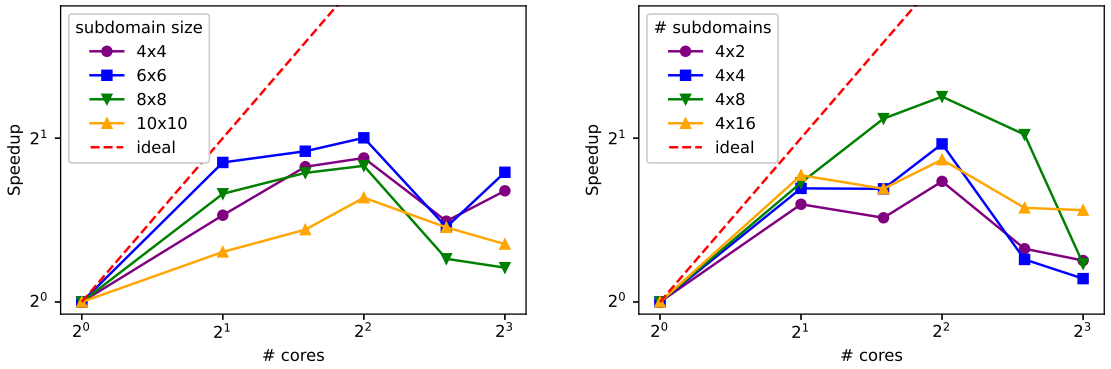



Figure 4: Speedup of time computation depending on number of cores

We can observe how the speedup increases as the number of nodes is augmented. However, it does not scale at the same rate as the ideal speedup (which would be achieved if the code was 100% parallelizable). Amdahl's curve tends to approach an asymptote when the time can no longer be reduced, no matter how many nodes are added. This is precisely what we can observe in the presented figure. It should be noted that this problem has been run on a personal laptop, and therefore, the size studied cannot be overly large. It is well known that Amdahl's curve improves as the amount of parallel computation increases. In this case, the curve will tend closer to the ideal as the size of the subdomains increases. In other words, the bigger is $N_r$ with respect to $N_P$, the more parallelizable the code will be. Figure 5 aims to study this phenomena.




(a) $N_s$ is kept constant. $N_r$ increases.

(b) $N_r$ is kept constant. $N_s$ increases.

Figure 5: Influence of number and size of subdomains in parallelization.

Figure 5a has been created with a regular problem of $4 \times 4$ subdomains. The size of these subdomains was increased as shown in the figure from $4 \times 4$ to $10 \times 10$ nodes. It can be observed that the best scalability occurs for an intermediate size ($6 \times 6$). This effect is due to the contrasting impacts observed at the extremes. When the problem is very small ($4 \times 4$), the effect of parallelization is not as efficient because the time spent on data communication compared to the calculation time within each subdomain increases. This means that scalability should be superior for larger problems ($10 \times 10$), however, it can be seen that this is not the case. The infrastructure used to test the problem comes into play here. In this case, the matrices used to solve the problem grow proportionally. Therefore, although the CPU can calculate faster, we face the laptop's cache memory limit. As such, the problem is bounded by the access time to the main memory.

Figure 5b intends to study what happens when the number of subdomains is increased while keeping their size constant (in this case at $5 \times 5$ nodes). Increasing the number of subdomains implies that each node must process a larger number of subdomains. In this case, it can be observed that efficiency is higher as more subdomains are used. This fact makes sense because we increase the workload of each node relative to the communication time for their synchronization. However, we also observe a performance decline for the largest problem ($4 \times 16$ subdomains). This is due, once again, to memory access limitations.

In conclusion, the scalability of parallelization is not optimal. In all cases, it can be seen that performance only increases up to the use of 4 nodes, and only a reduction of time by half is achieved. Several factors may cause this issue. The main one is due to the used hardware, the workload distribution among CPUs is transparent to the user and that makes it difficult to know how the computation is being carried out. A performance drop is particularly observed when the number of nodes approaches the machine's maximum of 8. However, in Figure 5b, we can observe that up to the second node, before overloading the machine, the scalability seems to be satisfactory. This indicates that, with the right infrastructure, the code could scale more efficiently.

## 4.2 Effect on design of subdomains

The primary objective of FETI methods is not merely to parallelize the problem to reduce computation time but to provide a method that scales with the problem size. Hence, in the work of [2], the effect of the subdomain design on scalability is studied. We define $H$ as the size of the subdomains, and in a 2D problem, it can be expressed as $N_s = O(1/H^2)$. The mesh size can be defined as $h$, and the problem size is approximated as $O(1/h^2)$. As a result, the following conditioning number, $\kappa$, is introduced for a 2D problem as shown below.

$$\kappa = O\left(1 + \log^2(H/h)\right) \tag{27}$$

This number allows us to define the approximate increase in the number of iterations required for a problem to converge. In this case, it is defined in terms of the subdomains and can be understood in the following manner:

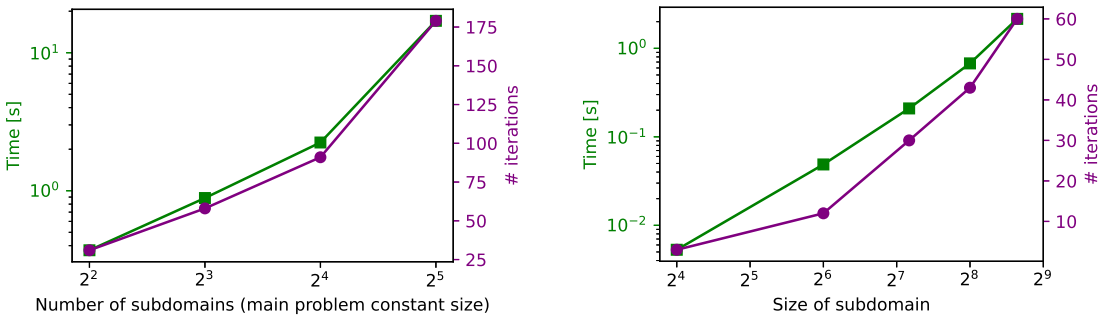
- When the number of subdomains $N_s$ (i.e., $O(1/H^2)$) is held constant and the size of the overall problem $O(1/h^2)$ is increased, the conditioning number asymptotically increases at the rate of $\log^2 1/h$. This implies that as we increase the subdomain size, the problem will take slightly longer to converge. However, its asymptotic nature indicates that scalability to large-sized problems is very good. In other words, a problem with small $N_r$ is solved as efficiently as one with larger $N_r$.

- When the problem size is kept constant and the number of subdomains that we are capable of dividing the problem into increases, the number of iterations is expected to decrease. In this case, although the size of the coarse problem will increase (as there will be more global nodes $P$ to solve), the difficulty of converging the solution for the local problem decreases.

- When the subdomain size, which can be expressed as $H/h$, is kept constant, the conditioning number remains constant. Therefore, it can be expected that for a problem with the same number of subdomain groups as CPU nodes, the time required to solve it should not increase.

In this case, the results have not turned out as expected. We present Figure 6 obtained for a regular 2D structural problem to study these conditions.

In Figure 6a, a problem with a fixed number of degrees of freedom $N = 2000$ and a constant tolerance $tol = 1 \times 10^{-8}$ was solved. It can be observed that the time increases as we increase the number of subdomains. This is precisely the opposite of what was expected. Although the size of the coarse problem increases, the resolution of the problem within the subdomain is simplified to a greater extent. Therefore, a decrease in the number of iterations needed to reach the solution's convergence should be observed. A possible explanation for this




(a) Increasing $N_s$ with fixed $N$.

(b) Increasing $N_r$ with fixed $N_s$.

Figure 6: Subdomains properties effect on time and iterations.

outcome could be the non-utilization of the Preconditioned Conjugate Gradient. Not using the preconditioner leads to an increase in the conditioning number, hence increasing the number of iterations.

The result shown in Figure 6b is more in line with expectations. In this case, a fixed size of the number of subdomains, $N_s = 9$ (in a 2D layout of $3 \times 3$) was used, and the problem size within each subdomain was increased (i.e., the size of $N_r$ increased). Evidently, the number of iterations grows as this happens because the problem in each subdomain is more complex (its conditioning number increases). Therefore, the result obtained in this Figure is more consistent, although it also grows at a faster rate than expected by Expression 27.

In general, possible improvements include the following points. Firstly, not using a preconditioner can significantly influence the number of iterations needed. In addition, all the tested problems are of relatively small size due to the infrastructure used. These types of methods are designed to solve large-scale problems, and that's where their best performance can be appreciated. Another potential improvement could lie in the algorithm programming. It is likely that some optimizations can be performed to maximize the performance of this method.

# 5 Conclusions

In this project, a customized implementation of the Finite Element Tear and Interconnection - Dual Primal (FETI-DP) method, capable of solving structural problems similar to those posed by architectural metamaterials, has been successfully developed. The mathematical foundation of the method has been presented and adapted, highlighting its ability to divide the problem into more easily solvable subdomains. This approach is well suited to the regular lattice-like structures of architected metamaterials, allowing to treat each unit cell of the lattice as an independent subdomain.

Code results have been presented, demonstrating the increased speedup when performing computations on subdomains in parallel. This efficiency is a testament to the power of the FETI-DP method and its suitability for solving large-scale problems in computational mechanics, particularly those involving architected metamaterials.

However, there is room for improvement. The implemented method does not seem to be as scalable as the one presented in the literature, probably due to differences in the implementation and the absence of a preconditioner in the code. Future work should focus on resolving these issues to further improve the efficiency and scalability of the method.

Despite these difficulties, the project has made significant advances in our understanding of architectural metamaterials and in our ability to predict their behavior under different loading conditions. The computational tool developed not only deepens our understanding of these innovative materials, but also lays the basis for their wider application in a multitude of engineering domains. Furthermore, the versatility of the FETI-DP method suggests potential applications beyond structural analysis, opening up new avenues of exploration in future work.

## Acknowledgments

I would like to express my gratitude to Professor P. Antolín Sánchez for his invaluable guidance throughout this project. His assistance in understanding the problem, providing necessary data, and helping with the code was instrumental to the success of this work.

# References


[1] Gene M. Amdahl. Validity of the single pro cessor approach to achieving large scale computing capabilities. *AFIPS spring joint computer conference*, 1967.

[2] Charbel Farhat, Michel Lesoinne, Patrick Letallec, Kendall Pierson, and Daniel Rixen. FETI-DP: A dual-primal unified FETI method part I: A faster alternative to the two-level FETI method. *International Journal for Numerical Methods in Engineering*, 50(7):1523–1544, 2001.

[3] Xiaoyu Zheng, Howon Lee, Todd H. Weisgraber, Maxim Shusteff, Joshua DeOtte, Eric B. Duoss, Joshua D. Kuntz, Monika M. Biener, Qi Ge, Julie A. Jackson, Sergei O. Kucheyev, Nicholas X. Fang, and Christopher M. Spadaccini. Ultralight, ultrastiff mechanical metamaterials. *Science*, 344(6190):1373–1377, 2014.