Hybridization of FETI Methods

Roberto Molina

Université Pierre et Marie Curie

Doctoral School **ED Sciences Mathématiques Paris Centre**

University Department **LJLL Laboratoire Jacques-Louis Lions**

Thesis defended by **Roberto Molina**

Defended on **19th December, 2017**

In order to become Doctor from Université Pierre et Marie Curie

Academic Field **Mathematics**

Thesis Title

Hybridization of FETI Methods

Thesis supervised by François-Xavier Roux

Committee members

Referees	Daniel J. Rixen	Professor at Technische Universität München	
	Luc Giraud	Senior Researcher at INRIA	
Examiners	Frédéric Hecht	Professor at LJLL	Committee President
	Nicole Spillane	Junior Researcher at École Polytechnique	
	Pierre Gosselet	Junior Researcher at CNRS École Normale Superieure Paris-Saclay	
Supervisor	François-Xavier Roux	Professor at LJLL	

Université Pierre et Marie Curie

Doctoral School **ED Sciences Mathématiques Paris Centre**

University Department **LJLL Laboratoire Jacques-Louis Lions**

Thesis defended by **Roberto Molina**

Defended on **19th December, 2017**

In order to become Doctor from Université Pierre et Marie Curie

Academic Field **Mathematics**

Thesis Title

Hybridization of FETI Methods

Thesis supervised by François-Xavier Roux

Committee members

Referees	Daniel J. Rixen	Professor at Technische Universität München	
	Luc Giraud	Senior Researcher at INRIA	
Examiners	Frédéric Hecht	Professor at LJLL	Committee President
	Nicole Spillane	Junior Researcher at École Polytechnique	
	Pierre Gosselet	Junior Researcher at CNRS École Normale Superieure Paris-Saclay	
Supervisor	François-Xavier Roux	Professor at LJLL	

Université Pierre et Marie Curie

École doctorale **ED Sciences Mathématiques Paris Centre**

Unité de recherche **LJLL Laboratoire Jacques-Louis Lions**

Thèse présentée par **Roberto Molina**

Soutenue le **19 décembre 2017**

En vue de l'obtention du grade de docteur de l'Université Pierre et Marie Curie

Discipline **Mathématiques**

Titre de la thèse

Hybridation de mèthodes FETI

Thèse dirigée par François-Xavier Roux

Composition du jury

Rapporteurs	Daniel J. Rixen	professeur au Technische Universität München	
	Luc Giraud	directeur de recherche à l'INRIA	
Examinateurs	Frédéric Hecht	professeur au LJLL	président du jury
	Nicole Spillane	chargé de recherche à l'École Polytechnique	
	Pierre Gosselet	chargé de recherche au CNRS École Normale Superieure Paris-Saclay	
Directeur de thèse	François-Xavier Roux	professeur au LJLL	

Keywords: numerical analysis, domain decomposition methods, algebra, scientific computation

Mots clés : analyse numérique, méthodes de décomposition de domaine, algèbre, sciences numériques

This thesis has been prepared at

LJLL Laboratoire Jacques-Louis Lions
Laboratoire Jacques-Louis Lions
4 place Jussieu
Université Pierre et Marie Curie
Boîte courrier 187
75252 Paris Cedex 05
France

☎ (33)(0) 1 44 27 42 98
(33)(0)1 44 27 72 00
Web Site `http://www.ljll.math.upmc.fr/`

This thesis is dedicated to all the people who, in one way or another, were part of this great journey.

Hybridization of FETI Methods

Abstract

In the domain decomposition framework, classic non-overlapping methods such as FETI and its variations are often used in industrial applications due to better performance and properties shown by them. Several improvements have been developed in order to achieve these good results. Following this line of work we present new methods based on FETI and new implementations of the recently developed S-FETI method.
The classics FETI-1LM and FETI-2LM methods are used to build a new hybrid FETI method as a fusion of both of them. These methods differ from one another in the boundary condition imposed on the interface. The basic idea is to develop a new algorithm that can use both methods at the same time by choosing in each interface the most suited condition depending on the characteristics of the problem. We search to have a faster and more robust code that can work with configurations that the base methods will not handle optimally by themselves. The performance of this method is tested on a problem with a contact surface, where most of the numerical issues are focused.
In the following part, we present a new implementation for the S-FETI method. As seen in the original presentation, the full reorthogonalization needed by S-FETI, and all FETI methods in general, impose the storage of all the directions created, making the memory cost somehow a relevant issue. The objective is to reduce the memory usage of this method, allowing this method to work on larger problems. We achieve this by using the sparsity properties of the search directions. We also propose other variations to lower the storage, but this time by reducing the directions saved at each iteration. Finally, an extension of the FETI-2LM method to his block version as in S-FETI is developed. Numerical results for the different algorithms are presented.

Keywords: numerical analysis, domain decomposition methods, algebra, scientific computation

LJLL Laboratoire Jacques-Louis Lions
Laboratoire Jacques-Louis Lions – 4 place Jussieu – Université Pierre et Marie Curie – Boîte courrier 187 – 75252 Paris Cedex 05 – France

Acknowledgements

I would like to thank first, to my family. Even in the distance, their support is my greatest strength.

I would also like to thank my advisor François-Xavier Roux. Thanks to him and his way to address the work, made these years in France of great professional learning, but also of personal growth.

Thanks to all the people from the University, colleagues, and administration. Their daily presence made of my work activities a great pleasure.

Finally, I will like to thanks to all the friends I made during my life in Paris. They were my happiness in the good days and my escape from the bad ones. Thank you very much, and I hope one day I will be able to repay you.

Contents

List of Tables

List of Figures

Introduction

In the last decades and thanks to the growth of raw computational power, faster, more robust and accurate algorithms had been developed to solve numerically a large variety of problems modeled by Partial Differential Equations (PDE). The use of multiple processors to increase the speed of calculations leads to the search for strategies in parallelism that allows profiting from these new computer architectures. Different parallel iterative and direct methods for solving linear systems have been developed [70],[24],[3], both with different trade-off regarding speed, memory, and accuracy. Results of iterative methods based on Krylov spaces usually depend on the eigenvalue distribution of the matrix representing the system, and for most problems, the memory requirements are not an issue. On the other hand, direct methods are more robust, but the use of memory can be an issue for some large systems.

For problems coming from the discretization of Finite Element Methods, we have properties that allow us to use different approaches. Considered a hybrid between iterative and direct methods [59], [18] these are the Domain Decomposition Methods (DDM). They are based on the partition of the domain of the problem into subdomains, where a smaller system of equations is defined. From this division, these methods are categorized into two large groups, the overlapping and non-overlapping methods.

In this work we will focus in some DDM with non-overlapping subdomains, mainly the Finite Elements Tearing and Interconnecting (FETI) and other related methods [34],[30],[31].

The primary objective of this thesis is to develop new FETI methods, improving the results shown by the existing algorithms in as much number of cases as it is possible. Also, we will expand the results on one modern FETI method to

extend its application to problems in which the current formulation does not allow it.

Contributions of this thesis

The following work is based on one of the best known non-overlapping domain decomposition method, the Finite Element Tearing and Interconnecting method [34],[33]. Since its first appearance, many have been the works based on this method, producing several improvements to it [27],[61] and at the same time enabling the formulation of new FETI methods [30],[31]. Within this context, today we can count with two of the most used FETI methods, namely the original FETI-1LM and the later developed FETI-2LM (both including the enhancements already done to them). We start our work presenting a new FETI method, based on these two methods and the similarities shared by them. In this new algorithm, we try to take advantage and combine the good properties of both individual base methods. After the presentation of Hybrid-FETI, and after a better understanding of the same, we can exhibit its advantages in some cases, in particular, the contact problems, where usually outperforms the FETI-1LM and FETI-2LM methods.

In a different line of work, this time following the development of a recent FETI method, the so-called Simultaneous-FETI (S-FETI) [38], we will continue the analysis of it, to find new, faster or more robust algorithms. Different variations will be formulated and tested, all of them trying to improve the existing results shown by the method, mainly in terms of memory use, but also regarding speed and robustness. We will extend the application of S-FETI to a broader class of problems by developing a new implementation based on a sparse storage. This new algorithm will reduce the memory limitations of the method.

The last parts of this thesis show some ideas are worth exploiting, with preliminary results and under current development. Based on the properties beneath the formulation of S-FETI, we try to develop new FETI algorithms of the same type. For example, we use the same block construction strategy used to build S-FETI starting from FETI, but this time we apply it to the FETI-2LM

method, leading to a new block version of this last one. This method along with some other ideas are still in its basic stage, so they present several issues; nonetheless, they may lead to new sources of research.

Before giving more details about FETI and the other FETI-like methods, we want to recall some of the essential linear algebra tools needed in this thesis to understand this type of algorithms. We refer to the iterative solvers for linear systems which are key elements in the different FETI methods. We will also link these ideas against the work in domain decomposition methods in general.

Iterative Methods

We start by showing the fundamental properties of the iterative *Krylov Methods* used in the solution of FETI problems.

Krylov Methods

In this section, we are interested in solving the following general problem

$$Ax = b \tag{1}$$

With $A \in \mathcal{M}_{n\times n}(\mathbb{R})$ a square real matrix, x the vector of unknown, b a vector of known values also called right-hand side (RHS), both terms are $\mathbb{R}^n$ vectors.

A big part of our work is based on the solution of such a linear system (the FETI operator) via an iterative method. First, we will consider the more general case where A is invertible, meaning that our system has a unique solution. The most frequent iterative methods used to solve Equation (1) are the Krylov methods, a name due to the fact that they are based on projections in a particular type of space, the *Krylov Spaces*.

The Krylov Space is defined by

Definition 0.1. *Let us consider x_0 as an initial solution of (1). A Krylov space, denoted by $\mathcal{K}_p$ is the space generated by the residual $g_0 := Ax_0 - b$ and its successive*

$p-1$ *iterative products*

$$\mathcal{K}_p = Span\{g_0, Ag_0, A^2 g_0, ..., A^{p-1} g_0\}$$

The Krylov methods consist of building and projecting our solution in this subspace, all by just using simple operations such as matrix-vector products, dot products or linear combinations of vectors. In the following, we will recall the proof of theoretical convergence of the Krylov methods and how to build in practice the solution (or approximated solution) of the system (1).

We note that this family of subspaces is strictly increasing and bounded, so it has a maximal dimension that we will call p_{max}. Also, from the definition of the subspaces $\mathcal{K}_p$ we have the next properties

Lemma 0.2. *If $A^p g_0 \in \mathcal{K}_p$ then $A^{p+q} g_0 \in \mathcal{K}_p$ for every $q > 0$*

Proof. By induction. For $q \geq 0$, if we have $A^{p+q} g_0 \in \mathcal{K}_p$ then $A^{p+q} g_0 = \sum_{k=0}^{p-1} \alpha_k A^k g_0$ and therefore

$$\begin{aligned} A^{p+q+1} g_0 &= \sum_{k=0}^{p-2} \alpha_k A^{k+1} g_0 + \alpha_{p-1} A^p g_0 \\ &= \sum_{k=0}^{p-2} \alpha_k A^{k+1} g_0 + \alpha_{p-1} \sum_{k=0}^{p-1} \beta_k A^k g_0 \\ &= \sum_{k=0}^{p-1} \gamma_k A^k g_0 \end{aligned}$$

□

Lemma 0.3. *The Krylov space succession is strictly increasing from 1 to p_{max} then it stagnates from $p = p_{max}$*

Proof. If p is the smallest integer that makes $A^p g_0$ dependent of previous vectors, then the vectors $(g_0, Ag_0, A^2 g_0, \ldots, A^{p-1} g_0)$ are linearly independent and $\mathcal{K}_q$ has a dimension of q, for every $q \leq p$. In particular, $\mathcal{K}_p$ has a dimension p. Furthermore,

$A^p g_0 \in \mathcal{K}_p$ and, from Lemma 0.2, every vector $A^{p+q} g_0$ is in $\mathcal{K}_p$, for every $q > 0$, which implies that $\mathcal{K}_{p+q} = \mathcal{K}_p$ for every $q > 0$. Then we have: $\mathcal{K}_1 \subset \cdots \subset \mathcal{K}_p = \mathcal{K}_{p+q}$ for every $q > 0$. And by definition of p_{max}, we have that $p = p_{max}$ □

Theorem 0.4. *The solution of the linear system $Ax = b$ is in the affine space $x_0 + \mathcal{K}_{p_{max}}$*

Proof. From Lemma 0.2 and Lemma 0.3 the vectors $(g_0, Ag_0, A^2 g_0, \ldots, A^{p_{max}-1} g_0)$ are linearly independent and

$$A^{p_{max}} g_0 = \sum_{k=0}^{p_{max}-1} \alpha_k A^k g_0 \tag{2}$$

In this equation, the coefficient α_0 is non null, from which, multiplying both terms by A^{-1} we obtain

$$A^{p_{max}-1} g_0 = \sum_{k=0}^{p_{max}-1} \alpha_k A^{k-1} g_0$$

which is contradictory with the linear dependency of the vectors. If we divide both terms in Equation (2) by α_0 and we pass all terms to one side, we have

$$g_0 + \sum_{k=1}^{p_{max}-1} \frac{\alpha_k}{\alpha_0} A^k g_0 - \frac{1}{\alpha_0} A^{p_{max}} g_0 = 0 \Leftrightarrow$$

$$Ax_0 - b + \sum_{k=1}^{p_{max}-1} \frac{\alpha_k}{\alpha_0} A^k g_0 - \frac{1}{\alpha_0} A^{p_{max}} g_0 = 0 \Leftrightarrow$$

$$A\left(x_0 + \sum_{k=1}^{p_{max}-1} \frac{\alpha_k}{\alpha_0} A^{k-1} g_0 - \frac{1}{\alpha_0} A^{p_{max}-1} g_0 \right) = b \Leftrightarrow$$

□

In practice, to build these spaces all we have to do is to compute the basis of the space, but we will never use the "natural" base because it degenerates numerically as it grows. In practice, if we use the regular double precision in a standard machine, after a few iterations the new values of the succession $A^p g_0$

start to be linearly dependent, and depending on the matrix A some values are too small or too big to be represented.

With this in consideration, we need to find another way to reconstruct the space. To do so, we build different basis, for example, the one called *Basis of Arnoldi* which has much better numerical properties of representation and stability. This base is constructed applying the modified *Gram-Schmidt* orthonormalization procedure to the successive matrix products. The algorithm defined in (1) illustrates this procedure.

Algorithm 1 Arnoldi iteration algorithm

Initialization
$g_0 = Ax_0 - b$
$v_0 = \frac{g_0}{\|g_0\|}$
loop Construction of the $p+1$ vector of the base
    $w = Av_p$
    for $i = 0$ to p **do**
        $\alpha_i = (w, v_i)$
        $w = w - \alpha_i v_i$
    end for
    $v_{p+1} = \frac{w}{\|w\|}$
end loop

Let V_p be the space of the first p basis constructed of the Krylov space. The next problem is to find an approximate solution x_p of the system using this basis. From Theorem 0.4, we know that the exact solution is in $x_0 + \mathcal{K}_{p_{max}}$, but in practice the approximate solution is found using limited arithmetic, so we search a projected solution in the space $x_0 + \mathcal{K}_p$. This way, the solution can be written as

$$x_p = x_0 + V_p z_p \tag{3}$$

where z_p is a p dimension vector. This approximation allows the writing of the error e_p and residual vectors g_p as

$$\begin{aligned} e_p &:= x_p - x = x_0 - x + V_p z_p = e_0 + V_p z_p \\ g_p &:= Ax_p - b = Ae_p = Ae_0 + AV_p z_p = g_0 + AV_p z_p \end{aligned} \tag{4}$$

A Krylov method consist, in one hand, of an algorithm to compute a base for the Krylov space, and on the other side, an optimal criterion to determine the approximate solution x_p. This criterion usually is to minimize the error or residual using some adapted norm.

Lanczos method

We will now explain how to build the solution in the particular case when A is a symmetric matrix. Let h_{ij} be the coefficient of orthogonalization of Av_j against v_i. Let $h_{j+1,j}$ be the norm of the vector w we get after the orthogonalization. Finally let V_p be the matrix of the p first vectors of the Arnoldi's base, then we have that

$$V_p^T V_p = I_p$$
$$AV_p = V_{p+1} H_{p+1,p}$$

where

$$H_{p+1,p} := \begin{bmatrix} h_{11} & h_{12} & \dots & \dots & h_{1p} \\ h_{21} & h_{22} & \dots & \dots & h_{2p} \\ 0 & h_{32} & h_{33} & \dots & h_{3p} \\ \vdots & \ddots & \ddots & \dots & \vdots \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & 0 & h_{p+1,p} \end{bmatrix}$$

and so

$$H_p := H_{pp} = V_p^T A V_p$$

For the case of a symmetric matrix, H_p is also symmetric, therefore tridiagonal, so the algorithm for the construction of the Basis of Arnoldi is simplified. The method described in Algorithm 2 shows the construction of the basis of Arnoldi for symmetric matrices, now called *Basis of Lanczos*. This algorithm has the property of only using short recurrences in the computation, so its cost is constant in every iteration.

If the matrix is also positive definite, the *Lanczos Method* consist on minimiz-

Algorithm 2 Algorithm of Lanczos

Initialization
$g_0 = Ax_0 - b$
$v_0 = \frac{g_0}{\|g_0\|}$
loop Construction of the $p+1$ vector of the base of Lanczos
    $w = Av_p$
    $h_{p,p-1} = h_{p-1,p}$
    $w = w - h_{p-1,p}v_{p-1}$
    $h_{pp} = (w \cdot v_p)$
    $w = w - h_{pp}v_p$
    $h_{p+1,p} = \|w\|$
    $v_{p+1} = \frac{w}{\|h_{p+1,p}\|}$
end loop

ing the error in the norm induced by A

$$\mathcal{E}(x_p) = \|x_p - x\|_A^2 = (A(x_p - x) \cdot (x_p - x)) = (g_p \cdot e_p) \tag{5}$$

The solution constructed using the Lanczos method has the following properties

Theorem 0.5. *The approximate solution x_p of the Lanczos method is the projection of x in $x_0 + \mathcal{K}_p$ for the inner product derived from A.*

Proof. From (5), x_p is the element from $x_0 + \mathcal{K}_p$ which has a distance to x minimal in the A-norm.

□

Corollary 0.6. *The residual vector $g_p = Ax_p - b$ of the Lanczos method is orthogonal to $\mathcal{K}_p$.*

Proof. Direct of the properties of projections in affine spaces. □

All that is missing is the practical computation of x_p. To do so, we know from (5) and (4) that

$$\mathcal{E}(x_p) = (A(e_0 + V_p z_p) \cdot (e_0 + V_p z_p)) = (AV_p z_p \cdot V_p z_p) + 2(g_0 \cdot V_p z_p) + (g_0 \cdot e_0) \tag{6}$$

To minimize this, we only need to use the part that depends on z_p and so the problem reduces to the minimization of the functional

$$\begin{aligned}\mathcal{J}_p(z_p) &= \frac{1}{2}(V_p^T A V_p z_p \cdot z_p) + (V_p^T g_0 \cdot z_p) \\ &= \frac{1}{2}(T_p z_p \cdot z_p) + (y_p \cdot z_p)\end{aligned}$$

where $T_p = H_p$ is the matrix of the orthonormalization coefficients. This is a classical minimization problem in finite dimension, from where we have the following results

Lemma 0.7. *If A is a symmetric positive definite matrix then $\mathcal{J}(x) = \frac{1}{2}(Ax \cdot x) - (b \cdot x)$ is strictly convex*

Proof.

$$\begin{aligned}&\mathcal{J}(\alpha x + (1-\alpha)y) = \\ &= \frac{1}{2}\alpha^2(Ax \cdot x) + \alpha(1-\alpha)(Ax \cdot y) + \frac{1}{2}(1-\alpha)^2(Ay \cdot y) - \alpha(b \cdot x) - (1-\alpha)(b \cdot y) \\ &= \alpha\mathcal{J}(x) + (1-\alpha)\mathcal{J}(y) + \\ &\quad + \frac{1}{2}\Big[(\alpha^2 - \alpha)(Ax \cdot x) + 2\alpha(1-\alpha)(Ax \cdot y) + ((1-\alpha)^2 - (1-\alpha))(Ay \cdot y)\Big] \\ &= \alpha\mathcal{J}(x) + (1-\alpha)\mathcal{J}(y) + \frac{1}{2}\alpha(\alpha-1)\left[(Ax \cdot x) - 2(Ax \cdot y) + (Ay \cdot y)\right]\end{aligned}$$

A is positive definite, so we have

$$(Ax \cdot x) - 2(Ax \cdot y) + (Ay \cdot y) = (A(x-y) \cdot (x-y)) > 0$$

whenever $x \neq y$. Now, if $\alpha \in]0,1[$ then $\alpha(1-\alpha) < 0$, hence

$$\frac{1}{2}\alpha(\alpha-1)\left[(Ax \cdot x) - 2(Ax \cdot y) + (Ay \cdot y)\right] < 0$$

□

Theorem 0.8. *The functional $\mathcal{J}(x) = \frac{1}{2}(Ax \cdot x) - (b \cdot x)$ admits and absolute minimum x that also verifies $Ax = b$.*

Proof. $\mathcal{J}$ is strictly convex and lower bounded, because $\mathcal{J}(x) \to +\infty$ when $\|x\| \to +\infty$. It is obviously differentiable with a value of

$$\mathcal{D}\mathcal{J}(x) \cdot y = (Ax \cdot y) - (b \cdot y) = ((Ax - b) \cdot y)$$

This functional has an absolute minimum in the unique point where its gradient is zero, which is the point where $Ax - b = 0$. □

Corollary 0.9. *The minimum of $\mathcal{E}(x_p)$ defined in (6) is the point $x_p = x_0 + V_p z_p$, z_p being the solution of the system*

$$T_p z_p = -y_p.$$

Proof. From previous theorem and lemma, all we have to prove is that $T_p = V_p^T A V_p$ is positive definite which comes directly from the fact that the matrix A is also positive definite and the vectors V_p are linearly independent. □

The Lanczos method consists on building T_p and y_p then find z_p and replacing it in Equation (3) to obtain the approximate solution x_p that minimizes the error in the A-norm.

One of the good things of this method is that the vectors of the basis are calculated using a short recurrence, but the main drawback is that for the computation of x_p we need to solve a bigger system every step, so the cost grows as the number of iteration increases.

Conjugate Gradient

Using the Krylov space approach, in this part, we introduce one of the most known iterative algorithms, the Conjugate Gradient method (CG). It can be considered as an improvement to the Lanczos method, in fact, the Lanczos method will be a much better algorithm if it could use a short recurrence for the calculation of the approximated solution x_p. To construct this, we need that the first component of z_p are the ones of z_{p-1} which will give a formula of the type

$$x_p = x_{p-1} + \alpha_p v_p$$

In order to do so, the base of the Krylov space must be one in which the projection matrix $W_p^T A W_p$ is a diagonal one. However, we will not be able to compute the error $\mathcal{E}(x_p)$ because e_0 will be unknown. A practical way to test the convergence of the method is to use the following dimensionless residual

$$\frac{\|Ax_p - b\|}{\|b\|} < \epsilon$$

The previous relation implies the need of computing the successive gradients to control the method. From the fact that $g_p \in \mathcal{K}_{p+1} \cap \mathcal{K}_p^{\perp}$ we have that $g_p = \rho v_{p+1}$, so rather than using the orthonormal base of vectors v_p we can use the orthogonal base of the gradients. Even if this basis goes to zero, there is no practical issue because the method will stop before any representation problem.

Let $G_p = (g_0, g_1, \ldots, g_{p-1})$, we know that, $G_p = V_p \Delta_p$ where Δ_p is a diagonal matrix. The projection matrix $G_p^T A G_p$ is also tridiagonal symmetric positive definite

$$G_p^T A G_p = \Delta_p^T V_p^T A V_p \Delta_p = \Delta_p T_p \Delta_p = \tilde{T}_p$$

which at the same time, admits the factorization $\tilde{T}_p = \tilde{L}_p \tilde{D}_p \tilde{L}_p^T$, again with $\tilde{D}_p$ a diagonal matrix, therefore

$$G_p^T A G_p = \tilde{L}_p \tilde{D}_p \tilde{L}_p^T \Leftrightarrow \tilde{L}_p^{-1} G_p^T A G_p \tilde{L}_p^{-t} = \tilde{D}_p$$

The previous equations shows that the matrix $W_p = G_p \tilde{L}_p^{-t}$ made of linear combinations of G_p is a A-orthogonal base of $\mathcal{K}_p$.

Considering that the projection matrix $W_p^T A W_p$ is diagonal, this basis is ideal to use in the computation of the solution of the optimization problem (5), as it can be built using the relation

$$W_p \tilde{L}_p^T = G_p \tag{7}$$

Let $(w_0, w_1, \ldots, w_{p-1})$ be the column vectors of W_p and $(\gamma_0, \gamma_1, \ldots)$ the sub-diagonal elements of $\tilde{L}_p^{-t}$, the Equation (7) implies

$$w_0 = g_0 \text{ and } \gamma_{j-1} w_{j-1} + w_j = g_j \; \forall j > 0 \tag{8}$$

With the different relations between x_p, g_p and w_p we can formulate a new method using only short recurrences in the construction of every new vector. In fact, from the previous equation, we have

$$g_0 = Ax_0 - b \text{ and } w_0 = g_0$$

Also, from the properties of the base W_p, we know that

$$x_p = x_{p-1} + \rho_{p-1} w_{p-1} \Leftrightarrow g_p = g_{p-1} + \rho_{p-1} A w_{p-1}$$

Considering that g_p is orthogonal to both $\mathcal{K}_p$ and w_{p-1}, we can obtain the value for ρ_{p-1}

$$(g_p \cdot w_{p-1}) = (g_{p-1} \cdot w_{p-1}) + \rho_{p-1}(Aw_{p-1} \cdot w_{p-1}) = 0 \Leftrightarrow \rho_{p-1} = -\frac{(g_{p-1} \cdot w_{p-1})}{(Aw_{p-1} \cdot w_{p-1})}$$

From Equation (8) we can build the new w_p with the previous w_{p-1} and g_p

$$w_p = g_p - \gamma_{p-1} w_{p-1}$$

The coefficient γ_{p-1} is also computed using the A-orthogonality relation between w_p and w_{p-1}

$$(w_p \cdot Aw_{p-1}) = (g_p \cdot Aw_{p-1}) - \gamma_{p-1}(w_{p-1} \cdot Aw_{p-1}) = 0 \Leftrightarrow \gamma_{p-1} = \frac{(g_p \cdot Aw_{p-1})}{(Aw_{p-1} \cdot w_{p-1})}$$

The method defined this way is called the $Conjugate\,Gradient$ method [44] and it can be summarized in Algorithm 3.

Remark: From a theoretical point of view, we do not need any orthogonalization to build the new descent direction w_p, but in finite precision arithmetic, errors are transmitted from each actualization. Therefore, in practice, to apply this method in the domain decomposition framework, we will need to reconjugate the vectors of the base, and so the storage of these vectors for a robust method is mandatory [66]. Considering this, the lines (11), (12) from the Algorithm 3 are now replaced by the loop necessary to build the coefficients γ_i and

Algorithm 3 Conjugate Gradient method

Initialization
$g_0 = Ax_0 - b$
$w_0 = g_0$
loop Iteration of the CG method
    $\rho_{p-1} = -(g_{p-1} \cdot w_{p-1})/(Aw_{p-1} \cdot w_{p-1})$
    $x_p = x_{p-1} + \rho_{p-1} w_{p-1}$
    $g_p = g_{p-1} + \rho_{p-1} Aw_{p-1}$
    if $(g_p \cdot g_p)/(b \cdot b) < \epsilon^2$ **then**
        End
    end if
    $\gamma_{p-1} = (g_p \cdot Aw_{p-1})/(Aw_{p-1} \cdot w_{p-1})$
    $w_p = g_p - \gamma_{p-1} w_{p-1}$
end loop

the reorthogonalized directions

$$\gamma_i = \frac{(g_p \cdot Aw_i)}{(Aw_i \cdot Aw_i)}, \quad i = 0, \ldots, p-1; \qquad w_p = g_p - \sum_{i=0}^{p-1} \gamma_i w_i$$

ORTHODIR

The following iterative method will also be presented using the Krylov spaces approach. The difference with the Conjugate Gradient method is that it can be used in cases where the matrix A is no longer symmetric. This is the case of some FETI methods such as FETI-2LM, but more details will be given in the following chapter.

In the case of A being a non-symmetric matrix, the H_p matrix is no longer tridiagonal. There is no short recurrences to build an orthogonal basis of the Krylov space $\mathcal{K}_p$. Also, A does not define an inner product, and the criterion of optimality $\mathcal{E}(x_p)$ may not be used. In this case, the choice for a stopping criterion is the computation of the square norm of the residual

$$\mathcal{R}(x_p) = (A(x_p - x) \cdot A(x_p - x)) = (g_p \cdot g_p) = \|x_p - x\|^2_{A^T A} = (A^T A(x_p - x) \cdot (x_p - x)) \quad (9)$$

We have similar properties to the symmetric case for the approximate solution

that minimizes $\mathcal{R}(x_p)$

Theorem 0.10. *The approximate solution x_p that minimizes $\mathcal{R}(x_p)$ in $x_0 + \mathcal{K}_p$ is the projection of x for the inner product associated with $A^T A$.*

Proof. Direct from Equation (9). □

Corollary 0.11. *The residual vector $g_p = Ax_p - b$ is orthogonal to $A\mathcal{K}_p$.*

Proof. From the properties of projections in affine spaces

$$(A^T A(x_p - x) \cdot w_p) = (A(x_p - x) \cdot Aw_p) = (g_p \cdot Aw_p) = 0, \forall w_p \in \mathcal{K}_p$$

□

We have naturally introduced the scalar product defined by $A^T A$ that is symmetric and positive definite if A is invertible. We could think that it would be appropriate to use the Conjugate Gradient method to the system

$$A^T Ax = A^T b$$

This equation is called the "$Normal\,equation$" and it does not have a very good conditioning. Due to its definition, it can be as much as the square of the original conditioning of A.

The best is to compute the solution using short recurrences, so from the Theorem 0.10 we know that any $A^T A$-orthogonal basis W_p of $\mathcal{K}_p$ implies that $W_p^T A^T A W_p$ should be diagonal.

If we look at the structure of $H_{p+1,p}$ the matrix $H_{p+1,p}^T H_{p+1,p}$ is full, in this case the previous basis has no use. To build a basis $A^T A$-orthogonal we only need to apply the modified Gram-Schmidt procedure to the vectors obtained by successive multiplication for the matrix, using the $A^T A$-norm. With this considerations, we have the following short recurrences

$$x_p = x_{p-1} + \rho_p w_p$$
$$g_p = g_{p-1} + \rho_p A w_p$$

From the A^TA-orthonormal properties we have

$$(g_p \cdot Aw_p) = 0 \Leftrightarrow (g_{p-1} \cdot Aw_p) + \rho_p(Aw_p \cdot Aw_p) = 0 \Leftrightarrow \rho_p = -(g_{p-1} \cdot Aw_p)$$

Finally, the method is described in Algorithm 4

Algorithm 4 ORTHODIR method

1: **Initialization**
2: $g_0 = Ax_0 - b$
3: $w_0 = g_0$
4: **loop** Iterate $p = 1, ...,$ until convergence
5: $\quad \rho = -\frac{(g_{p-1} \cdot Aw_{p-1})}{(Aw_{p-1} \cdot Aw_{p-1})}$
6: $\quad x_p = x_{p-1} + \rho w_{p-1}$
7: $\quad g_p = g_{p-1} + \rho Aw_{p-1}$
8: $\quad w_p = Aw_{p-1}$
9: $\quad$ **for** $i = 0$ to $p - 1$ **do**
10: $\quad\quad \gamma = -\frac{(Aw_p \cdot Aw_i)}{(Aw_i \cdot Aw_i)}$
11: $\quad\quad w_p = w_p + \gamma w_i$
12: $\quad$ **end for**
13: **end loop**

This algorithm forces to save the vectors of the basis, but this a standard in the DDM framework for numerical stability. In any case, we will deal with this issue in the next chapters.

Remark: A small variation of this method, equivalent to the regular and with the same properties can be made if we change the line (8) and build the next direction using the gradient, instead of the previous orthonormalized vector, i.e

$$w_p = g_p$$

Parallelization of Krylov methods

The codes presented previously for the CG and ORTHODIR methods represent both sequential algorithms. However, they can be transformed into their parallel counterparts.

The implementation of the parallel version of these iterative methods is based on a message-passing standard. In practical terms, this implies the use of libraries that use the most common parallel computing architectures. One of the most frequently used communication protocol is the so-called Message Passing Interface (MPI) [39].

Using this protocol, we only add two changes to the sequential CG and ORTHODIR algorithms

1. Exchange of data between processes (or subdomains, in the domain decomposition framework) to assemble the matrix-vector products. The details on how to compute these products will be given when presenting the first domain decomposition method used in this work, namely the FETI method.

2. Global reduction operations to add the contribution of each process when computing the different scalar products.

The first item show one of the fundamental links between parallel iterative and domain decomposition methods. This is more evident when we know that one of the strategies to parallelize matrix-vector products come directly from a partition in subdomains strategy [49]. In the DDM and this thesis in particular, the parallelization is direct as the matrices to solve come from the discretization of a PDE in a discrete mesh.

With the previous basic linear algebra, parallel computing and DDM considerations, we are ready to presents the main work of this thesis.

Remark: Using the MPI libraries (in general, any communication protocol needed to do exchanges in a parallel algorithm) produces synchronization points in the code that need to be reduced as much as possible to avoid major impact in the total computation time. This is also standard in the work of DDM and parallel algorithms.

Chapter 1

Hybrid FETI method

In the domain decomposition framework, the Finite Element Tearing and Interconnecting (FETI) method have proven to be very effective in the solution of real engineering applications during the last decades. For this reason, the development of FETI and other methods of the like keeps up to this day, always searching for faster, precise and more robust algorithms. In this context, we have developed a new method, based on two existing FETI methods, namely the FETI-1LM [34] and FETI-2LM [31] (as in one or two Lagrange multipliers). This new method, called Hybrid-FETI, tries to recover the good properties of each starting method, in configurations where the use of one or the other is not so clear, regarding speed or robustness. For the development of Hybrid-FETI, we need to understand the basics of both FETI methods from a theoretical point of view, but also from the implementation point of view it will be crucial to show how the new method works.

In this chapter we first introduce the FETI-1LM method in its classic version, including some considerations to reach the good performance of the method. These are mainly, the preconditioner used and the practical implementation of it. In the same way, the following part explains the FETI method with two Lagrange multipliers. Finally, we show our new algorithm that arises from combining the two previous FETI methods. The chapter ends with numerical results to asset the performance of all the methods shown in the chapter.

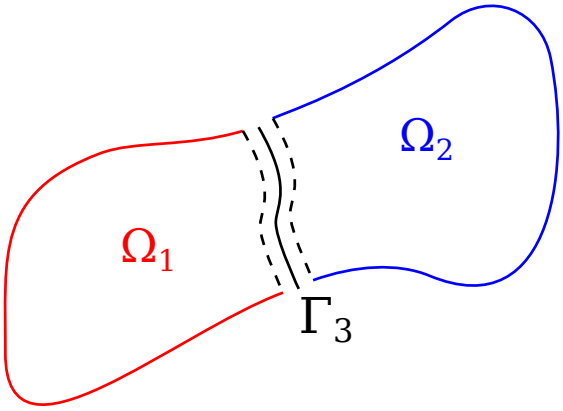

Figure 1.1 – Two subdomain splitting

1.1 Basic FETI method

1.1.1 Model problem and discretization

Model Problem

To begin, we will show the development of the method for a simple model, with the most basic domain decomposition configuration. All the ideas will be extended later to different elliptic problems and configurations. Let us first consider the Poisson problem with Dirichlet boundary condition

$$\begin{cases} -\Delta u &= f \quad in \quad \Omega \\ u &= 0 \quad on \quad \partial\Omega \end{cases} \tag{1.1}$$

where $\Omega \cap \mathbb{R}^d$, $d = 2, 3$ is a bounded domain. We need to find its variational form, so the Stokes formula is used in (1.1), and the problem is now:

Find $u \in H_0^1(\Omega)$ *such that*

$$\int_\Omega \nabla u \cdot \nabla v = \int_\Omega f v \qquad \forall v \in H_0^1(\Omega) \tag{1.2}$$

The domain is now divided into two smaller subdomains $\Omega^{(1)}$ and $\Omega^{(2)}$. Let $\Gamma^{(3)} = \partial\Omega^{(1)} \cap \partial\Omega^{(2)}$ be the interface between both subdomains as in Figure 1.1.

This division allows the formulation of two new smaller problems in each

subdomain that inherits the Dirichlet condition, or any other condition, in a part of the boundary. These problems are written as

$$\left\{\begin{array}{rcll} -\Delta u^{(s)} & = & f^{(s)} & in \quad \Omega^{(s)} \\ u^{(s)} & = & 0 & on \quad \partial\Omega^{(s)} \setminus \Gamma^{(3)} \end{array}\right. \tag{1.3}$$

with $s = 1, 2$. It is clear that the solution of (1.1) will satisfy these equations, but the contrary is not always true, due to the differences that occur in $\Gamma^{(3)}$.

We continue by using the Stokes formula again, to find the variational form of (1.3)

$$\int_\Omega \nabla u^{(s)} \nabla v^{(s)} = \int_\Omega f^{(s)} v^{(s)} + \int_{\Gamma^{(3)}} \frac{\partial u^{(s)}}{\partial n^{(s)}} v^{(s)}, \qquad \forall v^{(s)} \in H^1_{0\partial\Omega^{(s)}\setminus\Gamma^{(3)}}(\Omega^{(s)}) \tag{1.4}$$

For a function $v \in H^1_0(\Omega)$, in particular, the solution of the problem, its restriction to the subdomains $\Omega^{(s)}$ will be continuous on the interface $\Gamma^{(3)}$. On the other hand, two functions $u^{(s)}$ that satisfy the local Laplace equations (1.3) will not necessarily share the same values on $\Gamma^{(3)}$. Instead, they can be used to build a more general global solution, but not smooth enough as required.

To do this construction, but at the same time recover the unique solution of (1.1) the two variational equations (1.4) are added, giving the following variational equality

$$\int_\Omega \nabla u \nabla v = \int_\Omega f v + \int_{\Gamma^{(3)}} \left(\frac{\partial u^{(1)}}{\partial n^{(1)}} + \frac{\partial u^{(2)}}{\partial n^{(2)}} \right) v^{(3)} \qquad \forall v \in H^1_0(\Omega) \tag{1.5}$$

where $v^{(3)} = v^{(1)}|_{\Gamma^{(3)}} = v^{(2)}|_{\Gamma^{(3)}}$.

This formulation show that a new condition is necessary to have an equivalence between the solution of the global problem and that of the local ones. For u to be in $H^1_0(\Omega)$ this admissibility condition imposes the continuity on the interface

$$u^{(1)} \quad = \quad u^{(2)} \quad \text{on} \quad \Gamma^{(3)} \tag{1.6}$$

and also, for the same reason (explicit in Equation (1.5)) we need a condition on

the flux

$$\frac{\partial u^{(1)}}{\partial n^{(1)}} + \frac{\partial u^{(2)}}{\partial n^{(2)}} = 0 \quad \text{on} \quad \Gamma^{(3)} \tag{1.7}$$

In general, a non-overlapping domain decomposition method consists of two things. First, introducing boundary conditions on $\Gamma^{(3)}$ to complement the local equations (1.3). Second, iteratively find the values of these boundary conditions for which both continuity (1.6) and equilibrium (1.7) are satisfied. This last condition will imply that the local solutions will be the exact same as the global searched one (restricted to each subdomain).

Depending on the condition imposed, different basic methods can be derived, the Schur Complement method and the FETI method. (Later we will show a third method, also of the FETI type, that comes from using a different condition on the interface.)

The Schur Complement method consists in enforcing consistent Dirichlet boundary conditions on $\Gamma^{(3)}$ so the continuity condition (1.6) is automatically satisfied

$$u^{(1)} = u^{(2)} = u^{(3)} \quad on \quad \Gamma^{(3)} \tag{1.8}$$

The local Dirichlet problem to be solved in parallel for a given $u^{(3)}$ on each subdomain is

$$\begin{cases} -\Delta u^{(s)} = f^{(s)} & \text{in} \quad \Omega^{(s)} \\ u^{(s)} = 0 & \text{on} \quad \partial\Omega^{(s)} \setminus \Gamma^{(3)} \\ u^{(s)} = u^{(3)} & \text{on} \quad \Gamma^{(3)} \end{cases} \tag{1.9}$$

We have reduced the computations to find the value of $u^{(3)}$ for which the equilibrium interface condition (1.7) is satisfied. From equations (1.9), the functions $\frac{\partial u^{(1)}}{\partial n^{(1)}}$ and $\frac{\partial u^{(2)}}{\partial n^{(2)}}$ are continuous depending on $u^{(3)}$. The Schur Complement method consists in solving iteratively a condensed interface problem with $u^{(3)}$ as the unknown, and whose residual is equal to $\frac{\partial u^{(1)}}{\partial n^{(1)}} + \frac{\partial u^{(2)}}{\partial n^{(2)}}$.

The FETI method is based on enforcing consistent Neumann boundary conditions on $\Gamma^{(3)}$ so now the equilibrium interface condition (1.7) is automatically satisfied:

$$\frac{\partial u^{(1)}}{\partial n^{(1)}} = -\frac{\partial u^{(2)}}{\partial n^{(2)}} = \lambda \quad \text{on} \quad \Gamma^{(3)} \tag{1.10}$$

the local Neumann problem to be solved in parallel for a given λ on each subdomain is

$$\left\{\begin{array}{rcll} -\Delta u^{(s)} & = & f^{(s)} & \text{in} \quad \Omega^{(s)} \\ u^{(s)} & = & 0 & \text{on} \quad \partial\Omega^{(s)} \setminus \Gamma^{(3)} \\ \dfrac{\partial u^{(s)}}{\partial n^{(s)}} & = & \pm\lambda & \text{on} \quad \Gamma^{(3)} \end{array}\right. \tag{1.11}$$

Now we compute the value λ on the interface for which the continuity condition (1.6) is satisfied. From equations (1.11) we know that $u^{(1)}|_{\Gamma^{(3)}}$ and $u^{(2)}|_{\Gamma^{(3)}}$ are continuous functions depending on λ. The FETI method consists in solving iteratively a condensed interface problem to find λ and whose residual is equal to $u^{(1)}|_{\Gamma^{(3)}} - u^{(2)}|_{\Gamma^{(3)}}$.

A different interpretation of the FETI method can be considered if we see the unknown λ as the Lagrange multiplier of the continuity condition (1.6). The solution of the global variational problem (1.2) is the field u of $H_0^1(\Omega)$ that minimizes the energy functional

$$J(v) = \frac{1}{2}\int_\Omega \nabla v \cdot \nabla v - \int_\Omega f v \tag{1.12}$$

This minimization problem is equivalent to finding the couple of fields $(u^{(1)}, u^{(2)})$ of $H^1_{0\partial\Omega^{(1)}\setminus\Gamma^{(3)}}(\Omega^{(1)}) \times H^1_{0\partial\Omega^{(2)}\setminus\Gamma^{(3)}}(\Omega^{(2)})$ that minimizes the sum of the local energy functionals

$$\begin{aligned} J_1(v^{(1)}) + J_2(v^{(2)}) = & \frac{1}{2}\int_{\Omega^{(1)}} \nabla v^{(1)} \cdot \nabla v^{(1)} - \int_{\Omega^{(2)}} f^{(2)} v^{(2)} \\ & + \frac{1}{2}\int_{\Omega^{(2)}} \nabla v^{(2)} \cdot \nabla v^{(2)} - \int_{\Omega^{(2)}} f^{(2)} v^{(2)} \end{aligned} \tag{1.13}$$

under the continuity constraint we have that $u^{(1)}|_{\Gamma^{(3)}} = u^{(2)}|_{\Gamma^{(3)}}$. This condition can be written under the weak form

$$\int_{\Gamma^{(3)}} \left(u^{(1)} - u^{(2)}\right)\mu \quad = \quad 0 \quad \forall \mu \in H^{-\frac{1}{2}}(\Gamma^{(3)}) \tag{1.14}$$

Consider the Lagrangian

$$\begin{aligned} L(v^{(1)}, v^{(2)}, \mu) = & \frac{1}{2}\int_{\Omega^{(1)}} \nabla v^{(1)} \cdot \nabla v^{(1)} - \int_{\Omega^{(1)}} f^{(1)} v^{(1)} + \frac{1}{2}\int_{\Omega^{(2)}} \nabla v^{(2)} \cdot \nabla v^{(2)} \\ & - \int_{\Omega^{(2)}} f^{(2)} v^{(2)} - \int_{\Gamma^{(3)}} \left(v^{(1)} - v^{(2)}\right)\mu \end{aligned} \tag{1.15}$$

then we note that the saddle point $(u^{(1)}, u^{(2)}, \lambda)$ of L in $H^1_{0\partial\Omega^{(1)}\setminus\Gamma^{(3)}}(\Omega^{(1)}) \times H^1_{0\partial\Omega^{(2)}\setminus\Gamma^{(3)}}(\Omega^{(2)}) \times H^{-\frac{1}{2}}(\Gamma^{(3)})$ is precisely the point where the variational equations (1.11) and (1.14) are satisfied.

Discretization

Let us consider the discretization of the variational equation (1.2) using a finite element method. This process works for different elliptic partial differential equations and different finite element discretizations. From now on, we can consider the discussion as a more general work. Therefore, we start from any discretization of an elliptic PDE that will lead to a system of the following form

$$Kx = f \tag{1.16}$$

The global stiffness matrix of the discrete problem can be arranged to have the block structure showed in Equation (1.17). The subscript i denote the inner degrees of freedom of subdomains $\Omega^{(1)}$ and $\Omega^{(2)}$ and subscript b is used for the nodes on the interface $\Gamma^{(3)} = \partial\Omega^{(1)} \cap \Omega^{(2)} = \Omega^{(1)} \cap \partial\Omega^{(2)}$

$$\begin{bmatrix} K_{ii}^{(1)} & 0 & K_{ib}^{(1)} \\ 0 & K_{ii}^{(2)} & K_{ib}^{(2)} \\ K_{bi}^{(1)} & K_{bi}^{(2)} & K_{bb} \end{bmatrix} \begin{bmatrix} x_i^{(1)} \\ x_i^{(2)} \\ x_b \end{bmatrix} = \begin{bmatrix} f_i^{(1)} \\ f_i^{(2)} \\ f_b \end{bmatrix} \tag{1.17}$$

The formulation of each local matrix is made considering that every subdomain has its mesh and also that the nodes of the interface $\Gamma^{(3)}$ are shared by both subdomains, see Figure 1.2. So there are two interface blocks, one in each local matrix, noted with superscripts (1) and (2). The local stiffness matrices of the

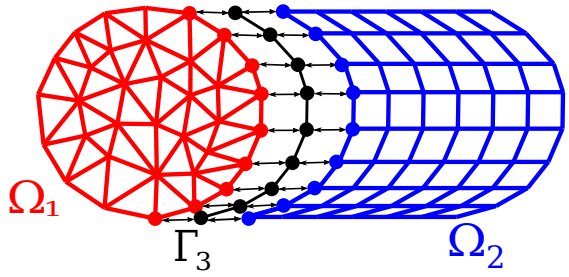

Figure 1.2 – Two subdomain divisions with duplicated nodes.

two subdomains are

$$K^{(1)} = \begin{bmatrix} K_{ii}^{(1)} & K_{ib}^{(1)} \\ K_{bi}^{(1)} & K_{bb}^{(1)} \end{bmatrix} \qquad K^{(2)} = \begin{bmatrix} K_{ii}^{(2)} & K_{ib}^{(2)} \\ K_{bi}^{(2)} & K_{bb}^{(2)} \end{bmatrix} \tag{1.18}$$

where $K_{bb}^{(1)} + K_{bb}^{(2)} = K_{bb}$.

The discretization of variational formulation of Equation (1.4) in subdomain $\Omega^{(s)}$ leads to the following system of equations

$$\begin{bmatrix} K_{ii}^{(s)} & K_{ib}^{(s)} \\ K_{bi}^{(s)} & K_{bb}^{(s)} \end{bmatrix} \begin{bmatrix} x_i^{(s)} \\ x_b^{(s)} \end{bmatrix} = \begin{bmatrix} f_i^{(s)} \\ f_b^{(s)} + h_b^{(s)} \end{bmatrix} \tag{1.19}$$

where $f_b^{(1)} + f_b^{(2)} = f_b$ and $h_b^{(s)}$ is the vector representing the discretization of the flux $\frac{\partial x^{(s)}}{\partial n^{(s)}}$ on $\Gamma^{(3)}$.

From this we have an explicit relation between the inner an the interface nodes

$$x_i^{(s)} = K_{ii}^{(s)^{-1}} f_i^{(s)} - K_{ii}^{(s)^{-1}} K_{ib}^{(s)} x_b^{(s)} \tag{1.20}$$

from (1.20) and (1.19) the relation between the trace and the flux of a vector satisfying the inner subset is derived

$$\begin{aligned} h_b^{(s)} &= K_{bi}^{(s)} x_i^{(s)} + K_{bb}^{(s)} x_b^{(s)} - f_b^{(s)} \\ &= K_{bi}^{(s)} (K_{ii}^{(s)^{-1}} f_i^{(s)} - K_{ii}^{(s)^{-1}} K_{ib}^{(s)} x_b^{(s)}) + K_{bb}^{(s)} x_b^{(s)} - f_b^{(s)} \\ &= (K_{bb}^{(s)} - K_{bi}^{(s)} K_{ii}^{(s)^{-1}} K_{ib}) x_b^{(s)} - (f_b^{(s)} - K_{bi}^{(s)} K_{ii}^{(s)^{-1}} f_i^{(s)}) \\ &= S_{bb}^{(s)} x_b^{(s)} - c_b^{(s)} \end{aligned} \tag{1.21}$$

In the last line, $S_{bb}^{(s)} := K_{bb}^{(s)} - K_{bi}^{(s)} {K_{ii}^{(s)}}^{-1} K_{ib}^{(s)}$ is the Schur complement matrix. It is the discretization of the Dirichlet to Neumann mapping that defines the bi-continuous one to one correspondence between the trace and the flux on the boundary (or interface in our case) of a field that satisfies the Laplace equation inside the subdomain. It is symmetric positive definite if the $K^{(s)}$ matrix is symmetric positive definite.

The discretization of continuity (1.6) and flux (1.7) are

$$x_b^{(1)} = x_b^{(2)} \tag{1.22}$$

$$h_b^{(1)} + h_b^{(2)} = 0 \tag{1.23}$$

The last condition also called equilibrium, combined with (1.19), gives the following interface equation

$$\begin{aligned} K_{bi}^{(1)} x_i^{(1)} + K_{bb}^{(1)} x_b - f_b^{(1)} + K_{bi}^{(2)} x_i^{(2)} + K_{bb}^{(2)} x_b - f_b^{(2)} = 0 \Leftrightarrow \\ K_{bi}^{(1)} x_i^{(1)} + K_{bi}^{(2)} x_i^{(2)} + (K_{bb}^{(1)} + K_{bb}^{(2)}) x_b = f_b^{(1)} + f_b^{(2)} \end{aligned} \tag{1.24}$$

Finally, for two vectors defined on subdomains $\Omega^{(1)}$ and $\Omega^{(2)}$ to be considered as the restrictions of the solution of the global discrete problem (1.17), they must meet

- the inner equations in each subdomain

$$\left\{ \begin{array}{rcl} K_{ii}^{(1)} x_i^{(1)} + K_{ib}^{(1)} x_b^{(1)} & = & f_i^{(1)} \\ K_{ii}^{(2)} x_i^{(2)} + K_{ib}^{(2)} x_b^{(2)} & = & f_i^{(2)} \end{array} \right. \tag{1.25}$$

- the interface equation

$$K_{bi}^{(1)} x_i^{(1)} + K_{bi}^{(2)} x_i^{(2)} + K_{bb}^{(1)} x_b^{(1)} + K_{bb}^{(2)} x_b^{(2)} = f_b^{(1)} + f_b^{(2)} \tag{1.26}$$

- the continuity across the interface $\Gamma^{(3)}$

$$x_b^{(1)} = x_b^{(2)} \tag{1.27}$$

If the continuity relation (1.27) is fulfilled and if we use the fact that $x_b^{(1)}$ and $x_b^{(2)}$ are both equal to the restriction of the global solution on $\Gamma^{(3)}$, then the inner equations (1.25) are the first two rows of (1.17). Also, the interface equations (1.26) are the third row; meaning that the methodology derived only from linear algebra is valid for any finite element discretization of an elliptic PDE.

The inner equations (1.25) are common solution vectors of local problems for any kind of boundary conditions on $\Gamma^{(3)}$. Equations (1.26) and (1.27) are the actual condensed interface problem. This comes from the fact that inner equations (1.25) allow to obtain $x_i^{(1)}$ and $x_i^{(2)}$ starting from $x_b^{(1)}$ and $x_b^{(2)}$.

1.1.2 FETI with one Lagrange Multiplier

The previous ideas are now formalized for the general case where we have $N_s > 2$ subdomains. Using any non-overlapping partition of the domain Ω into subdomains $\Omega^{(s)}$, $s = 1,\ldots,N_s$. We define the interface Γ

$$\Gamma = \bigcup_{1\leq s,q\leq N_s} (\partial\Omega^{(s)} \cap \partial\Omega^{(q)})$$

In the FETI method the discrete flux, noted λ, is the unknown vector defined in the nodes along the interface. The jump in the solutions of the local Neumann problems is the gradient of the condensed interface problem.

With the previous notation, the discretization of the local Neumann problems in some subdomain $\Omega^{(s)}$, can be written as

$$K^{(s)}x^{(s)} = f^{(s)} + t^{(s)^T} B^{(s)^T} \lambda \tag{1.28}$$

where $K^{(s)}$ are the local stiffness matrices; $f^{(s)}$ the right-hand side vectors; $t^{(s)} \in \mathcal{M}_{\#(\partial\Omega^{(s)})\times\#(\Omega^{(s)})}$ are trace operators which extracts boundary degrees of freedom from any subdomain $\Omega^{(s)}$. Finally, $B^{(s)} \in \mathcal{M}_{dim(\Gamma)\times dim(\partial\Omega^{(s)})}$ are discrete assembling matrices which connects pairwise degrees of freedom on the interface.

In a general case, the stiffness matrices $K^{(s)}$, the local solution vectors $x^{(s)}$ and

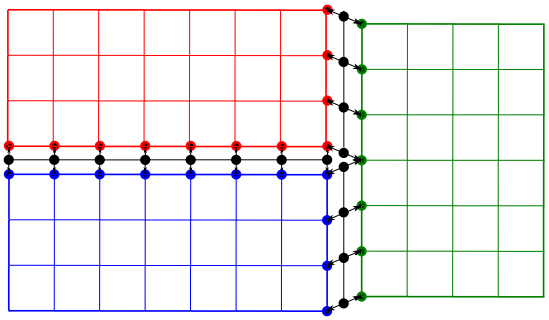

Figure 1.3 – Multiple interface node.

the local right-hand side vectors $f^{(s)}$ are defined as

$$K^{(s)} = \begin{bmatrix} K_{ii}^{(s)} & K_{ib}^{(s)} \\ K_{bi}^{(s)} & K_{bb}^{(s)} \end{bmatrix}$$

$$x^{(s)} = \begin{bmatrix} x_i^{(s)} \\ x_b^{(s)} \end{bmatrix}, f^{(s)} = \begin{bmatrix} f_i^{(s)} \\ f_b^{(s)} \end{bmatrix}$$

The i and b subscripts also denote the interior and interface nodes in the subdomain, respectively. So the trace operator $t^{(s)}$ applied to the solution $x^{(s)}$ is such that

$$x_b^{(s)} = t^{(s)} x^{(s)}$$

The discrete operators $B^{(s)}$ are the mapping of vectors defined in $\partial\Omega^{(s)}$ onto the complete interface. Applied to the solution in the local interface $x_b^{(s)}$ we can use them to write the continuity condition across the total interface, degree of freedom per degree of freedom

$$\sum_s B^{(s)} t^{(s)} x^{(s)} = 0 \tag{1.29}$$

the restriction of $B^{(s)}$ on the interface $\Gamma^{(ij)} = \partial\Omega^{(i)} \cap \partial\Omega^{(j)}$, noted $B^{(ij)}$, is defined as a signed boolean operator such that $B^{(ij)}$ and $B^{(ji)}$ have opposite signs, providing the continuity needed. Any node found in more than two subdomains, e.g., see Figure 1.3, will generate the same number of continuity conditions and flux as the number of interfaces that share it.

With the definition of $B^{(s)}$ and $t^{(s)}$, the solutions $x^{(s)}$ of (1.28) and (1.29) are the searched restrictions in every subdomain of the global discrete solution of (1.17). The definition of $B^{(s)}$ is such that ${t^{(s)}}^T {B^{(s)}}^T \lambda$ is zero for the inner nodes of $\Omega^{(s)}$. Also, in the interface we have that ${B^{(ij)}}^T \lambda + {B^{(ji)}}^T \lambda = 0$ thanks to the opposite sign. So, again, the assembly of local discrete Neumann equations (1.28) gives exactly the global discrete equation (1.16).

We define the gradient of the condensed interface problem as

$$g = \sum_s B^{(s)} t^{(s)} x^{(s)}$$

where $x^{(s)}$ is the solution of the local discrete Neumann problem (1.28). Continuity relation (1.29) will define the condensed interface problem for FETI.

"Floating" subdomains

For most subdomains, we face the common case of finding that $\partial\Omega^{(s)} \cap \partial\Omega$ is void. This means that the Dirichlet condition of the problem is not on $\Omega^{(s)}$, so the local discrete Neumann equations (1.28) are ill-posed. If $K^{(s)}$ comes from the Laplace equation, its kernel is a constant field in the subdomain; if it comes from three-dimensional linear elasticity, then the kernel is the subspace of rigid body motions (of dimension 6 in the case of subdomains simply connected).

The pseudo-inverse ${K^{(s)}}^+$ is now needed, and the Cholesky factorization with partial pivoting is used on the matrix $K^{(s)}$ to achieve this. This choice is also justified by the fact that it allows computing a generator of the kernel $R^{(s)}$ and a factorization of a maximal full rank sub-block. Given the pseudo-inverse ${K^{(s)}}^+$ and the kernel generator $R^{(s)}$, the solution $x^{(s)}$ of the discrete system of equation (1.28) can be written as a particular solution plus an undefined element of the kernel of $K^{(s)}$

$$x^{(s)} = {K^{(s)}}^+ (f^{(s)} + {t^{(s)}}^T {B^{(s)}}^T \lambda) + R^{(s)} \alpha^{(s)} \tag{1.30}$$

From equation (1.28), we see that the right-hand side belongs to the range of the matrix $K^{(s)}$, and so it is orthogonal to the kernel. This orthogonality constraint can be written

$${R^{(s)}}^T \left(f^{(s)} + {t^{(s)}}^T {B^{(s)}}^T \lambda \right) = 0 \tag{1.31}$$

This last equation is the admissibility condition for the forces in a floating subdomain. Its interpretation is that fields belonging to the kernel must have zero energy.

Condensed interface problem

Replacing $x^{(s)}$ from equation (1.30) in the continuity condition (1.29) we have

$$\sum_s B^{(s)} t^{(s)} K^{(s)^+} t^{(s)^T} B^{(s)^T} \lambda + \sum_s B^{(s)} t^{(s)} R^{(s)} \alpha^{(s)} = -\sum_s B^{(s)} t^{(s)} K^{(s)^+} f^{(s)} \tag{1.32}$$

To build the condensed interface problem, we use equations (1.32) and (1.31); leading to the system to be satisfied by both λ and the vector α of coefficients of the kernel components

$$\begin{bmatrix} F & G \\ G^T & 0 \end{bmatrix} \begin{bmatrix} \lambda \\ \alpha \end{bmatrix} = \begin{bmatrix} d \\ c \end{bmatrix} \tag{1.33}$$

Where:

- $F = \sum_s B^{(s)} t^{(s)} K^{(s)^+} t^{(s)^T} B^{(s)^T} = \sum_s B^{(s)} S_{bb}^{(s)^+} B^{(s)^T}$ dual Schur complement matrix
- $G\alpha = \sum_s B^{(s)} t^{(s)} R^{(s)} \alpha^{(s)}$, jump of zero energy fields defined by $\alpha^{(s)}$ in $\Omega^{(s)}$
- $G^T \lambda = \left(\ldots, B^{(s)} t^{(s)} R^{(s)}, \ldots \right)^T \lambda$
- $d = -\sum_s B^{(s)} t^{(s)} K^{(s)^+} f^{(s)}$
- $c = \left(\ldots, -f^{(s)^T} R^{(s)}, \ldots \right)^T$

The condensed interface system (1.33) is a hybrid system. Its solution λ satisfies the following orthogonality condition

$$\mu^T F \lambda = \mu^T d, \ \forall \mu / G^T \mu = 0 \tag{1.34}$$

Now consider λ_0, for example

$$\lambda_0 = AG(G^T AG)^{-1} c$$

where A is a symmetric positive definite matrix. This matrix is usually taken as the identity, but it can also be defined as the preconditioner (we will describe it later) or some scaling matrix. For details, see [62].

Then this λ_0 satisfies the admissibility constraint of Equation (1.31) and $G^T(\lambda - \lambda_0) = 0$. So, if P is any projector in the kernel of G^T, then from Equation (1.34) we have that λ is the solution of the following projected problem

$$P^T F P(\lambda - \lambda_0) = P^T(d - F\lambda_0) \tag{1.35}$$

The FETI method solves iteratively via a conjugate gradient algorithm the previous projected condensed interface problem (1.35), using the orthogonal projector in the kernel of G^T.

Interpretation of the projector P

The orthogonal projection in the kernel of G^T can be written algebraically

$$P = I - AG\left(G^T AG\right)^{-1} G^T \tag{1.36}$$

To compute the projection of a given vector g we mainly solve the problem

$$\left(G^T AG\right)\alpha = -G^T g \tag{1.37}$$

which is a global coarse grid problem whose unknowns are the coefficients of zero energy components of the solutions in the floating subdomains.

Now for a given approximation λ^p of the flux on the interface, the residual of the condensed interface problem is

$$g^p = \sum_s B^{(s)} S_{bb}^{(s)^+} B^{(s)^T} \lambda^p + \sum_s B^{(s)} t^{(s)} K^{(s)^+} f^{(s)} = \sum_s B^{(s)} t^{(s)} x^{(s)^{p+}}$$

where $x^{(s)^{p+}}$ is the solution of the local Neumann problems, computed using the pseudo-inverse matrices

$$x^{(s)^{p+}} = K^{(s)^+}\left(f^{(s)} + t^{(s)^T} B^{(s)^T} \lambda^p\right)$$

so the gradient is equal to the jump of these particular solutions. From equations (1.36) and (1.37), the projected gradient Pg^p is

$$Pg^p = g^p + AG\alpha^p = \sum_s B^{(s)} t^{(s)} x^{(s)p+} + \sum_s B^{(s)} t^{(s)} R^{(s)} \alpha^{(s)p}$$

the projected gradient Pg_p is equal to the jump of the particular local solutions of Neumann problems $x^{(s)p+}$ plus the term of zero energy fields with coefficients $\alpha^{(s)p}$

$$x^{(s)p} = x^{(s)p+} + R^{(s)} \alpha^{(s)p}$$

The definition of the constraint (1.31), associated with the orthogonal projector (1.37), entails that the zero energy components of the jump of the local solution fields $x^{(s)p}$ are minimal in the sense that this jump is orthogonal to all the traces of zero energy fields

$$G^T Pg^p = 0 \Leftrightarrow \left(B^{(s)} t^{(s)} R^{(s)} \right)^T Pg^p = 0 \qquad \forall s$$

The projected gradient Pg^p is obtained by computing the coefficients α^p of optimal local zero energy fields. For the linear elasticity problem, the zero energy fields are the rigid body motions. The underlying process is a kind of coarse grid smoothing of approximate solutions that ensures a convergence rate for the overall FETI process asymptotically independent upon the number of subdomains [27]. Hence, the FETI method with floating subdomains is a kind of two-level solver that is numerically scalable.

1.1.3 Local preconditioner

The coarse grid smoothing performed by the zero energy fields projector P gives a convergence rate independent upon the number of subdomains, but this is not enough to have a convergence rate that is also independent upon the mesh size. Is necessary the use of a preconditioner, which for the FETI method is one of the "Dirichlet" type.

Consider $t^{(s)}$ the trace or restriction operator on the local interface of subdomain $\Omega^{(s)}$. Then the contribution of this subdomain to the condensed interface

operator, defined in (1.32), is

$$B^{(s)} t^{(s)} K^{(s)^+} t^{(s)^T} B^{(s)^T}$$

and it just depends on the restriction on the interface $\Gamma^{(s)} = \partial\Omega^{(s)}$ of the pseudo-inverse of $K^{(s)}$, meaning

$$t^{(s)} K^{(s)^+} t^{(s)^T} = (K^{(s)}_{bb} - K^{(s)}_{bi} K^{(s)^{-1}}_{ii} K^{(s)}_{ib})^+ =: S^{(s)^+}_{bb}$$

As this matrix is the pseudo-inverse of the Schur complement matrix on the interface $\Gamma^{(s)}$, a preconditioner based on local contributions for FETI is

$$D^{-1} = \sum_s B^{(s)} S^{(s)}_{bb} B^{(s)^T} \tag{1.38}$$

where again $S^{(s)}_{bb}$ is the Schur complement. This preconditioner tries to approximate the global inverse of local sums by the sum of local inverses, meaning

$$\left(\sum_s B^{(s)} S^{(s)^+}_{bb} B^{(s)^T} \right)^+ \simeq \sum_s B^{(s)} S^{(s)}_{bb} B^{(s)^T}$$

The computation of this preconditioner applied to an interface vector w is done by solving the following local problems with Dirichlet boundary conditions on the interface $\Gamma^{(s)}$. They are imposed by the assembled local vector $t^{(s)^T} B^{(s)^T} w$

$$\begin{bmatrix} K^{(s)}_{ii} & K^{(s)}_{ib} \\ 0 & I \end{bmatrix} \begin{bmatrix} \tilde{w}^{(s)}_i \\ \tilde{w}^{(s)}_b \end{bmatrix} = t^{(s)^T} B^{(s)^T} w = \begin{bmatrix} 0 \\ w^{(s)}_b \end{bmatrix}$$

whose solution is

$$\tilde{w}^{(s)}_i = -K^{(s)^{-1}}_{ii} K^{(s)}_{ib} w^{(s)}_b$$

then, multiplying by the stiffness matrix we obtain

$$\begin{bmatrix} K^{(s)}_{ii} & K^{(s)}_{ib} \\ K^{(s)}_{bi} & K^{(s)}_{bb} \end{bmatrix} \begin{bmatrix} \tilde{w}^{(s)}_i \\ \tilde{w}^{(s)}_b \end{bmatrix} = \begin{bmatrix} 0 \\ (K^{(s)}_{bb} - K^{(s)}_{bi} K^{(s)^{-1}}_{ii} K^{(s)}_{ib}) \tilde{w}^{(s)}_b \end{bmatrix} = \begin{bmatrix} 0 \\ S^{(s)}_{bb} w^{(s)}_b \end{bmatrix}$$

With this preconditioner, it has been proved [52],[33] that the FETI method is asymptotically independent of the mesh size. With a condition number for the projected condensed operator bounded by

$$C(1+\log(\frac{H}{h}))^2$$

where h is the mesh size, H is the characteristic subdomain size and C is a constant that do not depend on any of the first parameters. Meaning, that when decreasing H the number of subdomains increases and when h goes to zero the finite element mesh is refined (more elements).

So, with both local and global preconditioners, the number of iterations does not depend anymore neither on the number of subdomains nor the mesh size. A second preconditioner, not mathematically optimal, can be introduced as an alternative to the Dirichlet one; the so-called "Lumped" preconditioner can be defined as

$$L^{-1} = \sum_s B^{(s)} K_{bb}^{(s)} B^{(s)^T} \tag{1.39}$$

where $K_{bb}^{(s)}$ is the finite element discretization matrix on the interface nodes. This preconditioner works as an approximation of the local Schur complements, with a much more economical implementation because it does not require any additional storage and involves only matrix-vector products of sizes equal to the subdomain interfaces.

Both preconditioners were generalized to treat heterogeneous problems [61] by just redefining the Boolean operator $B^{(s)}$ to a more general one

$$\tilde{B}^{(s)} := \beta^{(s)} B^{(s)}$$

such that $\sum_s \tilde{B}^{(s)} \tilde{B}^{(s)^T} = I$, with $\beta^{(s)}$ a diagonal scaling matrix, usually based on the diagonal coefficients of the local stiffness matrix on the interface, the so-called super-lumped scaling. This scaling is a consistent combination of the interface reaction forces from the Dirichlet problem in each subdomain (more details of this classic results are found in the next chapter). With this new scaled

assembling operators, the preconditioners are written as

$$D^{-1} = \sum_s \tilde{B}^{(s)} S_{bb}^{(s)} \tilde{B}^{(s)^T}$$

$$L^{-1} = \sum_s \tilde{B}^{(s)} K_{bb}^{(s)} \tilde{B}^{(s)^T}$$

The behavior of both preconditioners can be seen in [33], [61] and will be recalled in following sections.

1.1.4 FETI Algorithms

Using the definitions in (1.33) and the projector in (1.36), the CG algorithm to solve the FETI problem (1.35) can be summarized in Algorithm 5.

Algorithm 5 FETI Preconditioned Conjugate Projected Gradient with full reconjugation

1: **Initialization**
2: $\lambda_0 = AG[G^T AG]^{-1} c$
3: $g_0 = (F\lambda_0 - d)$
4: $w_0 = PD^{-1}P^T g_0$
5: **loop** $p = 0, 1, 2, \ldots$ until convergence
6: $\quad \rho_p = -\dfrac{(w_p, g_p)}{(w_p, Fw_p)}$
7: $\quad \lambda_{p+1} = \lambda_p + \rho_p w_p$
8: $\quad g_{p+1} = g_p + \rho_p F w_p$
9: $\quad w_{p+1} = PD^{-1}P^T g_{p+1}$
10: $\quad$ **for** $i = 0$ to p **do**
11: $\quad\quad \gamma_i = -\dfrac{(w_i, Fw_{p+1})}{(w_i, Fw_i)}$
12: $\quad\quad w_{p+1} = w_{p+1} + \gamma_i w_i$
13: $\quad\quad Fw_{p+1} = Fw_{p+1} + \gamma_i F w_i$
14: $\quad$ **end for**
15: **end loop**

As we can see, a full reconjugation, instead of the classical CG update without any, is also used (see the line (10) of the algorithm). This orthogonalization is needed because in finite precision arithmetic, the theoretical orthogonal proper-

ties of the CG method are lost. In particular, in the context of the FETI methods, the multiplication by the operator is not entirely accurate, making this part also crucial for good convergence rate [66].

Algorithm 6 FETI-1LM unsymmetric

Initialization
$\lambda_0 = AG[G^T AG]^{-1}(-R^T c)$
$g_0 = PD^{-1}P(F\lambda_0 - d)$
$w_0 = g_0$
$Fw_0 = PD^{-1}PFw_0$
loop ORTHODIR Iteration from $p = 0, 1, \ldots$ until convergence
    $\rho_p = -\dfrac{(Fw_p)^T g_p}{(Fw_p)^T (Fw_p)}$
    $\lambda_{p+1} = \lambda_p + \rho_p w_p$
    $g_{p+1} = g_p + \rho_p F w_p$
    loop Construction of the $p+1$ vector of the base $F^T F$-orthonormal
        $w_{p+1} = g_{p+1}$
        $Fw_{p+1} = PD^{-1}PFw_{p+1}$
        for $i = 0$ to p **do**
            $\gamma_i = -\dfrac{(Fw_i)^T (Fw_{p+1})}{(Fw_i)^T (Fw_i)}$
            $w_{p+1} = w_{p+1} + \gamma_i w_i$
            $Fw_{p+1} = Fw_{p+1} + \gamma_i Fw_i$
        end for
    end loop
end loop

Unsymmetric FETI algorithm

The previous method and algorithm only work when the matrix F is symmetric positive definite. For a more general case when the FETI operator F is no longer symmetric nor positive definite, the CG algorithm will not be appropriate (this is the case of the next two FETI method described in this chapter). We use instead, the ORTHODIR method with left preconditioner for unsymmetric matrices, mainly for its simple implementation and good convergence properties, as is equivalent to the GMRES algorithm [49, Chapter 12]. One of the main

theoretical differences between the two is the storage of previously computed directions which in both cases is needed to perform the full reorthogonalization that allows keeping the orthogonality between search directions and therefore a good convergence ratio.

The full description is given in Algorithm 6.

1.2 FETI with two Lagrange multipliers

1.2.1 FETI-2LM

In this section, we present the second FETI method that is used as a basis to build the new Hybrid-FETI. Originally introduced in [31] as a solver for acoustic problems, then extended in [67] and [68] as a robust solver for more general discretizations. The basic idea of this method is to impose Robin boundary conditions on the interface to "glue" the local approximations for them to be the restrictions of the global solution. The same way as in FETI, to have a method useful for different approximations of elliptic PDE, it will be derived from linear algebra.

To understand how this method works we start with a simple partition of the total domain Ω into two subdomains, as in Figure 1.1. From this splitting and some linear elliptic PDE problem defined in the global domain Ω we use any finite elements discretization coming from these equations. We use subscript i for internal nodes, b for the interface ones, with the interface defined as $\Gamma^{(3)} = \partial\Omega^{(1)} \cap \Omega^{(2)}$. From a finite element discretization, let us consider the contributions of each subdomain $\Omega^{(s)}$, $s = 1, 2$ to the matrix and right-hand side

$$K^{(s)} = \begin{bmatrix} K_{ii}^{(s)} & K_{ib}^{(s)} \\ K_{bi}^{(s)} & K_{bb}^{(s)} \end{bmatrix}, \qquad f^{(s)} = \begin{bmatrix} f_i^{(s)} \\ f_b^{(s)} \end{bmatrix}$$

where $K_{bb}^{(s)}$ represents the interaction matrices between the nodes on the interface and the interior of $\Omega^{(s)}$, independent of each other. The same happens for $f_b^{(s)}$. The global block equation system comes from assembling both contributions,

giving

$$\begin{bmatrix} K_{ii}^{(1)} & 0 & K_{ib}^{(1)} \\ 0 & K_{ii}^{(2)} & K_{ib}^{(2)} \\ K_{bi}^{(1)} & K_{bi}^{(2)} & K_{bb} \end{bmatrix} \begin{bmatrix} x_i^{(1)} \\ x_i^{(2)} \\ x_b \end{bmatrix} = \begin{bmatrix} f_i^{(1)} \\ f_i^{(2)} \\ f_b \end{bmatrix} \tag{1.40}$$

with $K_{bb} = K_{bb}^{(1)} + K_{bb}^{(2)}$ and $f_b = f_b^{(1)} + f_b^{(2)}$. Taking the second line to find $x_i^{(2)}$ and replacing it in the first and third lines, the system that remains can be written in the following way

$$\begin{bmatrix} K_{ii}^{(1)} & K_{ib}^{(1)} \\ K_{bi}^{(1)} & K_{bb}^{(1)} + (K_{bb}^{(2)} - K_{bi}^{(2)} K_{ii}^{(2)^{-1}} K_{ib}^{(2)}) \end{bmatrix} \begin{bmatrix} x_i^{(1)} \\ x_b \end{bmatrix} = \begin{bmatrix} f_i^{(1)} \\ f_b^{(1)} + (f_b^{(2)} - K_{bi}^{(2)} K_{ii}^{(2)^{-1}} f_i^{(2)}) \end{bmatrix}$$

The same treatment can be done, but eliminating the unknowns of subdomain 1. And so, for both subdomains we have

$$\begin{bmatrix} K_{ii}^{(s)} & K_{ib}^{(s)} \\ K_{bi}^{(s)} & K_{bb}^{(s)} + S_{bb}^{(q)} \end{bmatrix} \begin{bmatrix} x_i^{(s)} \\ x_b \end{bmatrix} = \begin{bmatrix} f_i^{(s)} \\ f_b^{(s)} + c_b^{(q)} \end{bmatrix} \tag{1.41}$$

with $S_{bb}^{(q)} = K_{bb}^{(q)} - K_{bi}^{(q)} K_{ii}^{(q)^{-1}} K_{ib}^{(q)}$ the Schur complement and $c_b^{(q)} := f_b^{(q)} - K_{bi}^{(q)} K_{ii}^{(q)^{-1}} f_i^{(q)}$ the condensed right-hand side on the subdomain $\Omega^{(q)}$, opposite to $\Omega^{(s)}$.

The previous equations show that the restriction of the global solution is, in each subdomain, a local solution of a problem with generalized Robin boundary condition on the interface. The operator of the generalized Robin condition is the Dirichlet to Neumann operator of the rest of the domain. In fact, the Dirichlet to Neumann operator describes exactly the behavior of the boundary of the outer domain. Enforcing a local generalized Robin boundary condition using the Dirichlet to Neumann operator of the rest of the domain makes the interface to behave locally exactly like the rest of the domain forces it to do. Then, the local problem formulation takes into account the coupling between the subdomain and the rest of the domain.

Local generalized Robin boundary conditions enforced on both sides of the same interface should be set up independently. This leads to introduce two

independent interface variables, one for each side.

$$\begin{bmatrix} K_{ii}^{(s)} & K_{ib}^{(s)} \\ K_{bi}^{(s)} & K_{bb}^{(s)} + A_b^{(s)} \end{bmatrix} \begin{bmatrix} x_i^{(s)} \\ x_b^{(s)} \end{bmatrix} = \begin{bmatrix} f_i^{(s)} \\ f_b^{(s)} + \lambda^{(s)} \end{bmatrix} \tag{1.42}$$

with $A_b^{(s)}$ to be defined. From these local equations, inner d.o.f can be eliminated to get the equivalent local condensed problem

$$\begin{aligned} (K_{bb}^{(s)} - K_{bi}^{(s)} K_{ii}^{(s)^{-1}} K_{ib}^{(s)} + A_b^{(s)}) + x_b^{(s)} &= f_b^{(s)} - K_{bi}^{(s)} K_{ii}^{(s)^{-1}} f_i^{(s)} + \lambda^{(s)} \\ (S_{bb}^{(s)} + A_b^{(s)}) x_b^{(s)} &= c_b^{(s)} + \lambda^{(s)} \end{aligned} \tag{1.43}$$

And the explicit relation between the trace of the solution and the Lagrange multiplier is

$$x_b^{(s)} = (S_{bb}^{(s)} + A_b^{(s)})^{-1} \lambda^{(s)} + (S_{bb}^{(s)} + A_b^{(s)})^{-1} c_b^{(s)} \tag{1.44}$$

The boundary conditions on both sides are not consistent, so the two interface conditions (1.26) and (1.27) must be enforced to make the solutions of the local problems (1.42) be the restrictions of the solution of the global problem (1.17). Thanks to the second line of local equation (1.42), the interface equilibrium condition gives

$$A_b^{(1)} x_b^{(1)} + A_b^{(2)} x_b^{(2)} = \lambda^{(1)} + \lambda^{(2)} \tag{1.45}$$

The residual of the condensed interface problem is then

$$\begin{aligned} x_b^{(1)} - x_b^{(2)} &= 0 \\ \lambda^{(1)} + \lambda^{(2)} - A_b^{(1)} x_b^{(1)} - A_b^{(2)} x_b^{(2)} &= 0 \end{aligned} \tag{1.46}$$

where $x_b^{(s)}$ is a function of $\lambda^{(s)}$ according to Equation (1.44). These two equations, combined with the local problem (1.42) are equivalent to the global block problem (1.40). With this formulation, equations (1.46) have a mixed form, to avoid this, a new combination of both equations is proposed, leading to a new

condensed interface problem

$$\begin{aligned} \lambda^{(1)} + \lambda^{(2)} - (A_b^{(1)} + A_b^{(2)})x_b^{(2)} = 0 \\ \lambda^{(2)} + \lambda^{(1)} - (A_b^{(2)} + A_b^{(1)})x_b^{(1)} = 0 \end{aligned} \tag{1.47}$$

These two forms are equivalent if the sum of the augmentation matrices $A_b^{(1)}+A_b^{(2)}$ is non singular. This is verified if the matrix $A_b^{(s)}$ is the Schur complement of the rest of the domain and easily satisfied when $A_b^{(s)}$ is any consistent approximation of the Dirichlet to Neumann operator of the rest of the domain.

Finally we compute $\lambda^{(1)}$ and $\lambda^{(2)}$. To do so, we have to obtain $x_b^{(s)}$ from (1.44) and use it in (1.47), then the condensed interface problem can be explicitly defined by

$$\begin{aligned} \begin{bmatrix} I & I - (A_b^{(1)} + A_b^{(2)})(S_{bb}^{(2)} + A_b^{(2)})^{-1} \\ I - (A_b^{(2)} + A_b^{(1)})(S_{bb}^{(1)} + A_b^{(1)})^{-1} & I \end{bmatrix} \begin{bmatrix} \lambda^{(1)} \\ \lambda^{(2)} \end{bmatrix} \\ = \begin{bmatrix} (A_b^{(1)} + A_b^{(2)})(S_{bb}^{(2)} + A_b^{(2)})^{-1} c_b^{(2)} \\ (A_b^{(2)} + A_b^{(1)})(S_{bb}^{(1)} + A_b^{(1)})^{-1} c_b^{(1)} \end{bmatrix} \end{aligned} \tag{1.48}$$

This method is called FETI-2LM, like 2-Lagrange multiplier FETI method.

1.2.2 Arbitrary mesh partition

We show the extension of the previous method to the case of an arbitrary mesh with a total number of subdomains $N_s > 2$. The idea is to take the two subdomain case and apply it to the different local interfaces created between neighbor subdomains.

To define this, we start by dividing the total interface boundary Γ into interface edges Γ^j considering that an interface edge is a collection of connecting interface nodes. The total interface can be written as

$$\Gamma = \bigcup_{1 \le s,q \le N_s} (\partial\Omega^{(s)} \cap \partial\Omega^{(q)})$$

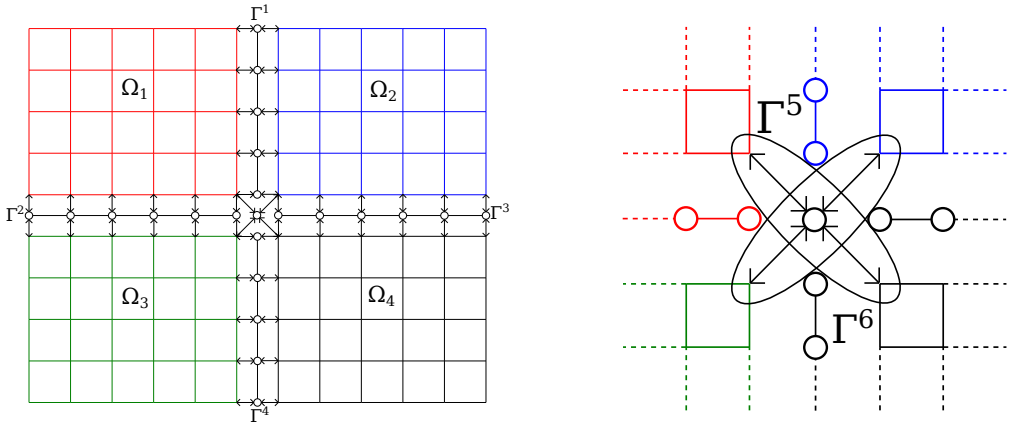

Figure 1.4 – Γ^j division example and crosspoint detail.

and we can define more precisely Γ^j as

$$\Gamma^j := \Gamma^{(sq)} = \partial\Omega^{(s)} \cap \partial\Omega^{(q)}, \qquad \forall s,q : \partial\Omega^{(s)} \cap \partial\Omega^{(q)} \neq \emptyset$$

The crosspoints are nodes where more than two subdomains are connected, see Figure 1.4 for an example of 4 subdomains and one crosspoint. We can see here, that more than four edges are defined in the crosspoint, as diagonal subdomains are neighbors. For each interface edge Γ^j and for each subdomain $\Omega^{(s)}$ that intersects it, let us define a boolean matrix $B^{(s)}_{\Gamma^j} \in \mathcal{M}_{dim(\Gamma^j)\times dim(\Omega^{(s)})}(\mathbb{R})$ by

$$B^{(s)}_{\Gamma^j} v^{(s)} = v^{(s)}_b|_{\Gamma^j}$$

where the vector $v^{(s)}$ defined in all the nodes of subdomain $\Omega^{(s)}$ can be written by separating the nodes of the complete local interface from the interior ones as $v^{(s)} = \begin{bmatrix} v^{(s)}_i \\ v^{(s)}_b \end{bmatrix}$.

Remark: This definition of interface edges ($\Gamma^j := \Gamma^{(sq)}$) can produce redundancies in the formulation of the problem. In practice these redundancies do not produce any negative impact in the FETI methods, on the contrary, for some cases such as the preconditioning in FETI-1LM they play a vital role, as we will see later in the text.

With the previous definitions we can rewrite the local problem (1.42) and the continuity conditions (1.46) for the general case

$$\begin{aligned}\left(K^{(s)}+\sum_{\Gamma^j\subseteq\Omega^{(s)}} B_{\Gamma^j}^{(s)^T} A_{\Gamma^j}^{(s)} B_{\Gamma^j}^{(s)}\right)x^{(s)} &= f^{(s)} - \sum_{\Gamma^l\subseteq\Omega^{(s)}} B_{\Gamma^l}^{(s)^T}\lambda_{\Gamma^l}^{(s)}, \qquad s=1,\dots,N_s\\ B_{\Gamma^j}^{(s)}x^{(s)} - x_b|_{\Gamma^j} &= 0 \qquad \forall\Gamma^j,\quad \Omega^{(s)}\supseteq\Gamma^j\\ \sum_{\Omega^{(q)}\supseteq\Gamma^j}\left(\lambda_{\Gamma^j}^{(q)} - A_{\Gamma^j}^{(q)}x_b|_{\Gamma^j}\right) &= 0 \qquad \forall\Gamma^j\end{aligned}$$

where, $A_{\Gamma^j}^{(s)}$ is the augmentation matrix in subdomain $\Omega^{(s)}$ for the interface Γ^j. The same way $\lambda_{\Gamma^j}^{(s)}$ is the Lagrange multiplier in the interface edge Γ^j on the side of the subdomain $\Omega^{(s)}$. The vector $x_b|_{\Gamma^j}$ is the restriction of the global unknown x to Γ^j. Let us also define, for every $\Omega^{(s)}$ the total augmentation matrix

$$A^{(s)} := \sum_{\Gamma^j\subseteq\Omega^{(s)}} B_{\Gamma^j}^{(s)^T} A_{\Gamma^j}^{(s)} B_{\Gamma^j}^{(s)}$$

Finally, we can do the condensation of previous equations to obtain the interface problem

$$\begin{aligned}\sum_{\Omega^{(q)}\supseteq\Gamma^j}&\left(\lambda_{\Gamma^j}^{(q)} - A_{\Gamma^j}^{(q)}B_{\Gamma^j}^{(s)}\left(K^{(s)}+A^{(s)}\right)^{-1}\sum_{\Gamma^l\subseteq\Omega^{(s)}} B_{\Gamma^l}^{(s)^T}\lambda_{\Gamma^l}^{(s)}\right)\\ &= -\sum_{\Omega^{(q)}\supseteq\Gamma^j} A_{\Gamma^j}^{(q)}B_{\Gamma^j}^{(s)}\left(K^{(s)}+A^{(s)}\right)^{-1} f^{(s)}, \qquad \forall\Gamma^j,\forall\Omega^{(s)}\supseteq\Gamma^j\end{aligned} \tag{1.49}$$

This problem can also be written as $F\lambda = d$ where F is, in general, a non-symmetric matrix, defined in the whole interface, so again, the ORTHODIR method with full reorthogonalization is used to solve the problem. One of the differences with FETI-1LM is that there are no efficient local preconditioners for this method, so for the cases where convergence is expected in both methods, the 2LM may be slower. On the contrary, there are cases (e.g., anisotropic materials, heterogeneities in the interface, etc.) when bad numerical features located across the interface will not allow the convergence of FETI-1LM. In this case, the use

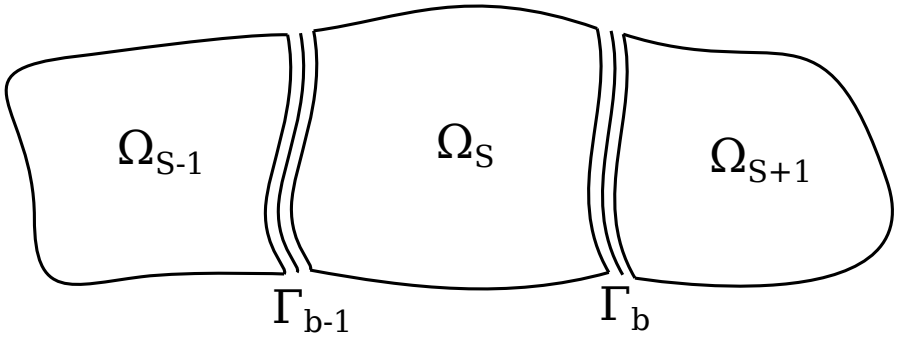

Figure 1.5 – One way splitting.

of two independent Lagrange multipliers appears as an alternative where the regularization terms can assure the convergence, making the FETI-2LM a more robust method.

1.2.3 Optimal Interface Boundary Conditions

As seen in the definition of the FETI-2LM operator, the results of this method depend on the determination of the augmentation matrix $A^{(s)}$. Different works have been developed to achieve the best numerical choice. Next, we will show some of them.

From a theoretical point of view, in the case of two subdomains, the optimal augmentation matrices are defined as

$$\begin{aligned} A_b^{(1)} &:= S_{bb}^{(2)} \\ A_b^{(2)} &:= S_{bb}^{(1)} \end{aligned} \tag{1.50}$$

This choice is proved by the following theorem.

Theorem 1.1. *In a case of a two-domain splitting, the simple (Jacobi) iterative algorithm for 2-Lagrange multiplier with augmented term equal to the complete outer Schur complement defined as in Equation (1.50) converges in one iteration at most.*

Proof. From the definition of $A_b^{(s)} := S_{bb}^{(q)}$, $s = 1, 2$, $q = 1, 2$, $s \neq q$, the matrix that defines the method (1.48) is equal to the identity. □

In a more general case of a one-way division in N subdomains with no crosspoints, the optimal augmentation can be proved to be the Schur complement of the rest of the domain. Consider a one-way splitting as in Figure 1.5 and denote the nodes in the interface with the subscript $b-1$ for the left interface and b for the right. Then the local contributions can be written as

$$K^{(s)} = \begin{bmatrix} K_{ii}^{(s)} & K_{ib-1}^{(s)} & K_{ib}^{(s)} \\ K_{b-1i}^{(s)} & K_{b-1b-1}^{(s)} & 0 \\ K_{bi}^{(s)} & 0 & K_{bb}^{(s)} \end{bmatrix} \quad , \quad f^{(s)} = \begin{bmatrix} f_i^{(s)} \\ f_{b-1}^{(s)} \\ f_b^{(s)} \end{bmatrix}$$

If we eliminate the inner nodes, the contributions to the condensed matrix and right-hand side of subdomain $\Omega^{(s)}$ are

$$\begin{bmatrix} S_{b-1b-1}^{(s)} & S_{b-1b}^{(s)} \\ S_{bb-1}^{(s)} & S_{bb}^{(s)} \end{bmatrix} = \begin{bmatrix} K_{b-1b-1}^{(s)} - K_{b-1i}^{(s)} K_{ii}^{(s)^{-1}} K_{ib-1}^{(s)} & -K_{b-1i}^{(s)} K_{ii}^{(s)^{-1}} K_{ib}^{(s)} \\ -K_{bi}^{(s)} K_{ii}^{(s)^{-1}} K_{ib-1}^{(s)} & K_{bb}^{(s)} - K_{bi}^{(s)} K_{ii}^{(s)^{-1}} K_{ib}^{(s)} \end{bmatrix}$$

$$\begin{bmatrix} c_{b-1}^{(s)} \\ c_b^{(s)} \end{bmatrix} = \begin{bmatrix} f_{b-1}^{(s)} - K_{b-1i}^{(s)} K_{ii}^{(s)^{-1}} f_i^{(s)} \\ f_b^{(s)} - K_{bi}^{(s)} K_{ii}^{(s)^{-1}} f_i^{(s)} \end{bmatrix}$$

After assembling the local contributions, the global condensed problem is a block tridiagonal system

$$\begin{bmatrix} \cdots & \cdots & 0 & 0 \\ S_{b-1b-2}^{(s-1)} & S_{b-1b-1}^{(s-1)} + S_{b-1b-1}^{(s)} & S_{b-1b}^{(s)} & 0 \\ 0 & S_{bb-1}^{(s)} & S_{bb}^{(s)} + S_{bb}^{(s+1)} & S_{bb+1}^{(s+1)} \\ 0 & 0 & \cdots & \cdots \end{bmatrix} \quad , \quad \begin{bmatrix} \cdots \\ c_{b-1}^{(s-1)} + c_{b-1}^{(s)} \\ c_b^{(s)} + c_b^{(s+1)} \\ \cdots \end{bmatrix}$$

If this system is factorized by successive condensations from both sides up to the subdomain $\Omega^{(s)}$, then the following condensed problem is obtained in subdomain $\Omega^{(s)}$

$$\begin{bmatrix} S_{b-1b-1}^{(-)} + S_{b-1b-1}^{(s)} & S_{b-1b}^{(s)} \\ S_{bb-1}^{(s)} & S_{bb}^{(s)} + S_{bb}^{(+)} \end{bmatrix} \begin{bmatrix} x_{b-1}^{(s)} \\ x_b^{(s)} \end{bmatrix} = \begin{bmatrix} c_{b-1}^{(-)} + c_{b-1}^{(s)} \\ c_b^{(s)} + c_b^{(+)} \end{bmatrix}$$

where the terms with (+) and (−) superscript are recursively denoted by

$$\begin{aligned}
S^{(-)}_{b-1b-1} &= S^{(s)}_{b-1b-1} - S^{(s)}_{b-1b}\left[S^{(s)}_{bb}+S^{(+)}_{bb}\right]^{-1} S^{(s)}_{bb-1}\\
S^{(+)}_{bb} &= S^{(s)}_{bb} - S^{(s)}_{bb-1}\left[S^{(-)}_{b-1b-1}+S^{(s)}_{b-1b-1}\right]^{-1} S^{(s)}_{b-1b}\\
c^{(-)}_{b-1} &= c^{(s)}_{b} - S^{(s)}_{bb-1}\left[S^{(-)}_{b-1b-1}+S^{(s)}_{b-1b-1}\right]^{-1}\left[c^{(-)}_{b-1}+c^{(s)}_{b-1}\right]\\
c^{(+)}_{b} &= c^{(s)}_{b-1} - S^{(s)}_{b-1b}\left[S^{(s)}_{bb}+S^{(+)}_{bb}\right]^{-1}\left[c^{(s)}_{b}+c^{(+)}_{b}\right]
\end{aligned}$$

The previous system suggests that the optimal term to add to the local matrix problem $K^{(s)}$ is $S^{(-)}$ on the left interface nodes and $S^{(+)}$ on the right ones. $\Omega^{(s)}$ is the only subdomain with non zero right-hand side, then $c^{(-)}_{b-1}=0$ and $c^{(+)}_{b}=0$ and the system is the exact condensation of local augmented problem

$$\begin{bmatrix} K^{(s)}_{ii} & K^{(s)}_{ib-1} & K^{(s)}_{ib} \\ K^{(s)}_{b-1i} & K^{(s)}_{b-1b-1}+S^{(-)}_{b-1b-1} & 0 \\ K^{(s)}_{bi} & 0 & K^{(s)}_{bb}+S^{(+)}_{bb} \end{bmatrix} \begin{bmatrix} x^{(s)}_{i} \\ x^{(s)}_{b-1} \\ x^{(s)}_{b} \end{bmatrix} = \begin{bmatrix} f^{(s)}_{i} \\ f^{(s)}_{b-1} \\ f^{(s)}_{b} \end{bmatrix} \tag{1.51}$$

In this case, we have the following theorem

Theorem 1.2. *In a case of a one way splitting, the Jacobi iterative algorithm for the two-Lagrange multiplier FETI method with augmented term equal to the complete outer Schur complement (as in Equation (1.51)) converges in a number of iteration equal to the number of subdomains minus one.*

Proof. See Roux, Magoulès, Salmon, and Series [67]. □

With this theorem, we know that in every subdomain the optimal augmentation term, in a one-way splitting, correspond to the Schur complement of the outer domain, which in practice is impossible to compute. Furthermore, this matrix is a full matrix, even if it was available for free, using it in the generalized Robin boundary condition would make the local problem too expensive to solve, so it is mandatory to find sparse approximations.

In this context, several approaches have been proposed [68], from which we can name the following:

Schur neighbor: In a physical context, the complete outer Schur complement matrix represents the interactions of all the degree of freedom of the subdomains, condensed on the interfaces. This condensation only with the neighboring subdomains, leads to approximate the complete outer Schur complement with the neighbor Schur complement. In this case, the cost of computation and exchange of data are reduced to the neighboring subdomains only. Even though this is a good theoretical approximation, in practice it still has a prohibitive cost. The following two approximations are based on this first approach. They try to mimic the behavior of the neighbor Schur complement using a sparse and not expensive approximation of it, which gives good convergence for the FETI-2LM method.

Lumped approximation: It has been shown in [67] that approximate the neighbor Schur complement matrix $K_{bb}^{(s)} - K_{bi}^{(s)} {K_{ii}^{(s)}}^{-1} K_{ib}^{(s)}$ with its first term $K_{bb}^{(s)}$ gives acceptable results. This alternative is straightforward to implement because this matrix is computed by the neighboring subdomain during the assembly procedure and the integration of the contribution of the interface nodes. Only one exchange with the neighboring subdomain is required for this regularization procedure, the results are however not as good as we would hope, so a third and better approximation will be finally used.

Sparse approximation based on patches along the interface: This sparse approximation for the neighboring Schur complement, leads to better results than the lumped approximation, as shown in [67] and is the only one used in our numerical results section. The goal is to emulate the spectral density of the neighbor Schur complement matrix as close as possible.

A dense, but cheaper approximation can be made if the condensation process is done only considering a neighboring area of the interface, that is, the Schur complement is computed by

$$\tilde{S}^{(s)} = K_{bb}^{(s)} - K_{bj}^{(s)} {K_{jj}^{(s)}}^{-1} K_{jb}^{(s)}, \qquad s = 1, 2, \forall j \in V_d$$

where the nodes j are no longer all the interior nodes, but instead they belong to

the set

V_d = { indexes of interior nodes, such that the minimum connectivity distance between each of these nodes and the nodes belonging to the interface is lower than $d, d \in \mathbb{N}$}.

Even though this procedure reduces the computational cost of the Schur complement, it generates a dense augmentation matrix, which adds an extra cost to the forward-backward substitution in the method. However it is from this idea that the sparse approximation is built.

We start again with the two subdomain case, as the general case is direct. The idea is to perform the condensation only in a small part of the interface, called patch [68]. Let us define the following subsets of indexes for the nodes of $\Omega^{(s)}, s = 1, 2$:

$V_{\Omega^{(s)}}$ = { indexes of nodes inside the subdomain $\Omega^{(s)}$}

V_Γ = { indexes of nodes inside the interface Γ}

V_p^j = { indexes of the nodes in V_Γ such that the minimum connectivity distance between each of these nodes and the node labelled j is lower or equal than $p, p \in \mathbb{N}$}

$V_{p,d}^j$ = { indexes of the nodes in $V_{\Omega^{(s)}}$ such that the minimum connectivity distance between each of these nodes and the nodes in V_p^j is lower or equal than $d, d \in \mathbb{N}$}

In other words, V_p^j is a patch of radius p around some node j, and $V_{p,d}^j$ is the neighboring area of depth d of this patch.

This approximation consists of defining a sparse augmentation matrix to avoid the increase in the bandwidth of the local problem matrix. This augmentation matrix is built by local condensation along the interface and extraction of some coefficients. The algorithm to compute the augmentation matrix $A_b^{(q)}$, $q = 1, 2$ is described in Algorithm 7.

Algorithm 7 Sparse approximation of neighbor Schur complement

1: Initialization of p and d.
2: Construction of the sparse structure of the interface matrix $A_b^{(q)} \in \mathbb{R}^{dimV_\Gamma \times dimV_\Gamma}$.
3: Construction of the sparse structure of the subdomain matrix $K^{(s)} \in \mathbb{R}^{dimV_{\Omega^{(s)}} \times dimV_{\Omega^{(s)}}}$.
4: Assembly of the matrix $K^{(s)}$.
5: **for** j in V_Γ **do**
6: Extraction of the coefficients $K^{(s)}_{mn}, (m,n) \in V^j_{p,d} \times V^j_{p,d}$, and construction of the sparse matrix $\tilde{K}^{(s)} \in \mathbb{R}^{dimV^j_{p,d} \times dimV^j_{p,d}}$ with these coefficients.
7: Computation of the dense matrix $\tilde{S}^{(s)}$ by condensation of the matrix $\tilde{K}^{(s)}$ on the patch V^j_p.
8: Extraction of the coefficients of the line associated with the node j from the matrix $\tilde{S}^{(s)}$ and insertion inside the matrix $A_b^{(q)}$ at the line associated with the node j.
9: **end for**
10: Construction of the symmetric matrix $A_b^{(q)} = \frac{1}{2}\left(A_b^{(q)^T} + A_b^{(q)}\right)$.

The augmentation matrix built up to the last line of the algorithm is non-symmetric, so we symmetrize the matrix to avoid this drawback.

The same algorithm is used in any other subdomain in the general case, considering one neighbor at the time to build one augmentation matrix $A^{(s)}_{\Gamma^j}$ for every neighbor that will be added to the corresponding local matrix $K^{(s)}$. As an example the regular mesh with $\mathbb{Q}_1$-finite elements is presented in Figure 1.6. This numbering leads to the definition, for the node 13, with $p = 1$ and $d = 1, 2$, of the subsets of indexes

$$V_1^{13} = \{7, 13, 19\}$$
$$V_{1,1}^{13} = \{1, 2, 8, 14, 20, 26, 25\}$$
$$V_{1,2}^{13} = \{1, 2, 8, 14, 20, 26, 25, 3, 9, 15, 21, 27, 33, 32, 31\}$$

These subsets correspond to the overlapping layers represented Figure 1.7.

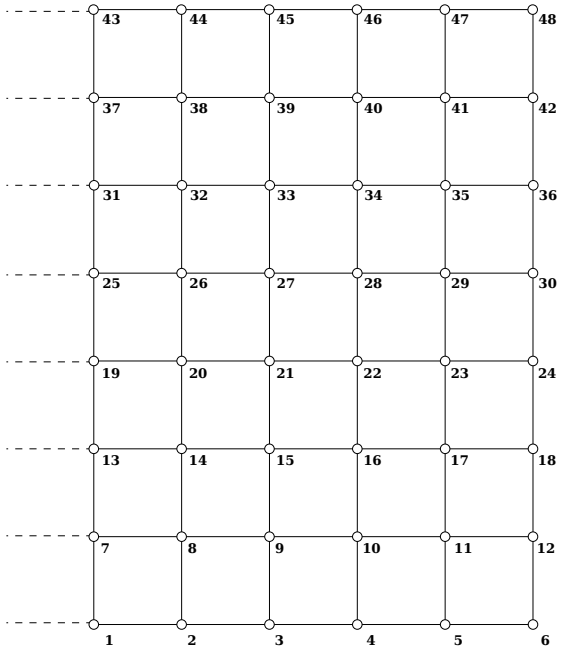

Figure 1.6 – Nodes numbering in subdomain $\Omega^{(s)}$.

With the explained computation of the augmentation matrices, the final algorithm for FETI-2LM is the iterative ORTHODIR method, already described theoretically in previous section Algorithm 6. The projection and preconditioning are not to be taken into account because the local problems are all well posed. As usual, the full orthogonalization process is kept as it is needed to avoid the loss of orthogonality produced when multiplying by the FETI operator (Although in this case is part of the ORTHODIR method).

In practice, even if the algorithms are similar between this method and the non-symmetric version of FETI-1LM, we have to point out that most of the implementation differences are in the construction of local problems. In the 2LM case, we have to add the computed $A^{(s)}$ matrices. The other big difference is in the computation by this new operator. In this case, is done by changing the assembly of the residuals according to the definition in Equation (1.49).

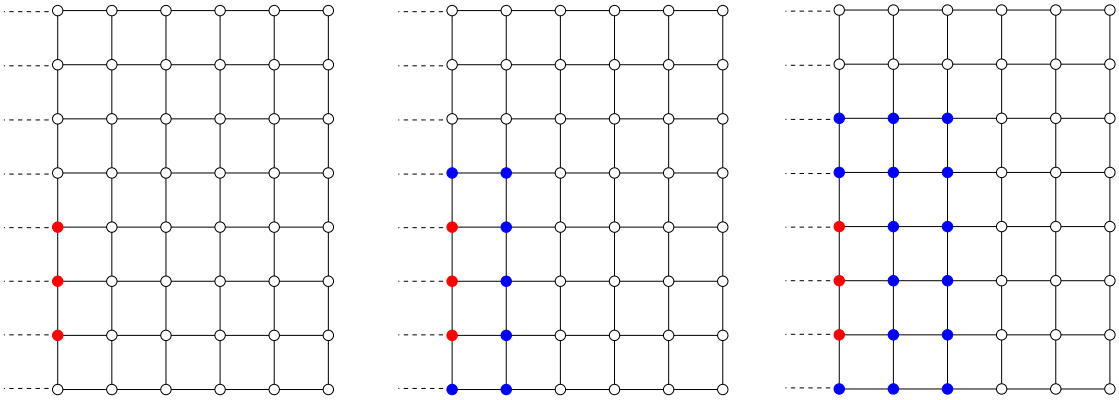

Figure 1.7 – Interface patch of size $p = 1$ (red) with one and two layers $d = 1, 2$ (blue).

1.3 New FETI as a hybrid between one and two Lagrange methods

1.3.1 Development

Now we are ready to present a new method that arises "directly" if we think of the properties of each previous method and the fact that both share similarities in their implementations. The main idea is to mix the good convergence and speed of the FETI-1LM method with the robustness of the FETI-2LM and apply it in more complex scenarios that usually we have to face in real life applications.

To show this, we can see in Figure 1.8 a contact problem coming from the scheme model of a rolling bearing. These kinds of problems can be easily extended to other contact schemes where we want to compute the interactions between objects of different materials, that can be as different as steel and rubber. The problems with heterogeneous contact boundary conditions imposed with the penalty method [81] are the type of situations that can make the discretization matrices of the model (and consequently the FETI operator) very ill-conditioned.

In this configuration a natural interface will be one that divides the two materials, if this is the case, then the FETI-1LM method with the extended preconditioner may be appropriate to handle the heterogeneities (even here the convergence is not assured if the differences are "extreme"). However, this is

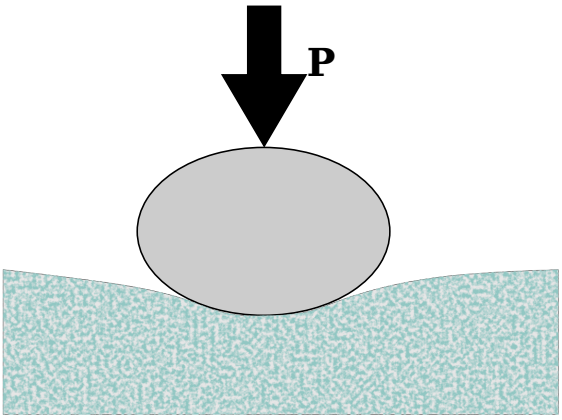

Figure 1.8 – A contact problem.

not always the case when you do an automatic partitioning of the global mesh, or if, for example, we use different discretizations that lead to different types of meshing, then we will have the case of a non-conforming mesh. Other cases were anisotropic materials needs to be modeled, will also give bad numerical features on the interface. For those problems, the FETI-1LM will not assure the usual fast convergence. The FETI-2LM method is, in general, a more robust method, due to the formulation of two independent Lagrange multipliers, that can overcome these numerical issues.

If we think now in the interfaces that are completely contained in one of the two material, or in general do not present numerical issues, the regular FETI method will be more suited as we will get a faster convergence.

Using one or the other will gives us either fast or accurate result, but we can try to get both of these good features if we choose properly which method to apply in every part of the interface. If we manage to make the choice that better suits every local interface, then we will most likely have a hybrid method with both good qualities.

We will call this method Hybrid-FETI and its formulation can be described mathematically as previous methods, but this time starting from a different most simple configuration, as we need at least two different local interfaces. For simplicity reasons, the basic problem consists of three subdomains and two interfaces that are entirely independent of one another (no crosspoints) as shown in Figure 1.9. In any case, the generalization is straightforward since the

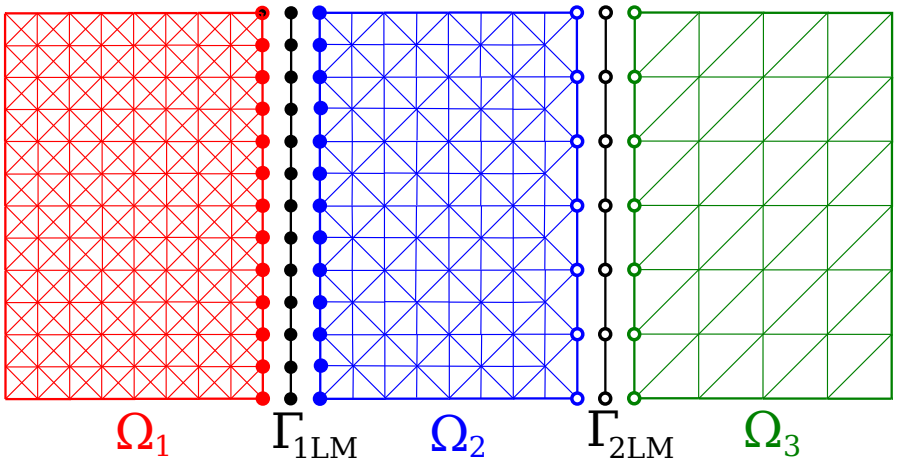

Figure 1.9 – Three subdomain divisions.

interfaces are always disconnected with FETI approaches.

Using the subscript notation of previous sections, we have i for interior nodes and b for the ones on the interface $\Gamma = \Gamma^1 \cup \Gamma^2$. Then, for each subdomain $\Omega^{(s)}$, $s = 1, 2, 3$ the contribution to the stiffness matrix and right-hand side of a finite element discretization is

$$K^{(s)} = \begin{bmatrix} K_{ii}^{(s)} & K_{ib}^{(s)} \\ K_{bi}^{(s)} & K_{bb}^{(s)} \end{bmatrix}, \qquad f^{(s)} = \begin{bmatrix} f_i^{(s)} \\ f_b^{(s)} \end{bmatrix} \tag{1.52}$$

The global discretization matrix of the form $Kx = f$ (as in (1.16)) can be written from the assembling of the local problems

$$\begin{bmatrix} K_{ii}^{(1)} & 0 & 0 & K_{ib}^{(1)} \\ 0 & K_{ii}^{(2)} & 0 & K_{ib}^{(2)} \\ 0 & 0 & K_{ii}^{(3)} & K_{ib}^{(3)} \\ K_{bi}^{(1)} & K_{bi}^{(2)} & K_{bi}^{(3)} & K_{bb} \end{bmatrix} \begin{bmatrix} x_i^{(1)} \\ x_i^{(2)} \\ x_i^{(3)} \\ x_b \end{bmatrix} = \begin{bmatrix} f_i^{(1)} \\ f_i^{(2)} \\ f_i^{(3)} \\ f_b \end{bmatrix}$$

up to this part, we have the same as previous methods. Now we need to look in detail the interactions that happen in this particular interface, so we can separate it and impose the different boundary conditions, to form the correspondent local problems for each method.

If we use the subscript b_1 for the interface Γ^1 in which we will impose the Neumann condition as in the 1LM method and the subscript b_2 for the 2LM interface where a Robin condition is imposed, then the previous local contributions are

$$K_{bi}^{(1)} = \begin{bmatrix} K_{b_1 i}^{(1)} \\ 0 \end{bmatrix}, \qquad K_{bi}^{(2)} = \begin{bmatrix} K_{b_1 i}^{(2)} \\ K_{b_2 i}^{(2)} \end{bmatrix}, \qquad K_{bi}^{(3)} = \begin{bmatrix} 0 \\ K_{b_2 i}^{(3)} \end{bmatrix}$$
$$K_{ib}^{(1)} = \begin{bmatrix} K_{ib_1}^{(1)} & 0 \end{bmatrix}, \quad K_{ib}^{(2)} = \begin{bmatrix} K_{ib_1}^{(2)} & K_{ib_2}^{(2)} \end{bmatrix}, \quad K_{ib}^{(3)} = \begin{bmatrix} 0 & K_{ib_2}^{(3)} \end{bmatrix}$$

As for the interaction between the interfaces, we have

$$K_{bb} = K_{bb}^{(1)} + K_{bb}^{(2)} + K_{bb}^{(3)} = \begin{bmatrix} K_{b_1 b_1}^{(1)} & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} K_{b_1 b_1}^{(2)} & 0 \\ 0 & K_{b_2 b_2}^{(2)} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & K_{b_2 b_2}^{(3)} \end{bmatrix}$$
$$f_b = f_b^{(1)} + f_b^{(2)} + f_b^{(3)} = \begin{bmatrix} f_{b_1}^{(1)} \\ 0 \end{bmatrix} + \begin{bmatrix} f_{b_1}^{(2)} \\ f_{b_2}^{(2)} \end{bmatrix} + \begin{bmatrix} 0 \\ f_{b_2}^{(3)} \end{bmatrix}$$

And the unknown displacements are

$$x_b = \begin{bmatrix} x_{b_1} \\ x_{b_2} \end{bmatrix}$$

With the previous separation, we can form the local Neumann, Robin and Hybrid problems to solve in each subdomain. For the first problem, we introduce a single langrage multiplier

$$\begin{bmatrix} K_{ii}^{(1)} & K_{ib_1}^{(1)} \\ K_{b_1 i}^{(1)} & K_{b_1 b_1}^{(1)} \end{bmatrix} \begin{bmatrix} x_i^{(1)} \\ x_{b_1}^{(1)} \end{bmatrix} = \begin{bmatrix} f_i^{(1)} \\ f_{b_1}^{(1)} + \lambda_{b_1}^{(1)} \end{bmatrix} \tag{1.53}$$

For the second subdomain, we form the hybrid local problem, where the Lagrange multiplier introduced for the first subdomain is shared in the "left" interface, and we add two extra multipliers to the interface in the “right” along

with the augmentation matrix, as in the 2LM method. With this we form

$$\begin{bmatrix} K_{ii}^{(2)} & K_{ib_1}^{(2)} & K_{ib_2}^{(2)} \\ K_{b_1i}^{(2)} & K_{b_1b_1}^{(2)} & 0 \\ K_{b_2i}^{(2)} & 0 & K_{b_2b_2}^{(2)} + A_{b_2}^{(2)} \end{bmatrix} \begin{bmatrix} x_i^{(2)} \\ x_{b_1}^{(2)} \\ x_{b_2}^{(2)} \end{bmatrix} = \begin{bmatrix} f_i^{(2)} \\ f_{b_1}^{(2)} + \lambda_{b_1}^{(2)} \\ f_{b_2}^{(2)} + \lambda_{b_2}^{(2)} \end{bmatrix} \tag{1.54}$$

And for the third subdomain

$$\begin{bmatrix} K_{ii}^{(3)} & K_{ib_2}^{(3)} \\ K_{b_2i}^{(3)} & K_{b_2b_2}^{(3)} + A_{b_2}^{(3)} \end{bmatrix} \begin{bmatrix} x_i^{(3)} \\ x_{b_2}^{(3)} \end{bmatrix} = \begin{bmatrix} f_i^{(3)} \\ f_{b_2}^{(3)} + \lambda_{b_2}^{(3)} \end{bmatrix} \tag{1.55}$$

In the interface between subdomains 1 and 2 we have one Lagrange multiplier, with opposite values stored in each subdomain, we will denote it as

$$\lambda_{b1} = \lambda_{b1}^{(1)} = -\lambda_{b1}^{(2)}$$

Now, using the same condensation process as in the previous methods, we will use the explicit relation between the multipliers and the displacements to eliminate the unknowns of the inner nodes of each subdomain. Consider the first subdomain, where we know from the first equation in (1.53) that

$$x_i^{(1)} = -K_{ii}^{(1)^{-1}} K_{ib_1}^{(1)} x_{b_1}^{(1)} + K_{ii}^{(1)^{-1}} f_i^{(1)}$$

introducing it in the second equation, we have

$$\begin{aligned} \lambda_{b_1} &= K_{b_1i}^{(1)} x_i^{(1)} + K_{b_1b_1}^{(1)} x_{b_1}^{(1)} - b_{f_1}^{(1)} \\ &= -K_{b_1i}^{(1)} K_{ii}^{(1)^{-1}} K_{ib_1}^{(1)} x_{b_1}^{(1)} + K_{b_1b_1}^{(1)} x_{b_1}^{(1)} - f_{b_1}^{(1)} + K_{b_1i}^{(1)} K_{ii}^{(1)^{-1}} f_i^{(1)} \\ &= S_{b_1b_1}^{(1)} x_{b_1}^{(1)} - c_{b_1}^{(1)} \end{aligned}$$

here $S_{b_1b_1}^{(1)} := K_{b_1b_1}^{(1)} - K_{b_1i}^{(1)} K_{ii}^{(1)^{-1}} K_{ib_1}^{(1)}$ is the Schur complement of the subdomain 1 in the interface 1LM and $c_{b_1}^{(1)} := f_{b_1}^{(1)} - K_{b_1i}^{(1)} K_{ii}^{(1)^{-1}} f_i^{(1)}$. With this, we can have the displacement as a function of λ_{b1}

$$x_{b_1}^{(1)} = F^{(1)} \lambda_{b_1} + F^{(1)} c_{b_1}^{(1)} \tag{1.56}$$

where

$$F^{(1)} := S_{b_1 b_1}^{(1)^{-1}}$$

is the inverse of the Schur complement. For the second subdomain, we have from the first line in (1.54) that

$$x_i^{(2)} = -K_{ii}^{(2)^{-1}} K_{ib_1}^{(2)} x_{b_1}^{(2)} - K_{ii}^{(2)^{-1}} K_{ib_2}^{(2)} x_{b_2}^{(2)} + K_{ii}^{(2)^{-1}} f_i^{(2)}$$

replacing it in the second and third equation, we have

$$\begin{aligned}
\begin{bmatrix} \lambda_{b_1} \\ \lambda_{b_2}^{(2)} \end{bmatrix} &= \begin{bmatrix} K_{b_1 i}^{(2)} x_i^{(2)} + K_{b_1 b_1}^{(2)} x_{b_1}^{(2)} - b_{b_1}^{(2)} \\ K_{b_2 i}^{(2)} x_i^{(2)} + (K_{b_2 b_2}^{(2)} + A_{b_2}^{(2)}) x_{b_2}^{(2)} - b_{b_2}^{(2)} \end{bmatrix} \\
&= \begin{bmatrix} K_{b_1 b_1}^{(2)} - K_{b_1 i}^{(2)} K_{ii}^{(2)^{-1}} K_{ib_1}^{(2)} & -K_{b_1 i}^{(2)} K_{ii}^{(2)^{-1}} K_{ib_2}^{(2)} \\ -K_{b_2 i}^{(2)} K_{ii}^{(2)^{-1}} K_{ib_1}^{(2)} & K_{b_2 b_2}^{(2)} - K_{b_2 i}^{(2)} K_{ii}^{(2)^{-1}} K_{ib_2}^{(2)} + A_{b_2}^{(2)} \end{bmatrix} \begin{bmatrix} x_{b_1}^{(2)} \\ x_{b_2}^{(2)} \end{bmatrix} + \begin{bmatrix} -f_{b_1}^{(2)} + K_{b_1 i}^{(2)} K_{ii}^{(2)^{-1}} f_i^{(2)} \\ -f_{b_2}^{(2)} + K_{b_1 i}^{(2)} K_{ii}^{(2)^{-1}} f_i^{(2)} \end{bmatrix} \\
&= \begin{bmatrix} S_{b_1 b_1}^{(2)} & S_{b_1 b_2}^{(2)} \\ S_{b_2 b_1}^{(2)} & S_{b_2 b_2}^{(2)} + A_{b_2}^{(2)} \end{bmatrix} \begin{bmatrix} x_{b_1}^{(2)} \\ x_{b_2}^{(2)} \end{bmatrix} - \begin{bmatrix} c_{b_1}^{(2)} \\ c_{b_2}^{(2)} \end{bmatrix}
\end{aligned}$$

where the S terms denotes the corresponding Schur complements of the subdomain 2 in the interfaces 1LM and 2LM and $c_{b_1}^{(2)} := f_{b_1}^{(2)} - K_{b_1 i}^{(2)} K_{ii}^{(2)^{-1}} f_i^{(2)}$, $c_{b_2}^{(2)} = f_{b_2}^{(2)} - K_{b_1 i}^{(2)} K_{ii}^{(2)^{-1}} f_i^{(2)}$. So the displacements of both local interfaces are

$$\begin{bmatrix} x_{b_1}^{(2)} \\ x_{b_2}^{(2)} \end{bmatrix} = F^{(2)} \begin{bmatrix} \lambda_{b_1} \\ \lambda_{b_2}^{(2)} \end{bmatrix} + F^{(2)} \begin{bmatrix} c_{b_1}^{(2)} \\ c_{b_2}^{(2)} \end{bmatrix}$$

here, we have

$$F^{(2)} := \begin{bmatrix} S_{b_1 b_1}^{(2)} & S_{b_1 b_2}^{(2)} \\ S_{b_2 b_1}^{(2)} & S_{b_2 b_2}^{(2)} + A_{b_2}^{(2)} \end{bmatrix}^{-1} \tag{1.57}$$

We can divide $F^{(2)}$ in blocks, such that

$$F^{(2)} =: \begin{bmatrix} F_{b_1 b_1}^{(2)} & F_{b_1 b_2}^{(2)} \\ F_{b_2 b_1}^{(2)} & F_{b_2 b_2}^{(2)} \end{bmatrix}$$

then we can also build a separate explicit relation for both displacements in this

subdomain

$$\begin{bmatrix} x_{b_1}^{(2)} \\ x_{b_2}^{(2)} \end{bmatrix} = \begin{bmatrix} F_{b_1b_1}^{(2)} \lambda_{b_1} + F_{b_1b_2}^{(2)} \lambda_{b_2}^{(2)} + F_{b_1b_1}^{(2)} c_{b_1}^{(2)} + F_{b_1b_2}^{(2)} c_{b_2}^{(2)} \\ F_{b_2b_1}^{(2)} \lambda_{b_1} + F_{b_2b_2}^{(2)} \lambda_{b_2}^{(2)} + F_{b_2b_1}^{(2)} c_{b_1}^{(2)} + F_{b_2b_2}^{(2)} c_{b_2}^{(2)} \end{bmatrix} \tag{1.58}$$

For the third subdomain, we do the same treatment to obtain

$$x_i^{(3)} = -K_{ii}^{(3)^{-1}} K_{ib_2}^{(3)} x_{b_2}^{(3)} + K_{ii}^{(3)^{-1}} f_i^{(3)}$$

and then replace it in the second equation of (1.55)

$$\begin{aligned} \lambda_{b_2}^{(3)} &= K_{b_2i}^{(3)} x_i^{(3)} + (K_{b_2b_2}^{(3)} + A_{b_2}^{(3)}) x_{b_2}^{(3)} - f_{b_2}^{(3)} \\ &= -K_{b_2i}^{(3)} K_{ii}^{(3)^{-1}} K_{ib_2}^{(3)} x_{b_2}^{(3)} + (K_{b_2b_2}^{(3)} + A_{b_2}^{(3)}) x_{b_2}^{(3)} - f_{b_2}^{(3)} + K_{b_2i}^{(3)} K_{ii}^{(3)^{-1}} f_i^{(3)} \\ &= (S_{b_2b_2}^{(3)} + A_{b_2}^{(3)}) x_{b_2}^{(3)} - c_{b_2}^{(3)} \end{aligned}$$

With analogous definitions of each term, as before. The equivalence in this case is given by the next equation

$$x_{b_2}^{(3)} = F^{(3)} \lambda_{b_2}^{(3)} + F^{(3)} c_{b_2}^{(3)} \tag{1.59}$$

where

$$F^{(3)} := (S_{b_2b_2}^{(3)} + A_{b_2}^{(3)})^{-1}$$

is the inverse of the Schur complements plus the augmentation term in this subdomain.

The continuity conditions needed in every case will vary depending on the interface. For the interface 1LM, eliminating the jump of the interface solution is enough to achieve this, meaning

$$x_{b_1}^{(1)} - x_{b_1}^{(2)} = 0 \tag{1.60}$$

but for the interface 2LM, the interface condition is not consistent, as seen in

(1.46). We need to add another constraint that gives us a residual of the form

$$\begin{aligned} x_{b_2}^{(2)} - x_{b_2}^{(3)} &= 0 \\ \lambda_{b_2}^{(2)} + \lambda_{b_2}^{(3)} - A_{b_2}^{(2)} x_{b_2}^{(2)} - A_{b_2}^{(3)} x_{b_2}^{(3)} &= 0 \end{aligned}$$

this mixed form is recombined as in the 2LM method. Together with Equation (1.60) we obtain the conditions needed to impose continuity across complete the interface

$$\begin{aligned} x_{b_1}^{(1)} - x_{b_1}^{(2)} &= 0 \\ \lambda_{b_2}^{(2)} + \lambda_{b_2}^{(3)} - (A_{b_2}^{(2)} + A_{b_2}^{(3)}) x_{b_2}^{(2)} &= 0 \\ \lambda_{b_2}^{(2)} + \lambda_{b_2}^{(3)} - (A_{b_2}^{(2)} + A_{b_2}^{(3)}) x_{b_2}^{(3)} &= 0 \end{aligned}$$

With these equations plus the relations between the Lagrange multipliers and the different displacement, coming from the solution of local problems (1.56),(1.58),(1.59), we can do the condensation of previous problems on to the interface for the Hybrid method

$$\begin{bmatrix} F^{(1)} + F_{b_1 b_1}^{(2)} & -F_{b_1 b_2}^{(2)} & 0 \\ \left(A_{b_2}^{(2)} + A_{b_2}^{(3)}\right) F_{b_2 b_1}^{(2)} & I - \left(A_{b_2}^{(2)} + A_{b_2}^{(3)}\right) F_{b_2 b_2}^{(2)} & I \\ 0 & I & I - \left(A_{b_2}^{(2)} + A_{b_2}^{(3)}\right) F^{(3)} \end{bmatrix} \begin{bmatrix} \lambda_{b_1} \\ \lambda_{b_2}^{(2)} \\ \lambda_{b_2}^{(3)} \end{bmatrix}$$

$$= \begin{bmatrix} -F^{(1)} c_{b_1}^{(1)} + F_{b_1 b_1}^{(2)} c_{b_1}^{(2)} + F_{b_1 b_2}^{(2)} c_{b_2}^{(2)} \\ \left(A_{b_2}^{(2)} + A_{b_2}^{(3)}\right)\left(F_{b_2 b_1}^{(2)} c_{b_1}^{(2)} + F_{b_2 b_2}^{(2)} c_{b_1}^{(2)}\right) \\ \left(A_{b_2}^{(2)} + A_{b_2}^{(3)}\right) F^{(3)} c_{b_2}^{(3)} \end{bmatrix}$$

This matrix is not symmetric, so an ORTHODIR method with full reconjugation is used to solve it. We name the previous method Hybrid-FETI as in a hybridization between FETI-1LM and FETI-2LM.

1.3.2 Extension to a general problem

For the case of a general problem with $N_s > 2$, we proceed as in the 2LM method, and we divide the total interface into interface edges (local neighboring interface) so that each one is treated depending on the chosen method. We split the total interface nodes into edges, that correspond to a collection of connecting interface nodes, denoted Γ^j as we did for the 2LM case.

We add a unique marker to these edges to know a priori which one corresponds to each method; we denote the marked edges as Γ^j_{1lm} and Γ^j_{2lm} for both FETI-1LM and FETI-2LM methods respectively. In particular, crosspoints are edges of one node, where we can arbitrarily mark them 1LM or 2LM.

We also define the boolean matrix $B^{(s)}_{\Gamma^j}$ for a subdomain as the restriction of a vector defined in $\Omega^{(s)}$ to its values on the interface Γ^j, i.e

$$B^{(s)}_{\Gamma^j} v^{(s)} = v^{(s)}_b|_{\Gamma^j}$$

where $v^{(s)} = \begin{bmatrix} v^{(s)}_i \\ v^{(s)}_b \end{bmatrix}$. We keep the notation for the interior and interface nodes as i and b respectively.

With these definitions, we can write the local problem for subdomain $\Omega^{(s)}$ as well as the conditions to get the continuity, for the general case

$$\left(K^{(s)} + \sum_{\Gamma^j_{2lm} \subseteq \Omega^{(s)}} B^{(s)^T}_{\Gamma^j_{2lm}} A^{(s)}_{\Gamma^j_{2lm}} B^{(s)}_{\Gamma^j_{2lm}} \right) x^{(s)} = f^{(s)} - \sum_{\Gamma^l \subseteq \Omega^{(s)}} B^{(s)^T}_{\Gamma^l} \lambda^{(s)}_{\Gamma^l}, \qquad s = 1, \dots, N_s$$

$$B^{(s)}_{\Gamma^j} x^{(s)} - x_b|_{\Gamma^j} = 0 \qquad \forall \Gamma^j, \quad \Omega^{(s)} \supseteq \Gamma^j$$

$$\sum_{\Omega^{(q)} \supseteq \Gamma^j_{2lm}} \left(\lambda^{(q)}_{\Gamma^j_{2lm}} - A^{(q)}_{\Gamma^j_{2lm}} x_b|_{\Gamma^j_{2lm}} \right) = 0 \qquad \forall \Gamma^j_{2lm}$$

where $A^{(s)}_{\Gamma^j_{2lm}}$ is the augmentation matrix in subdomain $\Omega^{(s)}$ for the interface Γ^j_{2lm}. The other terms are analogous to FETI-2LM. Let us also define the total augmentation matrix for a subdomain that contains an edge "$2LM$"; if there are

no interfaces 2LM, its value is zero

$$A^{(s)}_{2lm} := \sum_{\Gamma^j_{2lm} \subseteq \Omega^{(s)}} B^{(s)^T}_{\Gamma^j_{2lm}} A^{(s)}_{\Gamma^j_{2lm}} B^{(s)}_{\Gamma^j_{2lm}}$$

With this equation, we can do the condensation and obtain the problem to solve on the interface. In general, the two methods are used, so the equations on the interface will be separated. Finally, we have the following interface problems

$$\begin{aligned}
&\sum_{\Omega^{(q)} \supseteq \Gamma^j_{2lm}} \left(\lambda^{(q)}_{\Gamma^j_{2lm}} - A^{(q)}_{\Gamma^j_{2lm}} B^{(s)}_{\Gamma^j_{2lm}} \left(K^{(s)} + A^{(s)}_{2lm} \right)^{-1} \sum_{\Gamma^l \subseteq \Omega^{(s)}} B^{(s)^T}_{\Gamma^l} \lambda^{(s)}_{\Gamma^l} \right) \\
&= - \sum_{\Omega^{(q)} \supseteq \Gamma^j_{2lm}} A^{(q)}_{\Gamma^j_{2lm}} B^{(s)}_{\Gamma^j_{2lm}} \left(K^{(s)} + A^{(s)}_{2lm} \right)^{-1} f^{(s)} \qquad \forall \Gamma^j_{2lm}, \forall \Omega^{(s)} \supseteq \Gamma^j_{2lm}
\end{aligned}$$

$$\begin{aligned}
&B^{(s)}_{\Gamma^j_{1lm}} \left(K^{(s)} + A^{(s)}_{2lm} \right)^{+} \left(\lambda_{\Gamma^j_{1lm}} + \sum_{\Gamma^l_{2lm} \subseteq \Omega^{(s)}} B^{(s)^T}_{\Gamma^l_{2lm}} \lambda^{(s)}_{\Gamma^l_{2lm}} \right) + B^{(s)}_{\Gamma^j_{1lm}} R^{(s)} \alpha^{(s)} \\
&+ B^{(q)}_{\Gamma^j_{1lm}} \left(K^{(q)} + A^{(q)}_{2lm} \right)^{+} \left(\lambda_{\Gamma^j_{1lm}} - \sum_{\Gamma^l_{2lm} \subseteq \Omega^{(q)}} B^{(q)^T}_{\Gamma^l_{2lm}} \lambda^{(q)}_{\Gamma^l_{2lm}} \right) - B^{(q)}_{\Gamma^j_{1lm}} R^{(q)} \alpha^{(q)} = \\
&- B^{(s)}_{\Gamma^j_{1lm}} \left(K^{(s)} + A^{(s)}_{2lm} \right)^{+} f^{(s)} + B^{(q)}_{\Gamma^j_{1lm}} \left(K^{(q)} + A^{(q)}_{2lm} \right)^{+} f^{(q)} \qquad \forall \Gamma^j_{1lm} = \Omega^{(s)} \cap \Omega^{(q)}
\end{aligned}$$

where the term $R^{(s)}$ (or $R^{(q)}$) is a kernel generator for the matrix $K^{(s)}$. The coefficients $\alpha^{(s)}$ are such that the admissibility condition (1.31) for this "floating domain" is fulfilled. We need then to use the same projection strategy explained in section 1.1.2, to treat these subdomains. In the case that some subdomain contains a 2LM interface large enough (for example, two or more nodes for a 2D elasticity problem, three for 3D elasticity, etc.) the term $A^{(s)}_{2lm}$ makes that there is no longer a kernel, so $R^{(s)}$ is zero.

This interface problem can also be written as $F\lambda = d$ with F being non-symmetric; hence we use the ORTHODIR method version with left preconditioner and full reconjugation. We will talk about this preconditioner in the next section.

1.3.3 Preconditioner

If we want this method to be scalable with respect to the mesh size, we need to find a preconditioner as in FETI-1LM. Up to this day, no optimal preconditioner has been found for FETI-2LM, but this method has been tested and shown to be numerically scalable for the mesh size on its own [31]. No extra computation needs to be done for the interface edges where the Robin condition is imposed. On the other hand, for the edges where one Lagrange multiplier is used, a preconditioner is mandatory to keep the convergence independent of the mesh size.

We will base the preconditioner on local information, just as the FETI-1LM does. If we look at the Dirichlet preconditioner, we know that it is computed by solving a local problem with imposed Dirichlet boundary conditions. From a mechanical point of view, this condition is the jump in the displacements fields along the subdomain interfaces. The solution of this problem allows to compute the corresponding interface traction forces and therefore a correction to the FETI operator.

From this process, we built the Dirichlet preconditioner, as already detailed in subsection 1.1.3, by applying it to all the interface edges and then adding these local contributions.

The multiplication by the operator that defines this preconditioner needs the solution of the following Dirichlet problem in $\Omega^{(s)}$

$$\begin{bmatrix} K_{ii}^{(s)} & K_{ib}^{(s)} \\ 0 & I \end{bmatrix} \begin{bmatrix} w_i^{(s)} \\ w_b^{(s)} \end{bmatrix} = \begin{bmatrix} 0 \\ \{w_b^{(s)}\} \end{bmatrix}$$

the subscript i is for interior nodes and b for the interface. Therefore the jump is imposed along the complete local interface.

We can use this preconditioner as a first approximation for our method, in this case, the definition will be similar to the one in FETI-1LM, but we apply it only on the interface edges Γ^j marked as 1LM, leaving the 2LM interfaces untouched.

We compute the multiplication of a vector w restricted to the interface edges

Γ^j_{1lm} by our preconditioner as

$$D^{-1}_{\Gamma^j_{1lm}} w|_{\Gamma^j_{1lm}} = \left(B^{(s)}_{\Gamma^j_{1lm}} S^{(s)}_{bb} B^{(s)^T}_{\Gamma^j_{1lm}} + B^{(q)}_{\Gamma^j_{1lm}} S^{(q)}_{bb} B^{(q)^T}_{\Gamma^j_{1lm}} \right) w|_{\Gamma^j_{1lm}}, \quad \forall \Gamma^j_{1lm} : \Gamma^j_{1lm} = \partial\Omega^{(s)} \cap \partial\Omega^{(q)}$$

where $S^{(s)}_{bb}$ (or $S^{(q)}_{bb}$) is the Schur complement of the subdomain interior nodes into the local interface. The matrices $B^{(s)}$ are the signed assembly operators, but can also be taken as their generalization for heterogeneous problems, i.e $\tilde{B}^{(s)} = B^{(s)}\beta^{(s)}$ with $\beta^{(s)}$ some scaling matrix.

The multiplication by this operator is performed in an analogous way to the case 1LM. First computing the Schur complements in each side of the interface, then making the exchange with the neighbor subdomain. We can write the previous preconditioner as a global matrix of the form

$$D^{-1} w = \left(\sum_{\Gamma^j_{1lm} \subseteq \Gamma} D^{-1}_{\Gamma^j_{1lm}} \right) w$$

with w any vector defined in the interface Γ.

One of the problems that we encounter with this preconditioner occurs in some subdomains where the local interface is composed of both 1LM and 2LM edges. In this case, when we apply the previous preconditioner, we are also imposing a jump in the 2LM edges, but not of the same type, as a Robin condition is used here. In Figure 1.10 we can see the two different boundary conditions that can be used, the upper one corresponding to the regular Dirichlet preconditioner, and the bottom one representing a new option to build a different preconditioner.

The advantage of this boundary condition is that now we obtain the exact local inverse of the Hybrid-FETI operator. The preconditioner is changed to solve this new problem in the subdomains with shared edges. It involves the computation of a different Schur complement, this time considering the augmentation matrix. The objective is to improve the representation of the behavior of the local interfaces marked as 2LM and therefore of the whole subdomain.

Formally this is achieved by solving the following Dirichlet problem in every

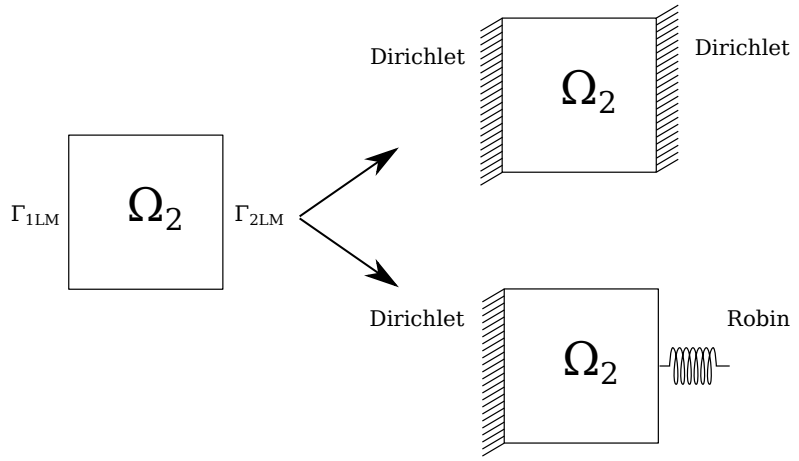

Figure 1.10 – Boundary conditions for both preconditioners.

subdomain with at least one edge marked as 1LM

$$\begin{bmatrix} K_{ii}^{(s)} & K_{ib_{1lm}}^{(s)} & K_{ib_{2lm}}^{(s)} \\ 0 & I & 0 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} w_i^{(s)} \\ w_{b_{1lm}}^{(s)} \\ w_{b_{2lm}}^{(s)} \end{bmatrix} = \begin{bmatrix} 0 \\ \{w_{b_{1lm}}^{(s)}\} \\ 0 \end{bmatrix}$$

where b_{1lm}, b_{2lm} are subscript for the nodes belonging to the local edges marked as 1LM and 2LM respectively. Here we impose the computed jump only in the 1LM edges and zero in the 2LM. Then, we compute the respective traction forces by multiplying against the matrix of the local problem. This leads to the computation of the following Schur complement

$$\begin{aligned} S_{bb}^{(s)} &= \begin{bmatrix} K_{b_{1lm}b_{1lm}}^{(s)} & 0 \\ 0 & K_{b_{2lm}b_{2lm}}^{(s)} + A_{b_{2lm}}^{(s)} \end{bmatrix} - \begin{bmatrix} K_{b_{1lm}i}^{(s)} \\ K_{b_{2lm}i}^{(s)} \end{bmatrix} K_{ii}^{(s)^{-1}} \begin{bmatrix} K_{ib_{1lm}}^{(s)} & K_{ib_{2lm}}^{(s)} \end{bmatrix} \\ &= \begin{bmatrix} \tilde{S}_{b_{1lm}b_{1lm}}^{(s)} & S_{b_{1lm}b_{2lm}}^{(s)} \\ S_{b_{2lm}b_{1lm}}^{(s)} & S_{b_{2lm}b_{2lm}}^{(s)} + A_{b_{2lm}}^{(s)} \end{bmatrix} \end{aligned}$$

The structure of this matrix is the same as usual and corresponds to the local inverse of the condensed interface problem (as seen in (1.57)).

With this computation of the Schur complement, the preconditioner for every interface edge Γ^j with one Lagrange multiplier is defined the same way as previ-

ous (remembering that the Schur complement now includes the augmentation matrices)

$$R^{-1}_{\Gamma^j_{1lm}} w|_{\Gamma^j_{1lm}} = \left(B^{(s)}_{\Gamma^j_{1lm}} \tilde{S}^{(s)}_{bb} B^{(s)^T}_{\Gamma^j_{1lm}} + B^{(q)}_{\Gamma^j_{1lm}} \tilde{S}^{(q)}_{bb} B^{(q)^T}_{\Gamma^j_{1lm}} \right) w|_{\Gamma^j_{1lm}},$$
$$\forall \Gamma^j_{1lm} : \Gamma^j_{1lm} = \partial\Omega^{(s)} \cap \partial\Omega^{(q)}$$

the definition of every term, other than the Schur complement, is the same as previous (including the consideration of the B matrices as scaling ones). This preconditioner can also be written in the form of a global matrix denoted R^{-1} as in Robin preconditioner

$$R^{-1} w = \left(\sum_{\Gamma^j_{1lm} \subseteq \Gamma} R^{-1}_{\Gamma^j_{1lm}} \right) w$$

1.3.4 Implementation

The parallel computation of the FETI methods is done by subdomain structures that are used to store the residuals of the FETI methods in general. These structures for both FETI-1LM and FETI-2LM are basically the same. In practical terms, even if we use one Lagrange multiplier (gradient) for the interface, the values of these vectors will be stored in both subdomains sharing the local edge (actually one saves $\lambda|_{\Gamma^j}$ and the other $-\lambda|_{\Gamma^j}$).

Independent of the method, to obtain the residuals, first we need to compute $x^{(s)}_b$. This is done via solving the corresponding local Neumann and Robin problems. In general, when using Hybrid-FETI there will be at least one subdomain where the local problem solved comes from imposing both conditions at the same time.

Let us recall how the gradients are computed for each method. To do so, let us take as an example the division of a domain into two subdomains $\Omega^{(1)}$, $\Omega^{(2)}$ with an interface Γ^b. We take this simple case because the procedure is the same for every interface edge in a more general partition.

The preconditioned gradient to compute for the 1LM method is

$$g = D^{-1}(F\lambda - d) = D^{-1}(x_b^{(2)} - x_b^{(1)})$$

where $x_b^{(s)}$, $s = 1, 2$ are the displacement of the interface nodes in each subdomain. No projection will be shown here as the implementation is analogous to the 1LM method and applies only to subdomains where the local interface is completely marked as 1LM.

From an implementation point of view, if we look at previous computation, we have that for each process the gradient is obtained as

$$g = \begin{bmatrix} g_b^{(1)} \\ g_b^{(2)} \end{bmatrix} = D^{-1} \begin{bmatrix} x_b^{(2)} - x_b^{(1)} \\ x_b^{(1)} - x_b^{(2)} \end{bmatrix}$$

where the process s computes and stores $g_b^{(s)}$, $s = 1, 2$. We also note that $g_b^{(1)} = -g_b^{(2)}$ as in this case the unknown is a single Lagrange multiplier along the interface.

Now for the 2LM method, the gradient is computed from equations (1.47), meaning that for each process we have

$$g = \begin{bmatrix} g_b^{(1)} \\ g_b^{(2)} \end{bmatrix} = \begin{bmatrix} \lambda_b^{(1)} + \lambda_b^{(2)} - (A_b^{(1)} + A_b^{(2)})x_b^{(2)} \\ \lambda_b^{(2)} + \lambda_b^{(1)} - (A_b^{(1)} + A_b^{(2)})x_b^{(1)} \end{bmatrix}$$

the values of $g_b^{(1)}$ and $g_b^{(2)}$ are different because two independent Lagrange multipliers are used for this method. However, the computation structure is the same as the FETI-1LM method. The difference lies in the exchanges needed to compute both gradients; in the case of the 2LM method, more than one exchange is needed.

The calculation of the gradients is a basis to the computation by the F operator (needed in the use of the iterative method). In the case of an arbitrary number of subdomains, the previous computations are extended by defining a division of the whole interface into interface edges. Therefore, the multiplication by the FETI operator of any vector defined in the interface v, is done by:

- Solve the local problems $\left(K^{(s)} + A^{(s)}_{2lm}\right)$ with the right-hand side equal to $t^{(s)^T} B^{(s)} v = \begin{bmatrix} 0 \\ v_b^{(s)} \end{bmatrix}$, then compute $x_b^{(s)}$.

- Assembly of the corresponding gradient, passing the information from one subdomain to its neighbors.

The first item is common between the FETI-1LM, FETI-2LM and our method. The difference lies in the existence of the augmentation terms when some of the edges are marked as 2LM. Is in the assembly of the gradient where we need to do the mix of both methods.

To explain this in detail, let us take the same case of a three subdomain division as in Figure 1.9 with one Lagrange multiplier to the "left" (1LM) and two to the "right" (2LM). If we want to solve it using the FETI-1LM method, we will have the following preconditioned gradient at each iteration

$$g = \begin{bmatrix} g_{b_1}^{(1)} \\ g_{b_1}^{(2)} \\ g_{b_2}^{(2)} \\ g_{b_2}^{(3)} \end{bmatrix} = D^{-1} \begin{bmatrix} x_{b_1}^{(2)} - x_{b_1}^{(1)} \\ x_{b_1}^{(1)} - x_{b_1}^{(2)} \\ x_{b_2}^{(3)} - x_{b_2}^{(2)} \\ x_{b_2}^{(2)} - x_{b_2}^{(3)} \end{bmatrix} = \left(D_{\Gamma^{b_1}}^{-1} + D_{\Gamma^{b_2}}^{-1}\right) \begin{bmatrix} x_{b_1}^{(2)} - x_{b_1}^{(1)} \\ x_{b_1}^{(1)} - x_{b_1}^{(2)} \\ x_{b_2}^{(3)} - x_{b_2}^{(2)} \\ x_{b_2}^{(2)} - x_{b_2}^{(3)} \end{bmatrix}$$

where the multiplication by the preconditioner was explained in the subsection 1.1.3.

If we think about solving it using only the FETI-2LM method, then the gradient will be

$$g = \begin{bmatrix} g_{b_1}^{(1)} \\ g_{b_1}^{(2)} \\ g_{b_2}^{(2)} \\ g_{b_2}^{(3)} \end{bmatrix} = \begin{bmatrix} \lambda_{b_1}^{(1)} + \lambda_{b_1}^{(2)} - (A_{b_1}^{(1)} + A_{b_1}^{(2)}) x_{b_1}^{(2)} \\ \lambda_{b_1}^{(2)} + \lambda_{b_1}^{(1)} - (A_{b_1}^{(2)} + A_{b_1}^{(1)}) x_{b_1}^{(1)} \\ \lambda_{b_2}^{(2)} + \lambda_{b_2}^{(3)} - (A_{b_2}^{(2)} + A_{b_2}^{(3)}) x_{b_2}^{(2)} \\ \lambda_{b_2}^{(3)} + \lambda_{b_2}^{(2)} - (A_{b_2}^{(3)} + A_{b_2}^{(2)}) x_{b_2}^{(1)} \end{bmatrix}$$

As for the preconditioned gradient of our method, the computation is done by

assembling

$$g = \begin{bmatrix} g_{b_1}^{(1)} \\ g_{b_1}^{(2)} \\ g_{b_2}^{(2)} \\ g_{b_2}^{(3)} \end{bmatrix} = D^{-1}_{\Gamma^{b_1}_{1lm}} \begin{bmatrix} x_{b_1}^{(2)} - x_{b_1}^{(1)} \\ x_{b_1}^{(1)} - x_{b_1}^{(2)} \\ \lambda_{b_2}^{(2)} + \lambda_{b_2}^{(3)} - (A_{b_2}^{(1)} + A_{b_2}^{(2)}) x_{b_2}^{(2)} \\ \lambda_{b_2}^{(3)} + \lambda_{b_2}^{(2)} - (A_{b_2}^{(2)} + A_{b_2}^{(1)}) x_{b_2}^{(1)} \end{bmatrix}$$

in here the preconditioner D^{-1} is reduced to $D^{-1}_{\Gamma^{b_1}_{1lm}}$ and affects the gradient only in the interface edge 1LM leaving the last two lines untouched.

The previous computation of the residual allows us to obtain likewise the multiplications by the FETI operator for each of the three methods, by changing only the corresponding gradient.

With this, the new method is complete, and a formal algorithm can be described and implemented. As already said, the operator is non-symmetric because it shares blocks of the non-symmetric FETI-2LM operator, so we use the iterative ORTHODIR method with left preconditioner and full reconjugation, for the same reasons as we did it in the unsymmetric and 2LM FETI methods.

The FETI-1LM method for unsymmetric matrices described in Algorithm 6 is the basis of our implementation. Both of them share a theoretical resemblance, changing the definition of our operator F and also modifying D to D_{1lm} when using the preconditioner, because it is used only in the 1LM interface edges.

In practice, the modifications are more complex, so we are listing them

- Marking of the interface edges.
- Correct construction of the local problems, meaning that we add the augmentation matrix in the subdomains with an edge 2LM.
- Change in the computation of the residuals the multiplication by the new operator.
- Modify the local Dirichlet problem to apply the different preconditioners and use them only in the 1LM interface edges.
- Project only in the subdomains with a complete local interface 1LM (this is done automatically, after computing the corresponding kernel in each subdomain).

In the following part, we will test numerically our method to show the advantages of it on some particular difficult problems.

1.4 Numerical results

In this section, we will show a comparative performance of the methods presented in this chapter.

To begin with, we present the formulation of the Poisson problem in 3D with Dirichlet condition in a part of the boundary. Let $\Omega \subseteq \mathbb{R}^3$ be a bounded domain and let $\partial\Omega_D \subseteq \partial\Omega$ be a part of the boundary, where zero Dirichlet conditions will be imposed. Given $f \in L^2(\Omega)$, the problem is

Find $u \in H^1(\Omega)$ *such that*

$$\begin{cases} -\nu\Delta u &= f \quad in \quad \Omega \\ u &= 0 \quad on \quad \partial\Omega_D \\ \dfrac{\partial u}{\partial n} &= 0 \quad on \quad \partial\Omega \setminus \Omega_D \end{cases}$$

with $\nu \in \mathbb{R}^+$, the parameter that will define the difference between materials and therefore in the local stiffness matrices of any possible division of Ω.

Let $V := \{v \in H^1(\Omega) : v|_{\partial\Omega_D} = 0\}$. Then, the variational formulation of previous problem is

Find $u \in V$ *such that*

$$\nu \int_\Omega \nabla u \cdot \nabla v dx = \int_\Omega f v, \quad \forall v \in V$$

This problem is discretized using tri-linear $\mathbb{Q}^1$ finite elements functions, leading to the known global system of equations

$$Kx = f$$

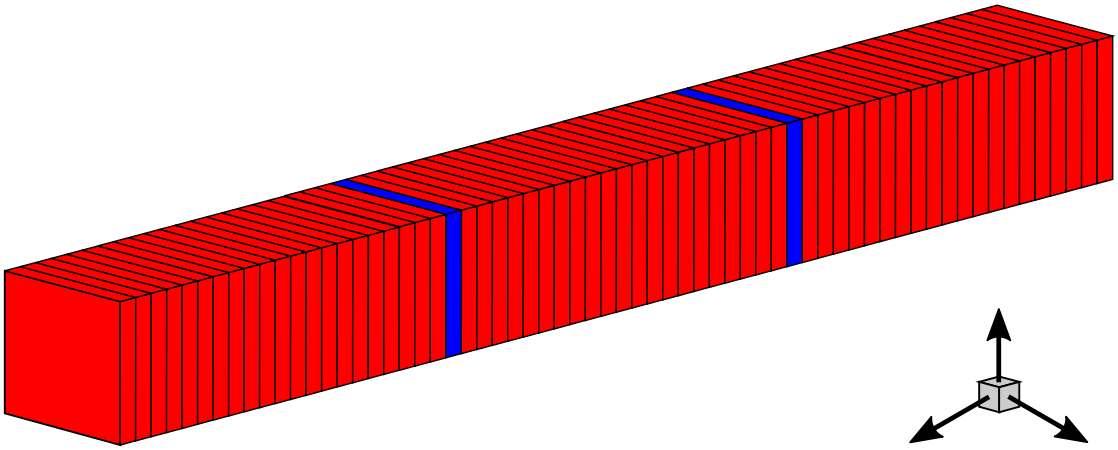

Figure 1.11 – Two material bar.

1.4.1 Two material bar

In our first test, we will model a bar of two different materials with some non zero source in one direction, see Figure 1.11. In this case, $\Omega = [0,10]\times[0,1]\times[0,1]$, $\mathbf{f} = (0,0,g)^T$. The boundary is $\partial\Omega_D = \{(x,y,z) \in \partial\Omega : x = 0 \vee x = 10\}$.

The problem consists of a bar with two blocks of a different material across the x-axis; they are located at the same distance of each boundary. The idea is to represent a localized "heterogeneous" zone in a general homogeneous domain, leaving only a few interfaces with a bad local conditioning. For each material we have the following values of their parameters

$$\nu_1 = 10^5, \text{ "blue" blocks}$$
$$\nu_2 = 10^0, \text{ otherwise}$$

Depending on the domain partition done, we change the size of the blocks to match the interfaces of the subdomains in the different divisions. Since the entire domain is divided into slices, we assign one of the values of ν to each subdomain, leaving only two in each case different from the rest, this way we just have heterogeneities across four of the interfaces.

But before doing any comparative test against other FETI methods, we need to check the performance of both possible preconditioners for this method,

Number of subdomains	Hybrid-FETI Dirichlet	Hybrid-FETI Robin
64	20	18
125	20	20
250	19	21

Table 1.1 – Convergence of Hybrid-FETI preconditioner (Number of iterations)

namely the Dirichlet or Robin one. We will test this problem for three different number of subdomains. The number of global elements in each computation is constant of 75 thousand elements per subdomain. Also, following the definition of Algorithm 7, we calculate the augmentation matrices for the 2LM interfaces by setting the patch size as 1 and the depth size as 3.

For all cases in this chapter, we use the following global stopping criterion

$$\frac{\|Kx^p - f\|_2}{\|f\|_2} < 10^{-4}$$

but we also use a second criterion, that is, the relative norm of the jump of the solution between subdomains [26].

$$\frac{\|x^p - x^{p-1}\|_2}{\|x^p\|_2} < 10^{-4}$$

The results are shown in Table 1.1. In this case, there is no significant difference between both of them, even in some cases, the Dirichlet preconditioner performs better than the Robin. We think that this is because the small number of interfaces where the preconditioner is different is not large enough to let the differences to be clear. In examples where more subdomains share the two types of interface, the difference is more evident (See the contact case (1.4.2)).

We also note that almost no augmentation in the number of iterations is present, even if we have increased the number of subdomains, this good quality is explained because most of the interfaces were marked as 1LM, meaning that we are very close to the regular FETI-1LM method. We hope in any case to keep this property even for the harder problems.

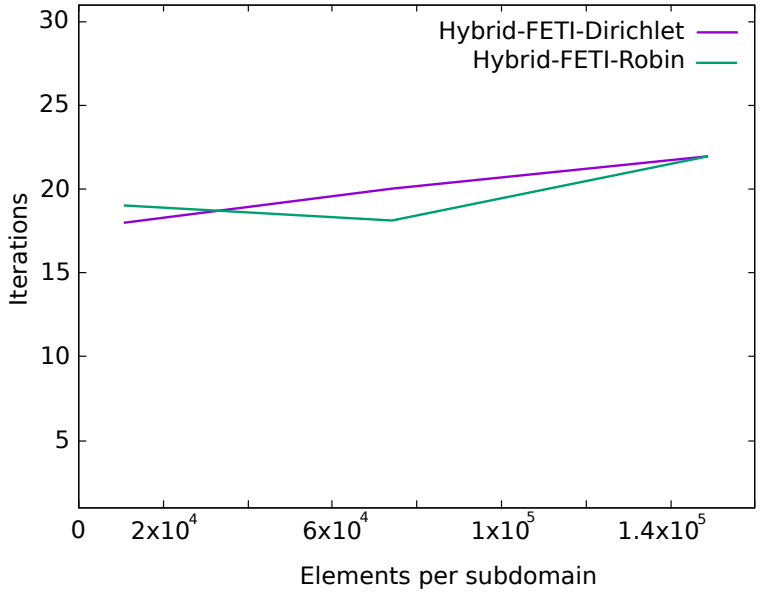

Figure 1.12 – Hybrid-FETI Iterations versus Elements number.

We will see now, how the Hybrid method performs when increasing the number of elements per subdomains. We hope that also some of the other properties of the methods FETI-1LM and FETI-2LM hold in the Hybrid case. In particular, the property of having only a small increase in the number of iterations when the number of elements in a subdomain is augmented. Hence, we test different values of h (mesh size) for a constant of 64 subdomains. The results are summarized in Figure 1.12.

From Table 1.1 and Figure 1.12 we can see that, as expected, the number of iterations after augmenting the local size of the problems or the number of subdomains does not change considerably. This is because this method is built from a combination of two numerically scalable methods, and we keep the projection and preconditioner that justify the scalability in the first place.

One of the main issues of the new method lies in the problem of which interfaces should be marked as 1LM or 2LM. The intuition tells that the interfaces where the materials change should be 2LM and the rest 1LM. We will now test this choice against a different one, where two additional interfaces are marked as 2LM, as seen in Figure 1.13. The idea is to cover the problematic local interfaces,

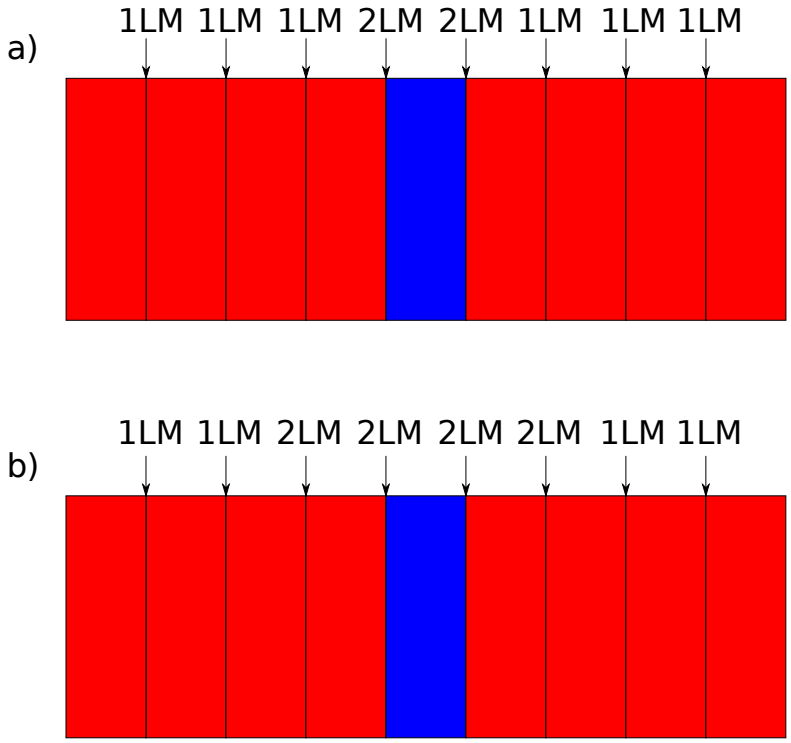

Figure 1.13 – a) Regular interface marking. b) Extra covering marking.

in this case, the jump of materials. In some other cases this problematic edges will not be so clear, so extending the 2LM marked interfaces will result mainly in a more robust method, as the 1LM edges may not always achieve convergence, but it will be probably slower.

For the rest of the chapter, no difference will be made between preconditioners as they are minimal.

The results can be seen in the Table 1.2, where in terms of speed, keeping the number of 2LM interfaces at a minimum will improve the convergence rate. Given that this is not completely clear for every case, the extension of 2LM interfaces to cover the difficult interfaces will give us a more robust method.

In the following part and to achieve better performance, when using the Hybrid-FETI method we will mark as 2LM the edges that connect different materials and all the rest will be treated as 1LM.

With the default parameters for the Hybrid method, we can now perform a

Number of subdomains	Regular marking	Extra interface marking
64	18	79
125	20	76
250	21	64

Table 1.2 – Convergence of different marking Hybrid-FETI (Number of iterations)

Number of subdomains	FETI-1LM	Unsymmetric FETI-1LM	FETI-2LM	Hybrid-FETI
64	13	15	151	14
125	14	14	283	15
250	13	17	715	15

Table 1.3 – Convergence of the different methods (Number of iterations)

comparison of it against the two basis methods. In general, the problem we are solving in this section cause no performance impact to the FETI-1LM with the basic Dirichlet preconditioner. However, what we hope to obtain in this case is an improvement in comparison to the behaviour of the FETI-2LM. The idea is to show later some examples where we only have a good convergence for the FETI-2LM method, in which we will also hope to be a better alternative.

We modify the configuration of the previous case to solve something closer to the type of problems where we want to apply this method. Hence, only a different block is considered, located in the center of the structure, with different parameters of value

$$\nu_1 = 10^5, \text{ center block}$$
$$\nu_2 = 10^0, \text{ otherwise}$$

With these considerations in Table 1.3 we can see the results in term of iterations for the different methods.

As expected, the symmetric FETI-1LM is the most suited method for this

type of problem. However we wanted to show the unsymmetric version of this method because the problem solved when working with contact models (one of our primary objectives) is non-symmetric. In this case, the performance is comparable to the one of the Hybrid-FETI. As for the FETI-2LM, the inferior performance is explained by the type of divisions used in this particular case. In FETI-2LM the information is transmitted neighbor by neighbor, so for it to cross the whole structure, we need as many iterations as the number of subdomains before achieving convergence.

Even if the FETI-2LM is at it worst, this case represents a good example of the problems we may find in applications, with two blocks separated by a small part with really bad conditioning (contact, non-conforming meshes, etc.). Here the performance of the Hybrid method outperforms that of FETI-2LM, meaning that we have improved the speed of the more robust method, which will be our next goal when we try to solve problems where the convergence for the FETI-1LM method is not assured.

1.4.2 Contact Problem

Our next test case will try to expose the advantages of this new method. To achieve this, we will present and solve a contact problem.

Contact problems are often found in the structural engineering context, particularly when analyzing the assembly of different substructures. They are characterized by a non-penetration condition in an active area of contact which is not known a priori; this condition is modeled as a constraint of a minimization problem. For these reasons, these problems may lead to stiff non-linear system of equations.

Several simplifications to the general contact models can be done to reduce the computational complexity of the solvers needed. In this line, we can name a still large class of contact problems that are characterized by the fact that one of the contacting bodies is much more stiffer than the other. Also, considering that the displacement of the contact body is constrained in only one direction, we then have the so-called unilateral contact problems. We also assume that the displacements are small and the deformation is linearly elastic (this is for

presentation purposes, as in practice a large displacements model will be used). The final assumption to do is not to consider the effects of friction, which will simplify the contact constraints that we will establish later.

We start the formulation of this simplified contact model by recalling the 2D linear elasticity problem in a standard finite element framework. The equations that model this problem can be written as

$$\begin{aligned} -div(\sigma(\mathbf{u})) &= \mathbf{f}, \text{ in } \Omega \\ \mathbf{u} &= \mathbf{0} \text{ in } \partial\Omega_D \\ \sigma(\mathbf{u}) \cdot \mathbf{n} &= 0 \text{ in } \partial\Omega \setminus \partial\Omega_D \end{aligned}$$

where

$$\begin{aligned} \sigma_{ij}(\mathbf{u}) &= 2\mu\varepsilon_{ij}(\mathbf{u}) + \lambda\delta_{ij}div(\mathbf{u}) \\ \varepsilon_{ij}(\mathbf{u}) &= \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \\ \mu &= \frac{E}{2(1+\nu)} \\ \lambda &= \frac{E\nu}{(1+\nu)(1-2\nu)} \end{aligned}$$

in this context μ and λ are the Lamé coefficients, E and ν are the Young's Modulus and Poisson ratio respectively. The different materials are defined by their values of E and ν. A variation of these values in a generated structure will create the heterogeneities along or across the interfaces.

This model corresponds to the static linear elasticity, but for the next part, the displacements and several other components of the model will have a temporal dependency that we will not note, as we will just be testing the Hybrid method to solve this standard static model in a fixed final time T.

Let us present one of the classic mathematical formulations of a contact problem. We start as usual with a domain Ω that denotes the initial state of a linear elastic body; $\partial\Omega_C$ will be the part of its boundary that is a potential area of contact, unknown a priori; n_C is the outward normal to $\partial\Omega_C$ at a point M; $\partial\Omega'_C$ is the potential area of contact with $\partial\Omega_C$ but for the other linear elastic

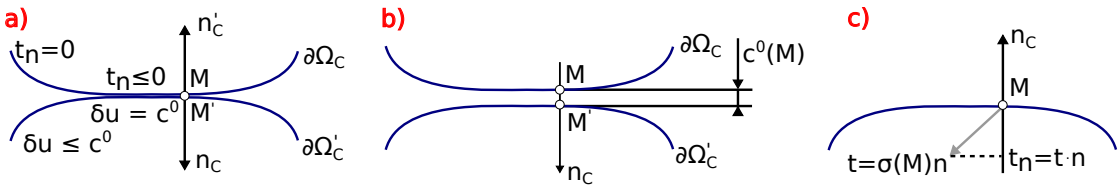

Figure 1.14 – a) Contact problem. b) Initial gap. c) Contact pressure.

body, and $n_C' = -n_C$ is the outward normal to $\partial\Omega_C'$ at the point M' facing point M (see the left image in Figure 1.14).

Let u denote the displacement field in one direction. The discontinuity of u in this normal direction at a point $M \in \partial\Omega_C$ can be written as

$$\begin{aligned}\delta u(M) &= u(M)\cdot n_C + u(M')\cdot n_C' = u(M)\cdot n_C - u(M')\cdot n_C \\ &= (u(M) - u(M'))\cdot n_C\end{aligned}$$

we write the non-penetration condition

$$\delta u(M) - c(M)^0 \leq 0$$

where $c(M)^0$ denotes the initial gap, that we assume small, at the point M in Ω (see center image of Figure 1.14).

Let $t = \sigma(M)n$ be the traction vector at point M, and $t_n = t\cdot n$ the normal component of t. If $\partial\Omega_C$ and $\partial\Omega_C'$ are in contact, the pressure at M is positive, which implies that $t_n \leq 0$ (see right image in Figure 1.14).

As already said, we assume unilateral conditions for the contact problem, which leads to the $Signorini - Fichera$ restrictions, defined by:

- If $\delta u(M) - c(M)^0 = 0$ then $t_n \leq 0$ and the contact is said to be active.
- If $\delta u(M) - c(M)^0 < 0$ then $t_n = 0$ and the contact is said to be inactive.

These relations can be rewritten equivalently as conditions on $\partial\Omega_C$, so for the unilateral frictionless contact problem, we have

$$
\begin{aligned}
t_n &\leq 0 \\
\delta u(M) - c(M)^0 &\leq 0 \\
t_n \cdot (\delta u(M) - c(M)^0) &= 0
\end{aligned}
$$

In a finite element framework, we need to state this contact model in a different way. Hence, we present now one of the variational formulations of this problem. In this case we will formulate the previous model as a minimization problem with variational inequalities.

We define the following spaces

$$V = \{\mathbf{v} \in (H^1(\Omega))^2 : \mathbf{v}|_{\partial\Omega_D} = \mathbf{0}\}$$

and

$$U = \{\mathbf{v} \in V : \mathbf{v} \cdot n - c(M)^0 \leq 0 \text{ on } \partial\Omega_C\}$$

We define then, the bilinear and linear forms

$$
\begin{aligned}
a(\mathbf{u}, \mathbf{v}) &:= 2 \int_\Omega \mu\varepsilon(\mathbf{u}) : \varepsilon(\mathbf{v}) dx + \int_\Omega \lambda(\nabla \cdot \mathbf{u})(\nabla \cdot \mathbf{v}) dx \\
b(\mathbf{v}) &:= \int_\Omega \langle \mathbf{f}, \mathbf{v} \rangle dx
\end{aligned}
$$

where

$$\langle \mathbf{f}, \mathbf{v} \rangle = \sum_{i=1}^{2} f_i v_i$$

and finally

$$\mathbf{F}(\mathbf{v}) = \frac{1}{2} a(\mathbf{v}, \mathbf{v}) - b(\mathbf{v})$$

then, for a given force $\mathbf{f} \in V'$ the variational formulation of the contact problem is

Find $\mathbf{u} \in U$ *such that*

$$a(\mathbf{u}, \mathbf{v} - \mathbf{u}) \geq f(\mathbf{v} - \mathbf{u}), \quad \forall \mathbf{v} \in U$$

or

$$\mathbf{F}(\mathbf{u}) \leq \mathbf{F}(\mathbf{v}) \quad \forall \mathbf{v} \in U$$

With this presentation, a minimization method can be applied to solve this constrained problem.

For the solution of contact problems, we can not forget that two types of non-linearities exist, the geometrical ones and in the materials. In the geometric non-linearities we encounter the ones derived from large displacements, deformation and contact constraints. An iterative incremental solution procedure is used then to obtain a non-linear solution as a series of linear ones.

Discretization of this problem via finite elements, plus the *Penalty method* or the *Lagrange multiplier method* [82] to impose the contact condition is one of the forms of solving this contact problem. However, different methods can be constructed starting from this. In this context, different FETI-like type of methods have been developed to solve this non-linear problem [4], [25], [23]. They are all based on the basic linear FETI method.

To state our comparative test case, we will use some results of a particular contact solver thanks to Alexandros Markopoulos [20]. So from now on, we will only consider a FETI-like with the penalty method to solve the contact problem. With this approach, there is a better treatment of the contact constraint with no extra variables, but at the cost of ill-conditioned problems that we will try to overcome.

For a contact problem, after the solver converges, we obtain a final active area of contact. With this final configuration we can "lock" the substructures, just by considering both contact bodies as a single conforming structure. This characteristic comes from the fact that a node-to-segment discretization is done in the contact area, which also treats with the large displacements in the model [80]. This type of discretization, along with the penalty method, implies that, virtual contact elements are added on one of the contact bodies. These elements, from a mechanical point of view, represent the elements that connect this body to an imaginary string, that when the contact is active, should have a very high stiffness, in order to impose the non-penetration condition.

In the first iterations of the non-linear solver, the penalty parameter is not necessarily very high and non-physical penetration may occur. However, when approaching to the computation of the final loads, we must increase the value of this parameter, to nullify the penetration and to have a correct imposition of the contact condition. This treatment produces the very ill-conditioning of the local stiffness matrices when the contact area is active.

Is for this reason that, unlike FETI-1LM, the FETI-2LM is a more suited method to solve this kind of problem. The difference is that in FETI-2LM the operator incorporates the augmentation or regularization matrices that come from the generalized Robin condition imposed in the interfaces. Added to the local stiffness problems, these matrices are usually taken as approximations of the Schur complement of the neighbor subdomain. Therefore, and even if the stiffness between neighbors subdomains can present as much as orders of magnitude of difference (this happens when the penalization parameter is acting in contact), they will be exchanged and taken into account by its neighbor.

To see this regularization, we just need to recall the definition of the FETI-2LM operator for a two subdomain case

$$\begin{bmatrix} I & I-(A_b^{(1)}+A_b^{(2)})(S_{bb}^{(2)}+A_b^{(2)})^{-1} \\ I-(A_b^{(2)}+A_b^{(1)})(S_{bb}^{(1)}+A_b^{(1)})^{-1} & I \end{bmatrix}$$

where we take $A_b^{(1)} \approx S_{bb}^{(2)}$ and $A_b^{(2)} \approx S_{bb}^{(1)}$, in order to have an operator less affected by the ill-conditioning produced by the penalization method.

After the computation of the final contact area, we can perform a standard stress analysis of that configuration. We consider the computed external loads produced by the interaction of both structures, including the penalization in the local stiffness matrices that share an interface with actual contact. In this static problem, we can compare the two standard FETI methods against the new Hybrid solver.

In the first problem to solve, we have a block on top of a semi-circle, see Figure 1.15. The top block is the stiffer body, and the bottom one is assumed to have large displacements. The values for their respective Young's modulus and

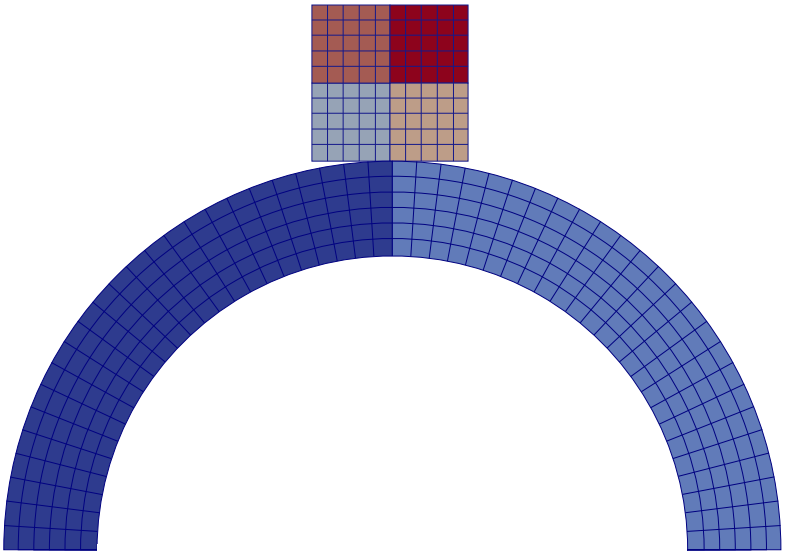

Figure 1.15 – First example, initial configuration and subdomain division.

Poisson ratio are

$$E_{top} = 10000, \quad \nu_{top} = 0.4$$
$$E_{bottom} = 6, \quad \nu_{bottom} = 0.3$$

The entire domain is divided into six subdomains, two for the bottom structure and four for the top one, then a vertical gravity-like force is applied, i.e., $f = (0, -10)$. The Dirichlet condition is used in the base of the bottom structure, and the two top corners of the block are also fixed, this is to give an initial stability.

Then, in Figure 1.16 we can see the solution after applying the contact solver. In here, we can also see the computed active contact zone (the bottom of the square block) in which the ill-conditioning is concentrated. The forces needed to obtain such a bending are also given by the solver.

One of the main issues shown in the previous test case was the definition of the 1LM or 2LM interfaces. When solving contact problems, we have the advantage of working with two separate and, a priori, known structures. This allows recognizing each subdomain as part of one or other body, which at the same time automatizes the process of marking the different interfaces in the

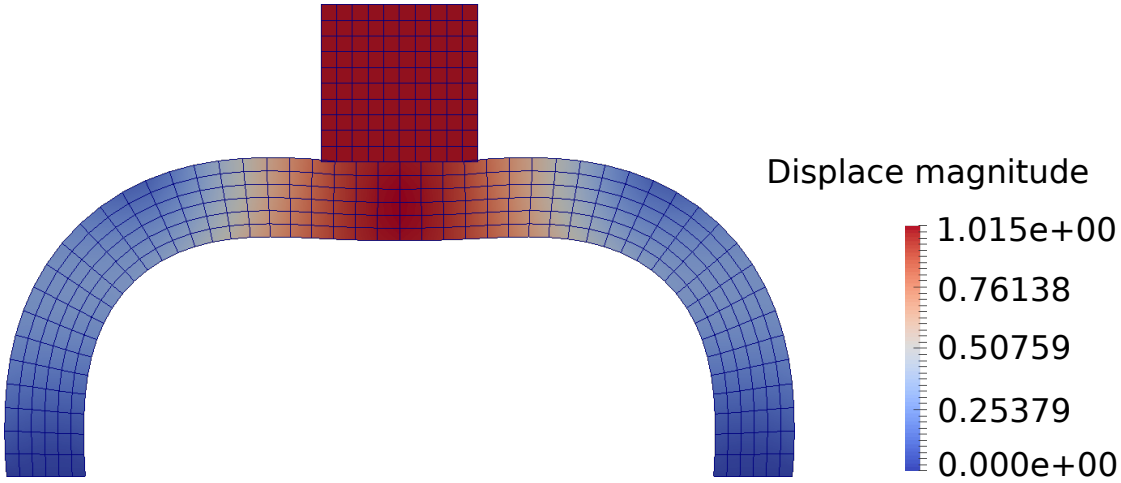

Figure 1.16 – First example, final configuration (solution).

following way

$$\begin{cases} Mark\ \Gamma^{(ij)}\ as\ 1LM & if\ \Omega^{(i)}, \Omega^{(j)}\ are\ in\ the\ same\ body \\ Mark\ \Gamma^{(ij)}\ as\ 2LM & Otherwise \end{cases}$$

Considering this process, plus the Robin or Dirichlet preconditioner; also, with a stopping criterion of 10^{-6} for the error. Choosing the patch size as 3 and the path depth as 5 for the Schur approximation matrices in the 2LM interfaces. Then, we can launch our tests for the final configuration of the contact problem, see results in Table 1.4.

The following final two cases of this chapter, model a physically more stable structure, where the same top block is now over a square bottom structure. This time we have augmented the number of subdomains to emphasize the advantages of the Hybrid method.

The top and first structure is divided into 16 subdomains. The bottom and second is divided into 9 subdomains for the first case and 25 subdomains for the second. Dirichlet conditions are imposed in the base of the second structure, and the top block is entirely free, as the contact zone can offer enough initial stability, see Figure 1.17.

Again, the contact solver gives us the final contact interfaces plus the forces

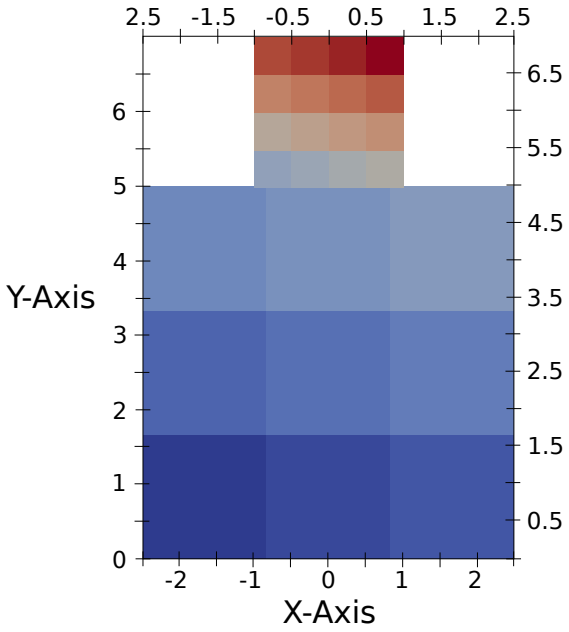

Figure 1.17 – Second example, initial configuration.

Subdomains	Global equations	FETI-1LM	FETI-2LM	Hybrid-FETI (Robin)	Hybrid-FETI (Dirichlet)
6	956	X	29	34	34
25	31954	61	119	50	51
41	5540	X	111	69	74

Table 1.4 – Convergence of the three contact examples for the different FETI methods (Number of iterations).

needed. The different methods are tested to compare the convergence rate. In the Table 1.4 we can see the number of iterations in each method for different problem sizes.

In the results, we can see that for both cases, the number of iterations for the Hybrid method is smaller than for the other two methods. Compared to the FETI-1LM we have gained in robustness since we only see convergence in the much more stable structure that is the second case. But even when there is convergence, we have improved its ratio, showing a better general performance for this type of problems. If we compare against the FETI-2LM method, we can see that in the biggest case the number of iterations used by the Hybrid method

is less than half of the ones needed initially and at the same time we manage to keep the robustness given by this method.

Finally, we point out that when the number of subdomains augments, we only see a small increase in the total iterations, in accord with the results shown in the first example of the two material bar.

1.5 Conclusion

The results obtained by the Hybrid-FETI method show us a positive first approach for the development of a general FETI method that tries to keep the good performance of the basic FETI-1LM method and the robustness of FETI-2LM. Its advantages are clearer in the context of solving the ill-conditioned problems arising from the simplest contact models. In these cases, the benefits regarding easier implementation and improved convergence shown in the numerical results give us a very good start in search of general better algorithms.

The next steps for this method are to perform tests in bigger and less academic configurations, but since our preliminary results show an improvement, we can only hope the same for the more complex cases. Also, a different type of problems can be tested, which are the class of non-conforming problems. In here the presence of mortar elements can affect the conditioning of the local matrices forcing the use of FETI-2LM in the complete structure [65]. However, in practice, only a small part of the problem needs such treatment, making our method also appropriate in this context.

Chapter 2

Block FETI methods

Different improvements to make more robust and efficient FETI methods have been developed in recent years, but for some harder engineering problems the performance is still very poor. To face these challenging problems, the Simultaneous-FETI or S-FETI method was developed, first in [60] for a two subdomain case, then generalized for arbitrary configurations in [38]. The main characteristic of S-FETI is the generation of more than one search direction at each step of the Conjugate Gradient method, leading to a more efficient and in general more robust method.

Within this new method, and as usual in the practical use of FETI methods, a full reorthogonalization is mandatory, forcing the storage of all the new search directions. Depending on the problem, specially when the number of subdomains increases, the total number of directions to store can be a drawback due to memory limitations, making the S-FETI method impractical for this type of problems.

In this chapter, we treat the problem of memory limitation of the S-FETI method. We use the sparse properties of the block of vectors from which we construct the search directions, to reduce the total memory allocated by the method. The strategy will be to rebuild at every iteration the search directions using this sparse blocks and some "small" coefficient matrices.

The introduction of these reconstruction steps will add computations to every processor, that will be more expensive. However, this cost is small compared to

the time spent in the communication of the different processes, so the total time used can be as good as the original method. The simplest of the implementations of this recursive reconstruction will not be enough to be near the performance of the regular S-FETI, but it will show a first look of the advantages in the storage and the maximal precision expected. Later we will show some optimizations based on the parallel properties of the code that will improve the computation time, making a more efficient algorithm.

This chapter starts by showing the basics of preconditioning in FETI; then we show how the ideas in the preconditioning steps lead to the formulation of the first S-FETI method with two subdomains. We continue by presenting the formal general method, including its implementation and cost. Then we show in detail how to sort search directions in the presence of linear dependency between them; we also add a second strategy to achieve this. We continue with the new storage, reconstruction of the search directions and the optimizations in the code. Finally we show some numerical results to compare the different algorithms.

2.1 Introduction and preliminarities

The development of the S-FETI method began in [60] when searching for improvements to the Dirichlet preconditioner in FETI. For this reason, in this section, we will follow the same path from the basics of the preconditioner that will lead to the first version of S-FETI.

2.1.1 Dirichlet preconditioner for two subdomains

We note that the following parts, up to the definition of the S-FETI method, are heavily based on the works of Daniel J. Rixen [60], [61], [62], and can be considered as classic results. However, we think that is necessary to recall them, to fully understand how the S-FETI method works. We will start by revisiting the Dirichlet preconditioner at its basic, for a two subdomain case. The notation, in this part, will be analogous to the previous chapter.

The objective of a preconditioner for the PCPG Algorithm 5 in FETI is to build in every iteration p a correction to the Lagrange multipliers. This rectification is based on an interface compatibility error, i.e., a jump denoted as g (we drop the notation g^p for simplicity.). In the case of an elasticity problem this is a jump in the displacement, but depending on the problem this may change.

Let us consider the two subdomain case as in Figure 1.2 from section 1.1.1. In here, the associated unknowns are

$$x^{(s)} = \begin{bmatrix} x_i^{(s)} \\ x_b^{(s)} \end{bmatrix}, \qquad s = 1, 2$$

the subscripts i, b represent interior and interface nodes respectively. The jump is written as

$$g = B^{(1)} x_b^{(1)} + B^{(2)} x_b^{(2)}$$

with B the signed boolean operator that defined this continuity condition as in Equation (1.29). Then, we can build a continuous interface approximate solution by averaging the already computed $x^{(s)}$

$$\begin{aligned} \hat{x}_b^{(1)} &= \frac{1}{2}\left(x_b^{(1)} - B^{(1)^T} B^{(2)} x_b^{(2)} \right) \\ \hat{x}_b^{(2)} &= \frac{1}{2}\left(x_b^{(2)} - B^{(2)^T} B^{(1)} x_b^{(1)} \right) \end{aligned}$$

The operator $B^{(s)^T} B^{(q)}$ is the correspondence between the numbering of the interface nodes in $\Omega^{(s)}$ and $\Omega^{(q)}$. The interface approximate solution corrections are

$$\begin{aligned} \delta x_b^{(1)} &= \hat{x}_b^{(1)} - x_b^{(1)} = -\frac{1}{2} B^{(1)^T} g \\ \delta x_b^{(2)} &= \hat{x}_b^{(2)} - x_b^{(2)} = -\frac{1}{2} B^{(2)^T} g \end{aligned} \tag{2.1}$$

If we modify the inner nodes to satisfy the internal equilibrium (the first equation in the local problem (1.28)), then we have the correction

$$\delta x_i^{(s)} = -K_{ii}^{(s)^{-1}} K_{ib}^{(s)} \delta x_b^{(s)}$$

Multiplying by the local matrix, the interface response corrections on the interface Γ are computed by

$$\delta f_b^{(s)} = S_{bb}^{(s)} \delta x_b^{(s)} \tag{2.2}$$

In general $\delta f_b^{(1)} \neq \delta f_b^{(2)}$ so the interface response is not uniquely defined. So again, an averaging is done as previously with the approximate solution; then the interface response correction will be chosen as

$$z = -\frac{1}{2}\left(B^{(1)} \delta f_b^{(1)} + B^{(2)} \delta f_b^{(2)}\right)$$

This corresponds to a new search direction in the CG algorithm. If we look at this definition, using (2.1) and (2.2), we can also write it as

$$z = \left(\frac{1}{4}\sum_{s=1}^{2} B^{(s)} S_{bb}^{(s)} B^{(s)^T}\right) g$$

which is equivalent to the definition of the Dirichlet preconditioner of subsection 1.1.3 with the scaling matrices in consideration. If the two matrices $S_{bb}^{(s)}$ are the same, for example in some symmetric splitting of a simple square, then the preconditioner is the exact inverse of the operator FETI. If not the case, only locally we have the exact inverse of this operator.

The difference between this Dirichlet preconditioner and its basic non-scaled form lies in what is defined as a *Consistent Preconditioner*. In the next part, we will give its definition and develop this subject in order to generalize the preconditioner to arbitrary partitions and heterogeneous problems.

2.1.2 Consistent Preconditioners

Preliminaries

We begin by recalling the signed boolean matrices needed to formulate the problem in a general domain division. For an arbitrary partition into $N_s \geq 2$ subdomains, we define

$$\Gamma^{(s)} = \partial\Omega^{(s)} \backslash \partial\Omega$$

this is the interface boundary of $\Omega^{(s)}$ for $s = 1,\dots,N_s$; then we define the global interface as

$$\Gamma = \underset{s}{\cup}\Gamma^{(s)}$$

Next, we consider the signed boolean operators $B^{(s)}$ as the matrices that maps the nodes from the local interface $\Gamma^{(s)}$ into the global one Γ. The sign of the represented nodes is such that the opposite sign is in the position of the same global node on the neighbor subdomain.

The global interface will again be divided into interface edges Γ^j, defined as

$$\Gamma^j := \Gamma^{(sq)} = \partial\Omega^{(s)} \cap \partial\Omega^{(q)}, \qquad \forall s = 1,\dots,N_s,\ \ q \in neighbor\{s\}$$

The crosspoints are nodes shared by more than two edges. It follows that $B^{(s)} : \Gamma^{(s)} \to \Gamma$ can be partitioned as

$$B^{(s)} = \begin{bmatrix} B^{(s)}_{\Gamma^1} \\ B^{(s)}_{\Gamma^2} \\ \vdots \\ B^{(s)}_{\Gamma^{n^{(s)}}} \end{bmatrix} \tag{2.3}$$

with $n^{(s)}$ the number of neighbors of $\Omega^{(s)}$ and where $B^{(s)}_{\Gamma^j}$ is the restriction of $B^{(s)}$ to Γ^j. We also define the global assembly operator

$$\mathbf{B} = \left[B^{(1)} \cdots B^{(N_s)}\right]$$

Next, we define the multiplicity of a node in Γ^j as m_j, and for each node its values are

$$m_j = |neighbors| + 1$$

the multiplicity varies in each node of the edge, but for simplicity, we will denote as a single one for each Γ^j. In general, $m_j \leq 2$ and for a crosspoint $m_j > 2$. Since one Lagrange multiplier is used to glue any pair of nodes (d.o.f) in an edge Γ^j, there are exactly $(m_j - 1)$ Lagrange multipliers applied to each d.o.f in any edge.

From Equation (2.3) and knowing that there are nodes shared by more than

two subdomains, we can see that

$$B_{\Gamma^j}^{(s)^T} B_{\Gamma^j}^{(s)} = (m_j - 1)I$$

then for each subdomain, we have that

$$\begin{aligned} B^{(s)^T} B^{(s)} &= \begin{bmatrix} B_{\Gamma^1}^{(s)^T} & \cdots & B_{\Gamma^{n^{(s)}}}^{(s)^T} \end{bmatrix} \begin{bmatrix} B_{\Gamma^1}^{(s)} \\ \cdots \\ B_{\Gamma^{n^{(s)}}}^{(s)} \end{bmatrix} \\ &= B_{\Gamma^1}^{(s)^T} B_{\Gamma^1}^{(s)} + \cdots + B_{\Gamma^{n^{(s)}}}^{(s)^T} B_{\Gamma^{n^{(s)}}}^{(s)} \\ &= (m_1 - 1)I + \cdots + (m_{n^{(s)}} - 1)I \\ &= diag(m_j - 1) \end{aligned}$$

Finally, we can write

$$B^{(s)^T} B^{(s)} + I = diag(m_j) \tag{2.4}$$

where $diag(m_j)$ is the diagonal matrix with the multiplicity of the nodes in the edges $\Gamma^j \subseteq \Gamma^{(s)}$.

Remark: This form of gluing connecting d.o.f introduces redundancies in the compatibility constraints at the crosspoints (continuity). However, we will see that this redundancy is essential for an efficient preconditioner.

Consistent Preconditioner

With previous definitions, we can analyze the preconditioner in terms of a general partition of the domain in $N_s > 2$ subdomains.

Recalling the physical interpretation of the projected gradient in the FETI-1LM method, we know it is a jump of the displacement field across the subdomain interface boundaries

$$g = \sum_{s=1}^{N_s} B^{(s)} x_b^{(s)}$$

So, from a mechanical point of view, the objective of a preconditioner $\tilde{F}^{-1}$ based

on local problems is to generate a correction of the Lagrange multipliers z and its corresponding local interface forces $B^{(s)^T} z = B^{(s)^T}[\tilde{F}^{-1} g]$. We want to reduce the jump g as much as possible.

The basic preconditioning, either Dirichlet or Lumped, applied to the projected gradient g is defined in general terms as

$$z = \tilde{F}^{-1} g = \sum_{s=1}^{N_s} B^{(s)} \left(S_{bb}^{(s)} \text{ or } K_{bb}^{(s)} \right) B^{(s)^T} g$$

this operator is built in a three-step procedure, starting with g

1 We define the approximate solution corrections $\delta x_b^{(s)}$ and these are imposed in the local interfaces $\Gamma^{(s)}$ as follows

$$\delta x_b^{(s)} = B^{(s)^T} g$$

this means that for every d.o.f. we impose a correction equal to the sum of the jumps with every neighboring d.o.f.

2 Next, the discrete Dirichlet-to-Neumann operator on the interface nodes $\delta f_b^{(s)}$ is evaluated

$$\delta f_b^{(s)} = \left(S_{bb}^{(s)} \right) \delta x_b^{(s)} \tag{2.5}$$

the difference between the Dirichlet and the Lumped preconditioners lies in this step. To construct the Lumped operator, we replace previous computation by

$$\delta f_b^{(s)} = \left(K_{bb}^{(s)} \right) \delta x_b^{(s)}$$

and we note that the Lumped matrix $K_{bb}^{(s)}$ assumes that the internal nodes are fixed.

3 Finally, the jump of internal nodal responses $\delta f_b^{(s)}$ is computed to obtain the correction z of the Lagrange multiplier

$$z = \sum_{s=1}^{N_s} B^{(s)} \delta f_b^{(s)}$$

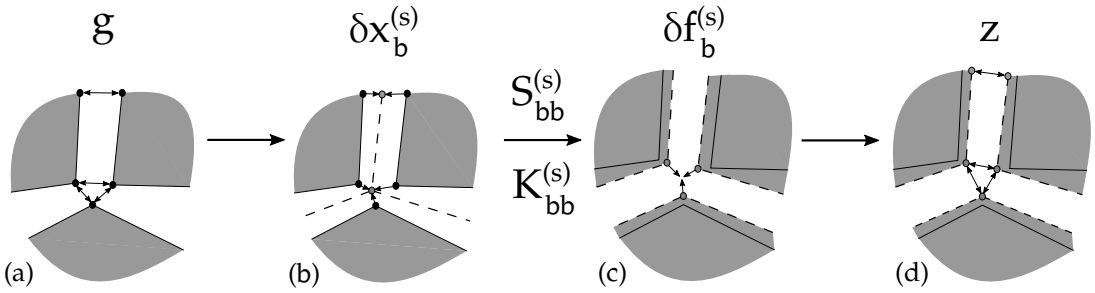

Figure 2.1 – Preconditioner construction

We can now define what a *Consistent preconditioner* is. It corresponds to a preconditioner built following the previous three steps, but where the approximate solution corrections $\delta x_b^{(s)}$ fulfill the compatibility condition; meaning that these corrections are continuous across the interface. Also, the Lagrange multiplier corrections z are chosen based on near-equilibrium concepts that will be explained in the next paragraphs.

The first condition is fulfilled if we use an averaging process as the one shown for the two subdomain case. This idea of imposing the same value (e.g., the average) to every node on the interface will also be valid for the general case, i.e., considering crosspoint.

Graphically, the three steps construction are depicted in Figure 2.1

Figure 2.1 (a) and (b) show how the increments $\delta x_b^{(s)}$ are chosen so that the corrected interface approximate solution $\hat{x}_b^{(s)}$ satisfies the interface compatibility. Following Equation (2.5) we compute the nodal interface responses required to maintain the increment $\delta x_b^{(s)}$. This mapping $\delta f_b^{(s)}$ does not satisfy the interface equilibrium unless the corrected solutions $\hat{x}_b^{(s)}$ are the exact final solutions. In Figure 2.1 (d) we compute the Lagrange multiplier corrections z. Interface Lagrange multipliers are naturally self-equilibrated in the sense that they result in interface values ${B^{(s)}}^T z$ that are in equilibrium. Hence, it is in general impossible to define z such that ${B^{(s)}}^T z$ restores the interface response corrections $\delta f_b^{(s)}$. Nevertheless, we require that z to be constructed in such a way that as $\delta f_b^{(s)}$ approaches equilibrium, ${B^{(s)}}^T z$ is exactly $\delta f_b^{(s)}$.

Let us consider the global interface response correction $\widehat{\delta f}_b$ as the vector

defined in the interface, such that

$$\delta f_b^{(s)} = \widehat{\delta f}_b|_{\Gamma^{(s)}}$$

We can say now that a *Consistent Preconditioner* is the one that has

1. Consistent approximate solution increments $\delta x_b^{(s)}$, i.e.

$$\sum_{s=1}^{N_s} B^{(s)} \hat{x}_b^{(s)} = \sum_{s=1}^{N_s} B^{(s)} \left(x_b^{(s)} + \delta x_b^{(s)} \right) = 0$$

2. Consistent Lagrange multiplier corrections z, i.e.
   if

$$\widehat{\delta f}_b \in Im(\mathbf{B}) \tag{2.6}$$

   then, for each $s = 1, \ldots, N_s$

$$B^{(s)^T} z = \delta f_b^{(s)}$$

Equation (2.6) is the interface equilibrium as it express that the sum of this boundary interactions $\delta f_b^{(s)}$ acting on an interface d.o.f. are zero, i.e

$$\forall i \in \Gamma, \qquad \sum_{s=1}^{N_s} \delta f_i^{(s)} = 0$$

At a subdomain level, the condition (2.6) can also be written

$$\delta f_b^{(s)} - B^{(s)^T} \sum_{\substack{r=1 \\ r \neq s}}^{N_s} B^{(r)} \delta f_b^{(r)} = 0 \tag{2.7}$$

With this definition, it can be expected that preconditioners that fulfill it are better than the ones that do not, see [61] for details. In this sense, we will explain the modifications done to the basic Dirichlet preconditioner to make it a consistent one; leading to a new structure that will be generalized to have a preconditioner also suited for heterogeneous problems (across the interface).

Remark:

1. The basic Dirichlet or Lumped preconditioners are not consistent. If we look at the definition of $\delta x_b^{(s)}$, we have that

$$\sum_{s=1}^{N_s} B^{(s)} \left(x_b^{(s)} + \delta x_b^{(s)} \right) = g + \sum_{s=1}^{N_s} B^{(s)} B^{(s)^T} g \neq 0$$

which violates the compatibility condition for the solution correction.

Furthermore, if we take the construction

$$z = \sum_{s=1}^{N_s} B^{(s)} \delta f_b^{(s)}$$

and we assume that Equation (Equation 2.7) holds, then we have

$$\begin{aligned} B^{(s)^T} z &= B^{(s)^T} \sum_{r=1}^{N_s} B^{(r)} \delta f_b^{(r)} \\ &= B^{(s)^T} B^{(s)} \delta f_b^{(s)} + B^{(s)^T} \sum_{\substack{r=1 \\ r \neq s}}^{N_s} B^{(r)} \delta f_b^{(r)} \\ &= \left(B^{(s)^T} B^{(s)} + I \right) \delta f_b^{(s)} \end{aligned}$$

and using Equation (2.4)

$$B^{(s)^T} z = diag(m_j) \delta f_b^{(s)} \neq \delta f_b^{(s)}$$

2. The natural self-equilibrium of the Lagrange multipliers can be expressed replacing $\delta f_b^{(s)}$ by $B^{(s)^T} z$ in Equation (2.7)

$$B^{(s)^T} z - B^{(s)^T} \sum_{\substack{r=1 \\ r \neq s}}^{N_s} B^{(r)} B^{(r)^T} z = 0$$

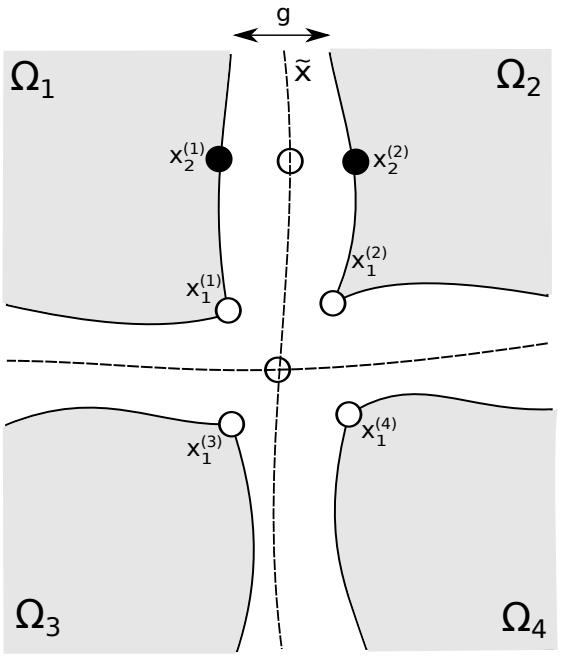

Figure 2.2 – 4 Subdomain problem

from here, we have

$$B^{(s)^T} = B^{(s)^T} \sum_{\substack{r=1 \\ r \neq s}}^{N_s} B^{(r)} B^{(r)^T}$$

this is a direct consequence of the fact that the compatibility conditions are defined between any pair of connecting d.o.f.

Consistent preconditioner for homogeneous problems

For homogeneous problems, consistent preconditioners are constructed by extending the averaging scheme used in the two subdomain case, see Equation (subsection 2.1.1). For clarity, we will start showing it for a 4 subdomain case as in Figure 2.2 then generalizing it for arbitrary meshes.

We assume that all subdomains have similar stiffness matrices, i.e., are built for subdomains of similar materials, geometrical and discretization properties. Therefore, the construction of the compatible interface solution is done by

imposing an averaging as follows

$$\hat{x}_1^{(1)} = \hat{x}_1^{(2)} = \hat{x}_1^{(3)} = \hat{x}_1^{(4)} = \frac{x_1^{(1)} + x_1^{(2)} + x_1^{(3)} + x_1^{(4)}}{4}$$
$$\hat{x}_2^{(1)} = \hat{x}_2^{(2)} = \frac{x_2^{(1)} + x_2^{(2)}}{2}$$

which implies that the following corrections are consistent

$$\begin{aligned}
\delta x_1^{(1)} &= \hat{x}_1^{(1)} - x_1^{(1)} \\
&= \frac{\left(x_1^{(2)} - x_1^{(1)}\right) + \left(x_1^{(3)} - x_1^{(1)}\right) + \left(x_1^{(4)} - x_1^{(1)}\right)}{4} \\
\delta x_1^{(2)} &= \frac{\left(x_1^{(1)} - x_1^{(2)}\right) + \left(x_1^{(3)} - x_1^{(2)}\right) + \left(x_1^{(4)} - x_1^{(2)}\right)}{4} \\
&\vdots \\
\delta x_2^{(1)} &= \frac{\left(x_2^{(2)} - x_2^{(1)}\right)}{2} \\
\delta x_2^{(2)} &= \frac{\left(x_2^{(1)} - x_2^{(2)}\right)}{2}
\end{aligned}$$

If the interface nodal responses are computed as in (2.5), then, in general, they will not be in equilibrium. For instance if we see the node 2, we have that $\delta f_2^{(1)} \neq \delta f_2^{(2)}$. So the Lagrange multiplier correction is constructed by doing the same averaging between the results of the Dirichlet-to-Neumann mapping, as follows

$$z_7 = \frac{-\delta f_2^{(1)} + \delta f_2^{(2)}}{2} \tag{2.8}$$

The minus sign in here comes from the fact that these vectors have opposite directions. By convention, the entries of $B_{\Gamma^j}^{(s)}$ (Γ^j connecting subdomains s and q) are $+1$ if $s > q$ and -1 otherwise. We chose to do the same averaging to keep a symmetric preconditioner. Doing the analogous averaging for the crosspoint of

multiplicity 4, we have the corrections

$$\begin{aligned} z_1 &= \frac{-\delta f_1^{(1)} + \delta f_1^{(2)}}{4}, \qquad z_2 = \frac{-\delta f_1^{(2)} + \delta f_1^{(3)}}{4} \\ z_3 &= \frac{-\delta f_1^{(3)} + \delta f_1^{(4)}}{4}, \qquad z_4 = \frac{-\delta f_1^{(1)} + \delta f_1^{(4)}}{4} \\ z_5 &= \frac{-\delta f_1^{(1)} + \delta f_1^{(3)}}{4}, \qquad z_6 = \frac{-\delta f_1^{(2)} + \delta f_1^{(4)}}{4} \end{aligned} \tag{2.9}$$

and the evaluation of $z = \tilde{F}^{-1} g$ is complete.

We can check that this Lagrange multiplier correction is consistent, in fact if

$$\sum_{s=1}^{4} \delta f_1^{(s)} = 0 \text{ and } \sum_{s=1}^{2} \delta f_2^{(s)} = 0$$

then

$$\begin{aligned} B^{(1)^T} z &= \begin{bmatrix} -z_1 - z_4 - z_5 \\ -z_7 \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{4}\left(\delta f_1^{(1)} - \delta f_1^{(2)}\right) + \frac{1}{4}\left(\delta f_1^{(1)} - \delta f_1^{(4)}\right) + \frac{1}{4}\left(\delta f_1^{(1)} - \delta f_1^{(3)}\right) \\ \frac{\delta f_1^{(1)} - \delta f_1^{(1)}}{2} \end{bmatrix} \\ &= \begin{bmatrix} \delta f_1^{(1)} \\ \delta f_2^{(1)} \end{bmatrix} \end{aligned}$$

and the same happens analogously with the other subdomains. So, for the 4 subdomain case the averaging process shows the construction of a consistent preconditioner (Dirichlet or Lumped).

Let us show the extension of this averaging process to arbitrary meshes. Consider any edge Γ^i, then the compatible solutions $\hat{x}_{\Gamma^i}^{(s)}$ can be constructed by averaging all the nodes belonging to Γ^i

$$\hat{x}_{\Gamma^j}^{(s)} = \sum_{r:\Gamma^i \subseteq \{\Gamma \cap \Gamma^{(r)}\}} \frac{1}{m_i} x_{\Gamma^i}^{(r)}$$

where m_i is the multiplicity of nodes in the edge Γ^i. Then, we build the following consistent corrections

$$\begin{aligned}\delta x_{\Gamma^j}^{(s)} &= \hat{x}_{\Gamma^j}^{(s)} - x_{\Gamma^j}^{(s)} \\ &= \sum_{\substack{r:\Gamma^i \subseteq \{\Gamma \cap \Gamma^{(r)}\} \\ r \neq s}} \frac{1}{m_i}\left(x_{\Gamma^i}^{(r)} - x_{\Gamma^i}^{(s)}\right)\end{aligned}$$

and extended to the total subdomain interface, we have

$$\begin{aligned}\delta x_b^{(s)} &= -B^{(s)^T} diag\left(\frac{1}{m_j}\right) \sum_{r=1}^{N_s} B^{(r)} x_b^{(r)} \\ &= -B^{(s)^T} E^{(s)} g\end{aligned}$$

where $E^{(s)}$ is a diagonal matrix with values that correspond to the multiplicity of nodes in the edges that intersect $\Omega^{(s)}$. This matrix can be seen as a *scaling* matrix.

Now we compute the interface responses $\delta f_b^{(s)}$ associated with these corrections $\delta x_b^{(s)}$. We simply use (2.5) the same way as before. As for the consistent Lagrange multiplier corrections, we generalize the averaging shown in (2.8) and (2.9) to obtain

$$z = \tilde{F}^{-1} g = diag\left(\frac{1}{m_i}\right) \sum_{r=1}^{N_s} B^{(r)} \delta f_b^{(r)}$$

Again, let us check the consistency of this correction. We assume the interface equilibrium condition (2.6) and also (2.4), then we have

$$\begin{aligned}B^{(s)^T} z &= diag\left(\frac{1}{m_i}\right) \sum_{r=1}^{N_s} B^{(s)^T} B^{(r)} \delta f_b^{(r)} \\ &= diag\left(\frac{1}{m_i}\right) B^{(s)^T} B^{(s)} \delta f_b^{(s)} + diag\left(\frac{1}{m_i}\right) \sum_{\substack{r=1 \\ r \neq s}}^{N_s} B^{(s)^T} B^{(r)} \delta f_b^{(r)} \\ &= diag\left(\frac{1}{m_i}\right) \left(diag\left(m_i\right) - I\right) \delta f_b^{(s)} + diag\left(\frac{1}{m_i}\right) \delta f_b^{(s)} \\ &= \delta f_b^{(s)}\end{aligned}$$

from which we have the consistency of the Lagrange multiplier correction z.

With this, we can write the consistent version of the Dirichlet and Lumped preconditioners

$$D^{-1} = \sum_s E^{(s)} B^{(s)} S_{bb}^{(s)} B^{(s)^T} E^{(s)}$$

$$L^{-1} = \sum_s E^{(s)} B^{(s)} K_{bb}^{(s)} B^{(s)^T} E^{(s)}$$

Consistent preconditioner for heterogeneous problems

The previous scaling works in homogeneous problems where the stiffness matrices of neighboring subdomains are similar, and we can think that the compatible field is in the "middle" of them. In heterogeneous problem this is not the case, as we can presume that the compatible solution will be closer to the subdomain with a higher stiffness (from a mechanical point of view), making the previous scaling to degrade in terms of convergence for this kind of problems. To correct this behaviour, we will define a more general scaling, also consistent, that can acknowledge these differences.

We start again by explaining the idea in a 4 subdomain case, as in Figure 2.2. In this configuration, we define the more general compatible field as

$$\begin{aligned}
\hat{x}_1^{(1)} &= \hat{x}_1^{(2)} = \hat{x}_1^{(3)} = \hat{x}_1^{(4)} \\
&= \beta_1^{(1)} \hat{x}_1^{(1)} + \beta_1^{(2)} \hat{x}_1^{(2)} + \beta_1^{(3)} \hat{x}_1^{(3)} + \beta_1^{(4)} \hat{x}_1^{(4)} \\
\hat{x}_2^{(1)} &= \hat{x}_2^{(2)} \\
&= \beta_2^{(1)} \hat{x}_2^{(1)} + \beta_2^{(2)} \hat{x}_2^{(2)}
\end{aligned}$$

where the $\beta_k^{(s)}$ terms are smoothing or weighting coefficients defined in every interface d.o.f. k of subdomain $\Omega^{(s)}$. These coefficients must also be constrained by

$$\begin{aligned}
\beta_1^{(1)} + \beta_1^{(2)} + \beta_1^{(3)} + \beta_1^{(4)} &= 1 \\
\beta_2^{(1)} + \beta_2^{(2)} &= 1
\end{aligned}$$

this implies that when the solution before weighting is compatible, then the

corrections vanish. In this case, the consistent corrections are

$$\begin{aligned}
\delta x_1^{(1)} &= \hat{x}_1^{(1)} - x_1^{(1)} \\
&= \beta_1^{(2)}\left(x_1^{(2)} - x_1^{(1)}\right) + \beta_1^{(3)}\left(x_1^{(3)} - x_1^{(1)}\right) + \beta_1^{(4)}\left(x_1^{(4)} - x_1^{(1)}\right) \\
&\vdots \\
\delta x_2^{(1)} &= \beta_2^{(2)}\left(x_2^{(2)} - x_2^{(1)}\right) \\
\delta x_2^{(2)} &= \beta_2^{(1)}\left(x_2^{(1)} - x_2^{(2)}\right)
\end{aligned}$$

the interface responses are again computed using Equation (2.5), so the corrections for the Lagrange multipliers are build using the same weighting

$$\begin{aligned}
z_1 &= -\beta_1^{(2)}\delta f_1^{(1)} + \beta_1^{(1)}\delta f_1^{(2)}, \qquad z_2 = -\beta_1^{(3)}\delta f_1^{(2)} + \beta_1^{(2)}\delta f_1^{(3)} \\
z_3 &= -\beta_1^{(4)}\delta f_1^{(3)} + \beta_1^{(3)}\delta f_1^{(4)}, \qquad z_4 = -\beta_1^{(4)}\delta f_1^{(1)} + \beta_1^{(1)}\delta f_1^{(4)} \\
z_5 &= -\beta_1^{(3)}\delta f_1^{(1)} + \beta_1^{(1)}\delta f_1^{(3)}, \qquad z_6 = -\beta_1^{(4)}\delta f_1^{(2)} + \beta_1^{(2)}\delta f_1^{(4)} \\
z_7 &= -\beta_2^{(2)}\delta f_2^{(1)} + \beta_1^{(2)}\delta f_2^{(2)}
\end{aligned}$$

the previous corrections are in fact consistent. Let us consider that the condition (2.6) is fulfilled, then for the subdomain 1, we have
if

$$\sum_{s=1}^{4} \delta f_1^{(s)} = 0 \text{ and } \sum_{s=1}^{2} \delta f_2^{(s)} = 0$$

then

$$\begin{aligned}
B^{(1)^T} z &= \begin{bmatrix} -z_1 - z_4 - z_5 \\ -z_7 \end{bmatrix} \\
&= \begin{bmatrix} \left(\beta_1^{(2)} + \beta_1^{(4)} + \beta_1^{(3)}\right)\delta f_1^{(1)} - \beta_1^{(1)}\left(\delta f_1^{(2)} + \delta f_1^{(3)} + \delta f_1^{(4)}\right) \\ \beta_2^{(2)}\delta f_2^{(1)} - (1 - \beta_2^{(2)})\delta f_2^{(2)} \end{bmatrix} \\
&= \begin{bmatrix} \delta f_1^{(1)} \\ \delta f_2^{(1)} \end{bmatrix}
\end{aligned}$$

The same happens for the other subdomains.

This process defines a generalization of the weighting procedure for an arbitrary number of subdomains. The analysis is analogous to previous, but considering a more general scaling matrix $\beta^{(s)}$. It is defined as the diagonal matrix of weighting coefficients of the interface d.o.f. belonging to the neighbors of $\Omega^{(s)}$.

The choice of these coefficients can be made using a physical criterion. In the case of an elasticity problem (works for any elliptic PDE problem) we decouple all the interface d.o.f. assuming that each one of them is connected to a stiffness-free covering subdomain via a spring. In other words, we see the stiffness of each subdomain lumped to its interface. The lumped stiffness matrix is diagonal equal to $diag\left(K_{bb}^{(s)}\right)$; this weighting procedure is usually called *Superlumped*.

In general, the coefficients are computed as the ratio between the stiffness of an interface d.o.f. in some subdomain and the sum of all the stiffness of the connecting nodes to this d.o.f. in all neighboring subdomains. For example, for the 4 subdomain case, we have

$$\begin{aligned}
\beta_1^{(1)} &= \frac{k_{11}^{(1)}}{k_{11}^{(1)} + k_{11}^{(2)} + k_{11}^{(3)} + k_{11}^{(4)}} \\
\beta_1^{(2)} &= \frac{k_{11}^{(2)}}{k_{11}^{(1)} + k_{11}^{(2)} + k_{11}^{(3)} + k_{11}^{(4)}} \\
&\vdots \\
\beta_2^{(1)} &= \frac{k_{22}^{(1)}}{k_{22}^{(1)} + k_{22}^{(2)}} \\
\beta_2^{(2)} &= \frac{k_{22}^{(2)}}{k_{22}^{(1)} + k_{22}^{(2)}}
\end{aligned}$$

And in general, these coefficients are computed for every edge Γ^i by

$$\beta_{\Gamma^i}^{(s)} = diag\left(K_{\Gamma^i}^{(s)}\right)\left\{\sum_{r:\Gamma^j \subseteq \{\Gamma \cap \Gamma^{(r)}\}} diag\left(K_{\Gamma^j}^{(s)}\right)\right\}^{-1}$$

So finally, adding this *Superlumped* scaling to the preconditioners Dirichlet and Lumped respectively, they can be written as

$$D^{-1} = \sum_s \beta^{(s)} B^{(s)} S_{bb}^{(s)} B^{(s)^T} \beta^{(s)} = \sum_s \tilde{B}^{(s)} S_{bb}^{(s)} \tilde{B}^{(s)^T} \tag{2.10}$$

$$L^{-1} = \sum_s \beta^{(s)} B^{(s)} K_{bb}^{(s)} B^{(s)^T} \beta^{(s)} = \sum_s \tilde{B}^{(s)} K_{bb}^{(s)} \tilde{B}^{(s)^T} \tag{2.11}$$

with

$$\tilde{B}^{(s)} := \beta^{(s)} B^{(s)}$$

Remark: The implementation of this scaling is computationally efficient; it has practically no difference in time cost with the basic form. We can see this negligible extra cost summarized as

- One unique extra communication between processes used to share the stiffness values from neighboring d.o.f.
- Construction of diagonal matrices $\beta^{(s)}$.
- At every multiplication by the preconditioner, we add two extra computations by this local diagonal scaling, which are negligible in parallel computation.

2.1.3 Simultaneous FETI

As seen in the previous subsection, the Dirichlet or Lumped preconditioners are two correctors of the Lagrange multipliers imposed in the interface. They use an averaging or scaling on the interface to make this correction consistent. The averaging is necessary because the interface computed responses $\delta f_b^{(s)}$ to obtain a compatible solution are, in general, different in two connected d.o.f.

The idea is now to use these two different corrected responses independently, as search directions for the Conjugate Gradient method, as shown in [60].

We start again by showing this in a two-subdomain division. Let us consider the iteration p of the CG method applied to the FETI-1LM method (no projection is considered in this simple case). Then, we first solve the local Dirichlet problem,

consisting in enforcing on the interface the compatibility gap g^p and then we compute the boundary reaction which writes

$$\delta f^{(s),p} = S_{bb}^{(s)} \tilde{B}^{(s)^T} g^p, \qquad s = 1, 2$$

In the classical Dirichlet preconditioner, we evaluate and approximate corrections to the Lagrange multipliers by computing a weighted inter-subdomain vector based on $\delta f^{(s),p}$. Let us now consider this boundary condition on each side of the interface as a descent direction for the CG algorithm by writing the new update of the interface Lagrange multiplier λ

$$\lambda^{p+1} = \lambda^p + \rho^{(1),p} \tilde{B}^{(1)} \delta f^{(1),p} + \rho^{(2),p} \tilde{B}^{(2)} \delta f^{(2),p}$$

assuming now that, instead of using these boundary reactions directly, we use

$$\delta\lambda^{(1),p} = \tilde{B}^{(1)} \delta f^{(1),p} + \sum_{s=1}^{2} \sum_{l=0}^{p-1} \alpha^{(s),l} \delta\lambda^{(s),l}$$

$$\delta\lambda^{(2),p} = \tilde{B}^{(2)} \delta f^{(2),p} + \alpha^{(2),p} \delta f^{(1),p} + \sum_{s=1}^{2} \sum_{l=0}^{p-1} \gamma^{(s),l} \delta\lambda^{(s),l}$$

also, $\delta\lambda^{(2),p}$ is orthogonal to $\delta\lambda^{(1),p}$ and both directions are orthogonal to all previous directions

$$\delta\lambda^{(1),p^T} F \delta\lambda^{(2),p} = 0$$
$$\delta\lambda^{(s),p^T} F \delta\lambda^{(r),l} = 0, \qquad r, s = 1, 2 \quad l = 0, \ldots, p-1$$

Then, the new updates are

$$\begin{aligned} \lambda^{p+1} &= \lambda^p + \rho^{(1),p} \delta\lambda^{(1),p} + \rho^{(2),p} \delta\lambda^{(2),p} \\ g^{p+1} &= g^p + \rho^{(1),p} F \delta\lambda^{(1),p} + \rho^{(2),p} F \delta\lambda^{(2),p} \end{aligned} \tag{2.12}$$

the direction coefficient $\rho^{(s),p}$ can be determined by these new orthogonality

relations

$$(g^{p+1}\cdot\delta\lambda^{(1),p}) = (g^{p}\cdot\delta\lambda^{(1),p}) + \rho^{(1),p}(F\delta\lambda^{(1),p}\cdot\delta\lambda^{(1),p}) + \rho^{(2),p}(F\delta\lambda^{(2),p}\cdot\delta\lambda^{(1),p})$$
$$0 = (g^{p}\cdot\delta\lambda^{(1),p}) + \rho^{(1),p}(F\delta\lambda^{(1),p}\cdot\delta\lambda^{(1),p})$$

$$(g^{p+1}\cdot\delta\lambda^{(2),p}) = (g^{p}\cdot\delta\lambda^{(2),p}) + \rho^{(1),p}(F\delta\lambda^{(1),p}\cdot\delta\lambda^{(2),p}) + \rho^{(2),p}(F\delta\lambda^{(2),p}\cdot\delta\lambda^{(2),p})$$
$$0 = (g^{p}\cdot\delta\lambda^{(2),p}) + \rho^{(2),p}(F\delta\lambda^{(2),p}\cdot\delta\lambda^{(2),p})$$

which implies

$$\rho^{(s),p} = -\frac{(g^{p}\cdot\delta\lambda^{(s),p})}{(\delta\lambda^{(s),p}\cdot F\delta\lambda^{(s),p})}, \qquad s = 1,2$$

that correspond to the same as the regular CG method. With this, we can note that one of the fundamental features of this new algorithm is that even if at each iteration the minimization is done with respect to two descent directions, the cost of this is equivalent that of a normal CG direction. Actually, recalling that the directions come from a Dirichlet problem, we note that

$$\begin{aligned} F\tilde{B}^{(1)}\delta f^{(1),p} &= \left(\tilde{B}^{(1)}S_{bb}^{(1)^{-1}}\tilde{B}^{(1)^{T}} + \tilde{B}^{(2)}S_{bb}^{(2)^{-1}}\tilde{B}^{(2)^{T}}\right)\tilde{B}^{(1)}S_{bb}^{(1)}\tilde{B}^{(1)^{T}}g^{p} \\ &= g^{p} + \tilde{B}^{(2)}S_{bb}^{(2)^{-1}}\tilde{B}^{(2)^{T}}\tilde{B}^{(1)}\delta f^{(1),p} \end{aligned}$$

$$\begin{aligned} F\tilde{B}^{(2)}\delta f^{(2),p} &= \left(\tilde{B}^{(1)}S_{bb}^{(1)^{-1}}\tilde{B}^{(1)^{T}} + \tilde{B}^{(2)}S_{bb}^{(2)^{-1}}\tilde{B}^{(2)^{T}}\right)\tilde{B}^{(2)}S_{bb}^{(2)}\tilde{B}^{(2)^{T}}g^{p} \\ &= g^{p} + \tilde{B}^{(1)}S_{bb}^{(1)^{-1}}\tilde{B}^{(1)^{T}}\tilde{B}^{(2)}\delta f^{(2),p} \end{aligned}$$

And in this case, we see that applying the FETI operator to both descent directions only requires one Neumann solution per subdomain.

In the generalized version, we do not expect to have this same property of cost equivalency between methods, but we do expect a reduced number of multiplications by the FETI operator that will lead to a faster method in general.

We will now show the extension done in [38] to this new method for arbitrary meshes.

2.1.4 The algorithm

The idea is to exploit the additive structure of the preconditioner in FETI and generate several search directions, instead of one, in every step of the CG method. The basic S-FETI method creates one for each subdomain, but straightforwardly, this number can be two times the total number of local interfaces in the problem.

In this section, the notation and definitions come from the previous chapter, where the original FETI method is described. With this in mind, we can remember the construction of a search direction in the classical FETI method. We denote this direction $w \in \mathbb{R}^n$, n being the size of the interface, and therefore the size of the system to solve.

The consistent Dirichlet operator, defined in (2.10), is first applied to the global residual vector g

$$w = D^{-1}g = \left(\sum_s D^{(s)^{-1}}\right) g$$

where

$$D^{(s)^{-1}} = \tilde{B}^{(s)} S_{bb}^{(s)} \tilde{B}^{(s)^T}$$

then the vector w is orthogonalized with respect to previous search directions to generate the new one.

The idea of S-FETI is to improve the minimization process in CG by spanning a search space, not from the addition of local contributions, but from each of these terms separately, letting the process to choose the best combination. Although more costly, this optimal combination minimizes the residual in the space generated by

$$Z = \left[D^{(1)^{-1}} g, D^{(2)^{-1}} g, \ldots, D^{(N_s)^{-1}} g\right] \tag{2.13}$$

where N_s is the number of subdomains.

Each one of these columns is projected and orthogonalized to give the block W of N_s columns, where each column corresponds to a new search direction. The update of the solution λ of the CG algorithm at the iteration p is done simultaneously (analogous to (2.12)) for all these orthogonal vectors in a similar

way that the classic CG, by adding the linear combination of these directions that minimizes the error in the F-norm

$$\lambda_{p+1} = \lambda_p + W_p \rho_p$$

with $\rho_p \in \mathbb{R}^{N_s}$, such that

$$\rho_p = (W_p^T F W_p)^+ Z_p^T g_p$$

Since the classical search direction is

$$w_p = W_p(1, \ldots, 1)^T \in range(W_p)$$

this new approximation is better than the usual one given by the CG algorithm.

We can see the need to find the inverse of the matrix $W_p^T F W_p \in \mathbb{R}^{N_s \times N_s}$. Because W_p is formed from local contributions, we would expect that it is full-ranked. This is not always the case (details will be given in later sections), and we only have a symmetric positive semidefinite matrix that can only be pseudo-inverted.

To avoid this pseudo-inversion, another equivalent option is to eliminate some directions to recover a smaller full-ranked family in W_p. The approximated solution will be the same, but fewer vectors would need to be stored at every iteration.

To build this smaller full-ranked family, a rank revealing Cholesky factorization with complete pivoting is used, giving

$$N^T (W_p^T F W_p) N = L L^T$$

where N is a permutation matrix and

$$L = \begin{bmatrix} \tilde{L} & 0 \\ \times & 0 \end{bmatrix}$$

with $\tilde{L}$ a lower triangular matrix of full rank. The F-orthogonalization of the

directions in the block W_p can be done by

$$\begin{aligned} W_p &\leftarrow W_p N \begin{bmatrix} \tilde{L}^{-T} \\ 0 \end{bmatrix} \\ FW_p &\leftarrow FW_p N \begin{bmatrix} \tilde{L}^{-T} \\ 0 \end{bmatrix} \end{aligned} \tag{2.14}$$

where we are suppressing the redundant directions. The computation of the optimization parameters ρ_p is now

$$\rho_p = W_p^T g_p$$

In Algorithm 8 we can find the description of this method. The definition of the projection and the matrices to compute λ_0 are defined as in previous FETI methods (see Algorithm 5).

Algorithm 8 S-FETI algorithm

Initialization
$\lambda_0 = AG[G^T AG]^{-1}(-R^T c)$
$g_0 = P^T(F\lambda_0 - d)$
$Z_0 = [\dots, D^{(s)^{-1}} g_0, \dots], \ \ s = 1, N_s$
$W_0 = PZ_0$
loop Iterate $p = 0, 1, 2, \dots$ until convergence
    $NLL^T N^T = W_p^T F W_p$
    $W_p = W_p N L^{-T}$
    $\rho_p = -W_p^T g_p$
    $\lambda_{p+1} = \lambda_p + W_p \rho_p$
    $g_{p+1} = g_p + P^T F W_p \rho_p$
    $Z_{p+1} = [\dots, D^{(s)^{-1}} g_{p+1}, \dots], \ \ s = 1, N_s$
    $W_{p+1} = PZ_{p+1}$
    for $i = 0$ to p **do**
        $\Phi_i = -W_i^T F W_{p+1}$
        $W_{p+1} = W_{p+1} + W_i \Phi_i$
        $FW_{p+1} = FW_{p+1} + FW_i \Phi_i$
    end for
end loop

We note that when building the block of search directions in every iteration, we are not making this block orthogonal to the ones computed in previous steps, meaning that in fact, we are losing the short recurrence of the Conjugate Gradient method, adding the need of doing a full orthogonalization at every iteration. In any case, this drawback is only theoretical since in practice this full orthogonalization is also needed to keep the numerical accuracy of FETI methods in general.

With this orthogonalization, it can be proved the following minimization property

Theorem 2.1. *The approximate solution computed by the p iteration of the S-FETI*

method minimizes the error $\|\lambda_p - P\lambda\|_F$ *over the space*

$$\lambda_p \in \bigoplus_{i=0}^{p-1} span\left\{W_i\right\}$$

where $\oplus$ *is the direct sum and* W_i *is defined by Algorithm 8*

Proof. The proof is done in [38] following the usual demonstration for CG, see [69]. □

One particularity of this method is that we are no longer minimizing over a Krylov space. This is because, at every iteration, the approximate solution is updated in the different directions given by the optimal combination of all local preconditioners, making the coefficients ρ to change from one iteration to another. This fact will not allow finding a bound (heuristic) for the number of iterations, but in any case, we will expect the robustness that can be explained by the similarities of this method and the FETI-Geneo method [75].

Remark The S-FETI method has been shown to be very efficient on hard problems where the classical FETI require many iterations. The convergence is comparable to the one in FETI-Geneo algorithm where a coarse space is constructed by solving in each subdomain and at every iteration the generalized eigenvalue problem

$$S_{bb}^{(s)} v^{(s)} - \mu B^{(s)^T} D^{-1} B^{(s)} v^{(s)} = 0 \qquad s = 1, \ldots, N_s$$

This problem allows isolating the part of the solution on which the preconditioner is not sufficiently efficient for the iterative solver to perform well. The vectors detected are the ones where the restriction of the global preconditioner is not a good approximation of the non-assembled local component $S_{bb}^{(s)}$ of the FETI operator. In S-FETI the solution space comes from the successive applications of the local non-assembled components $B^{(s)} S_{bb}^{(s)} B^{(s)^T}$ and the assembled FETI operator F, so the block of search directions spans a space where the local effects are taken into account. It is similar to the deflated space where the Geneo iterations take place, and thus convergence is expected to be very fast.

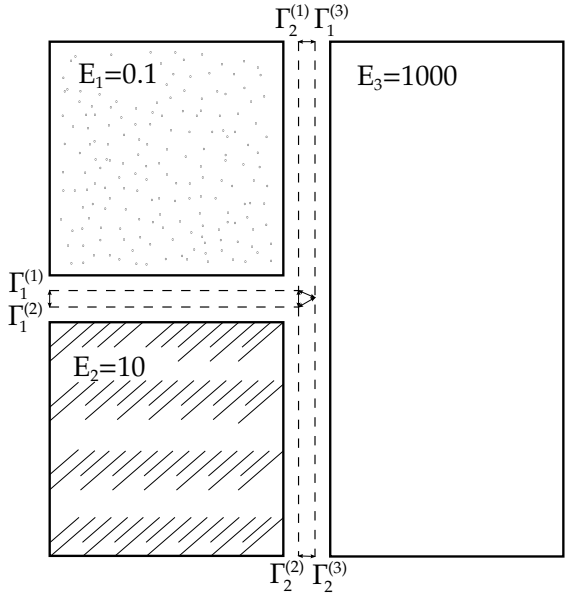

Figure 2.3 – Local interfaces with completely different subdomains (Young's modulus $0.1 \leq E \leq 1000$)

Extension to local interface division

In the regular first version of the S-FETI method, the preconditioner is decomposed in the local contributions

$$D^{(s)^{-1}} = B^{(s)} S_{bb}^{(s)} B^{(s)^T}$$

where $S_{bb}^{(s)}$ is the Schur complement of the internal nodes into the interface ones. This implies that we use the complete local interface $\Gamma^{(s)}$ to build the different search directions. However, different interactions can occur for the same subdomain, depending on the characteristics of each of its neighbors.

If all of the neighbors have similar properties, then the use of the complete local interface is justified. If this is not the case, for example in Figure 2.3, then we can again divide the local interface into the different interface edges, one for each neighbor, to take into account these differences. Increasing the number of search directions in each iteration and improving the approximation of the local behaviour can reduce the convergence ratio of the method.

The formal construction of these search directions comes from the definitions

of the preconditioner and the interface edges given in previous sections. Let us consider the Dirichlet preconditioner with any consistent scaling

$$D^{-1} = \sum_{s=1}^{N_s} \tilde{B}^{(s)} S_{bb}^{(s)} \tilde{B}^{(s)^T}$$

then in the S-FETI method we consider the local contributions separately

$$D^{(s)^{-1}} = \tilde{B}^{(s)} S_{bb}^{(s)} \tilde{B}^{(s)^T}$$

from which we create the N_s different search directions. This operator is applied in the CG method to the computed gradient g, defined in the entire interface Γ. To do this, we first multiply by the assembling scaled matrix $\tilde{B}^{(s)^T}$ and then we compute the forces needed to have this displacement

$$\delta f_{bb}^{(s)} = S_{bb}^{(s)} \tilde{B}^{(s)^T} g$$

We know that these forces are defined in the local interface $\Gamma^{(s)}$, but since their interactions may change from one neighbor to another, we create new vectors that reflect these interactions separately by considering the restrictions to the interface edges $\Gamma^{(sq)} = \partial\Omega^{(s)} \cup \partial\Omega^{(q)}$

$$\delta f_{\Gamma^{(sq)}}^{(s)} = \delta f_{bb}^{(s)}|_{\Gamma^{(sq)}} \qquad s = 1,\dots,N_s, \quad q = 1,\dots,n^{(s)}$$

where $n^{(s)}$ is the number of neighbor subdomains in $\Omega^{(s)}$. We extend this vector by zero to match the size of the interface $\Gamma^{(s)}$

$$\tilde{f}_{\Gamma^{(sq)}}^{(s)} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \delta f_{\Gamma^{(sq)}}^{(s)} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{2.15}$$

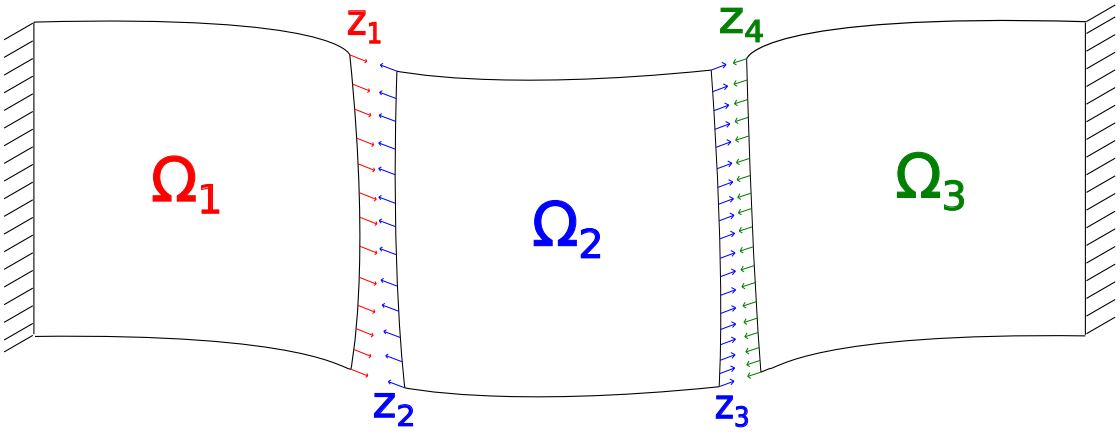

Figure 2.4 – Three subdomain subdivision and computed corrections

each one of this vectors is rescaled to recover the consistent property, and with this form, the columns of the block used to build the different search directions

$$D^{(s)^{-1}}_{\Gamma^{(sq)}} g = \tilde{B}^{(s)} \tilde{f}^{(s)}_{\Gamma^{(sq)}} \qquad s = 1,\dots,N_s, \quad q = 1,\dots,n^{(s)}$$

so finally, the search space will now be generated by the block

$$Z = \left[\dots, D^{(s)^{-1}}_{\Gamma^{(sq)}} g, \dots\right] \qquad s = 1,\dots,N_s, \quad q = 1,\dots,n^{(s)} \tag{2.16}$$

in this case, we will not have one column for each subdomain, but instead, there will be one for each local edge (or neighbor). This will allow spanning an even bigger search space at the cost of computing all the new directions.

Graphically we can see the difference between this block and the regular one, by looking at Figure 2.4 and noting that the blocks in both cases will be

$$\text{Subdomain division} \Rightarrow Z = \begin{bmatrix} z_1 & z_2 & 0 \\ 0 & z_3 & z_4 \end{bmatrix}$$

$$\text{Interface division} \Rightarrow Z = \begin{bmatrix} z_1 & z_2 & 0 & 0 \\ 0 & 0 & z_3 & z_4 \end{bmatrix}$$

We note here that when the problem is divided into much more subdomains, the block Z will show a sparse pattern. In every column, only the values associated

with $\Gamma^{(s)}$ (or in particular $\Gamma^{(sq)}$) are non zero. This fact will be very important in the practical implementation of the method, as we will see later in this section.

As done in the regular S-FETI, to compute the final search directions, each of the columns of Z needs to be projected and then orthogonalized with the previous stored vectors. Then we use the same rank revealing strategy (2.14) to compute the inverse of $W_p^T F W_p$ and eliminate the useless directions. The algorithm is analogous to the previous one and is described in Algorithm 9.

Algorithm 9 S-FETI algorithm with local interface subdivisions.

Initialization
$\lambda_0 = AG[G^T AG]^{-1}(-R^T c)$
$g_0 = P^T(F\lambda_0 - d)$
$Z_0 = [\dots, D^{(s)^{-1}}_{\Gamma^{(sq)}} g_0, \dots], \quad s = 1, N_s \quad q = 1, \dots, n^{(s)}$
$W_0 = PZ_0$
loop Iterate $p = 0, 1, 2, \dots$ until convergence
    $NLL^T N^T = W_p^T F W_p$
    $W_p = W_p N L^{-T}$
    $\rho_p = -W_p^T g_p$
    $\lambda_{p+1} = \lambda_p + W_p \rho_p$
    $g_{p+1} = g_p + P^T F W_p \rho_p$
    $Z_{p+1} = [\dots, D^{(s)^{-1}}_{\Gamma^{(sq)}} g_{p+1}, \dots], \quad s = 1, N_s \quad q = 1, \dots, n^{(s)}$
    $W_{p+1} = PZ_{p+1}$
    for $i = 0$ to p **do**
        $\Phi_i = -W_i^T F W_{p+1}$
        $W_{p+1} = W_{p+1} + W_i \Phi_i$
        $FW_{p+1} = FW_{p+1} + FW_i \Phi_i$
    end for
end loop

Both algorithm for S-FETI show an implementation in general lines of the method, but several optimizations can be done by using different strategies, all of this in order to reach for the maximum performance of this method.

2.1.5 Cost and implementation of S-FETI

One of the characteristics of this method is the reduction in the number of total iterations versus the classical FETI, at the cost of the extra computations in each step. However, even if the cost of each iteration does require a much larger computational effort, several strategies in the implementation of the code lead to efficient local and global optimizations that reduce these cost differences. We start by summarizing the main differences between the cost of S-FETI and FETI-1LM.

1. The number of exchanges phases is the same. The number of communications (exhanges) between neighbors are almost identical, with one less interface exchange when computing the Dirichlet preconditioner.

   The main difference is that the exchanges in S-FETI involve more data, increasing to $N_T \times N_T$ for the matrices $W_p^T F W_p$, Φ and to N_T for the vector ρ. In here, N_T is either the number of subdomains (in the basic S-FETI) or $N_T = \mathcal{O}(N_s)$ when using the local interface subdivision (actually is two times the number of local interfaces). Also, the full reorthogonalization process is now done in a block way, so we change the vector operations by matrix operations, so again, we have the same number of exchanges and computation, but with more data involved.

   The advantages of keeping the number of exchanges, but increasing the size of the data are more evident if we consider the time consumption of the communications within an MPI implementation [13].

   In general, the cost of communication between processes (in our case, the total number of processes is the number of subdomains) can be described by

   $$T(n) = \alpha + n\beta$$

where

$$n : number\ of\ data\ items.$$
$$\alpha : startup\ time.$$
$$\beta : transmission\ time\ per\ data\ item.$$

The startup cost is due to both hardware and software overhead on the sending and the receiving process. Typically, α is four to five orders of magnitude higher than β, where β is on the order of the cost of an instruction.

All these considerations, give us some general lower bounds in time for the communications. The most important one in the startup time, which depends on the number of processes (subdomains), and is independent of the type of communication, either send-receive or reduce operations. This bound is

$$T(n) \geq \log_2(p)\alpha, \qquad p : processes$$

For this reason, avoiding communication is one of the main issues in parallel programming. Therefore, in the S-FETI method, big part of the speedup comes from the reduction in the total number of exchange phases (as they are the same as in FETI per iteration, but the total iterations are reduced).

2. The addition of a Cholesky factorization of the small $N_T \times N_T$ dense symmetric positive semidefinite matrix $W_p^T F W_p$ is now needed.

   In the next section, we will give more details about this, and we will show an alternative to this procedure.

3. The number of stored directions is increased. Since the number of search directions increases at every iteration, we now store a block of directions W_p and FW_p instead of single vectors.

A priori this difference may not be very significant, but as we will see later, it will turn in a major drawback for some cases, specially when using the modified version of S-FETI, where even more directions are created.

4. In general, the most costly part of a FETI algorithm is the computation of solutions of the local Neumann and Dirichlet problems in each subdomain. In the case of the Dirichlet problem for the preconditioning, there is no difference between the methods, but the F operator must now be applied to the N_T columns in W_p, meaning that, a priori, N_T local Neumann problems should be solved at every iteration. However, these computations can be performed efficiently, see item 1 in the next part.

   As already said in the first point, communication cost is mainly driven by the startup time. When multiplying F by a block instead of a vector, the number of interchanges does not change, only the size of it, meaning that we will avoid the significant startup time, for each iteration reduced with this method.

   Another point to consider is that block operations are proportionally less expensive than single vector ones, e.g., N times a matrix-vector product versus a matrix-matrix product with N columns in the second matrix. This is because the computation time is driven by the memory access time. More details of this will be in the following implementation part.

   Finally, but more important in our case, is the clever use of the locality of data. Let us note that the matrix Z_{p+1} is a sparse matrix. This is due to the fact that each column associated with the preconditioner coming from the subdomain $\Omega^{(s)}$ is non zero only in the local interface $\Gamma^{(s)}$ (or $\Gamma^{(sq)}$). In contrast W_{p+1} it is not sparse, because of the projection and orthogonalization. Moreover, we can observe that

$$FW_{j+1} = FPZ_{j+1} + \sum_{i=0}^{j} FW_i\Phi_{ij}$$
$$= \left(FZ_{j+1} - FAG[G^TAG]^{-1}G^TZ_{j+1}\right) + \sum_{i=0}^{j} FW_i\Phi_{ij}$$

where we remember that A is a symmetric matrix, that can be taken as the preconditioner, some scaling matrix or the identity matrix in the simplest case.

The previous equation show that in every iteration we only need to solve the localized problems FZ_{p+1} using the sparsity of Z_{p+1}. The computation of the projection is done by obtaining the sparse matrix (FAG) during the initialization once and for all.

The theoretical optimal choice for A is to use the preconditioner, either Dirichlet or Lumped $A = D^{-1}$ or $A = L^{-1}$ respectively. Both choices are rather computationally expensive compared to some other alternatives. From here on, we will choose

$$A = \sum_{s=1}^{N_s} \tilde{B}^{(s)} diag\left(K_{bb}^{(s)}\right) \tilde{B}^{(s)^T} \tag{2.17}$$

This is the so-called $SuperLumped$ scaling, which has the property of being computationally inexpensive. This comes from the fact that we only need to multiply locally by a diagonal matrix. At the same time, the convergence rate of the FETI method is improved, see [26].

Remark: With all these differences, the final extra cost per iteration remains minimal when N_T is not too large. Also, the total of Neumann problems to solve in each subdomain can be done simultaneously as a block, greatly reducing the cost of the method. This extra cost needs to be weighted with the final number of iterations of the method, where a reduction of the order of N_T is expected,

making the total time of this method an improvement compared to the computation time in FETI-1LM.

To understand how some of the computations are done to improve the efficiency of S-FETI, we can now give some details of the practical implementation

1. Let us return to the last point in the cost of the S-FETI method, the simultaneous forward-backward substitutions. By definition, the columns of Z_p related to $\Omega^{(s)}$ (the s-column in the regular S-FETI) are nonzero only on the interface $\Gamma^{(sq)}$. So, given a column in Z_p, its product by F requires solving a Neumann problem just in the subdomain itself and its neighbor $\Omega^{(q)}$. On the other hand, given one subdomain $\Omega^{(s)}$, the workload associated with the computation of FZ_p is $\mathcal{N}$ Neumann solves, where

$$\text{Subdomain division} \Rightarrow \mathcal{N} = neighbors + 1$$
$$\text{Interface division} \Rightarrow \mathcal{N} = 2 \times neighbors$$

   In particular, this is much fewer than the rank of Z_p^T, which is in the order of the number of subdomains (the exact number of subdomains in the first version).

   If we compute the multiple local solutions at once, the efficiency is greatly improved on a multi-core machine. In fact, the difference between doing $\mathcal{N}$ forward-backward substitution for the separate vectors and doing a single substitution with $\mathcal{N}$-rhs (right-hand side) lies in the number of memory accesses between them. In the second case, we do almost the same amount of accesses than a single rhs (which is the number of non zero entries of the factorized matrix), meaning that even if we multiply the arithmetic complexity by $\mathcal{N}$ we will not have the bottleneck that is the memory access.

   To understand this, we need to see how the memory access on a single or multi-core machine works. On a single core machine, we identify 5 levels of memory from which the data must move before doing a computation at the top. These levels can be seen as a pyramid, where the top of the pyramid contains the memory that is the fastest, and also the smallest, and

Number of cores	Substitutions	Time (s)
1	1	0.7
12	12	1

Table 2.1 – Time for forward-backward substitution on a multi-core processor.

the bottom of the pyramid contains the memory that is the slowest but also the largest.

The best routines to take advantage of such a pyramid are the Basic Linear Algebra Subprograms of level 3 kernels (BLAS3). The reason BLAS3 can benefit of this hierarchy is that their memory interaction is organized in such a way that it takes advantage of the various stages of the pyramid. This leads to having an order of magnitude more computations than memory interactions. In Level-2 BLAS we have $O(n^2)$ data transfers and $O(n^2)$ operations, whereas in BLAS3 we have the same number of data moves $O(n^2)$ but we enlarge the operations to $O(n^3)$.

The pyramid concept does not entirely map the modern multi-core optimizations. The difference is that on multi-core, there are often many cores at the top of the pyramid, and they may have a shared memories between them. However, this does not change the fact that block operations are a much faster option.

As an example given in [38], we can name the case of a sparse matrix of dimension 2×10^5 on a 12-core Intel Nehalem processor, Santa Clara, California, US. In Table 2.1 we can see the time for a single forward-backward substitution versus 12 simultaneous substitutions on 12 cores using the PARDISO solver [71]. With these results, we can see that a good local optimization reduces the impact of the extra computations in S-FETI on multi-core machines.

2. Another critical issue is the parallel implementation of the projector P. Let us recall its definition

$$P = I - AG(G^T AG)^{-1} G^T$$

as previously said, we will consider A as the super lumped scaling to avoid expensive extra computational cost; the projector is then denoted by P_A. Even if this projection is not local, it only performs a low-rank correction. With this, the computation of PZ_p implies a small extra cost.

In practice what it is done, is first to consider

$$P_A Z_p = Z_p - (AG)\alpha_p \text{ where } \alpha_p \text{ solves } (G^T AG)\alpha_p = G^T Z_p$$

This will guarantee that $G^T P_I Z_p = 0$. We recall the definition of G

$$G = \left[B^{(1)} t^{(1)} R^{(1)} \dots B^{(N_s)} t^{(N_s)} R^{(N_s)}\right]$$

where $R^{(s)} = ker(\Omega^{(s)})$. Comparing the definition of G and Z_p we can see that their sparse pattern are very similar. In fact in both cases for each column associated with the subdomain $\Omega^{(s)}$ the pattern is the same; meaning that, given a column $Z_p^{(sq)}$ of Z_p, $G^T Z_p^{(sq)}$ is computed by applying only dot products by the columns of G corresponding to $\Omega^{(s)}$ and its neighbors.

The matrix $(G^T AG)$ is factorized during the initialization, so the computation of α_p is done via a forward-backward substitution. Furthermore, it can be done in parallel, each subdomain $\Omega^{(s)}$ solving the following system, using its own columns (the ones associated to $\Gamma^{(sq)}$, $q = 1, \dots, n^{(s)}$)

$$(G^T AG)\alpha_p^{(sq)} = G^T Z_p^{(sq)}$$

We know that α_p is a dense matrix with the same number of rows as the $rank(G^T)$ and with N_T columns (the same as Z_p). In any case, once it is computed, we only require a low-rank correction of Z_p in each subdomain, because just a few columns of G are nonzero in $\Omega^{(s)}$. With this, we compute the total correction

$$P_A Z_p = Z_p - AG\alpha_p \tag{2.18}$$

To understand better this procedure and the general implementation for the treatment of each sparse block, we will define what we call the *coarse modes*. These modes correspond to the non-null vectors in each one of

the columns of the matrices Z, G, FZ, FAG. They are stored in each local interface of each subdomain and have a correspondence with the named global matrices.

In the left part of Figure 2.5 we show from a subdomain point of view the different coarse modes stored. They are the local (but shared with neighbors) modes that describe Z or G, so for each interface, we save one local mode and one from its neighbor. This double storage allows performing computations of, for example, $G^T Z$, in every subdomain by only performing dot products between the coarse modes saved in each local edge of the subdomains. This way of storage is generalized to allow local computations between these four matrices only by performing operations between the different modes stored this way.

In the case of the multiplication by the operator F, we perform in each subdomain a forward-backward substitution of each one of the modes previously described. In the basic case, they are considered as a single mode (when doing the local subdivision it is $2 \times neighours$). Then we store the resulting (more numerous) modes in each interface.

Also in the left drawing, we see the subdomains sharing the interfaces involved in multiplications by the F operator.

In the right part of Figure 2.5 we show the coarse modes that describe FZ or FAG stored in one of the interfaces of certain subdomains. For each local interface, we save the modes after the forward-backward substitution plus the modes of the neighbor after the substitution.

This implementation in the form of coarse modes allows the computations of matrix products such as $G^T(FAG)$, only by performing dot products between these modes. This implementation also explains how a single substitution is done when several right-hand sides are present, just by arranging (copy) the coarse modes in a block matrix. The same way, we can improve multiple dot products computations by arranging the modes in order to perform matrix-vector products or matrix-matrix products that will enhance the performance of this type of computations (see the previous point).

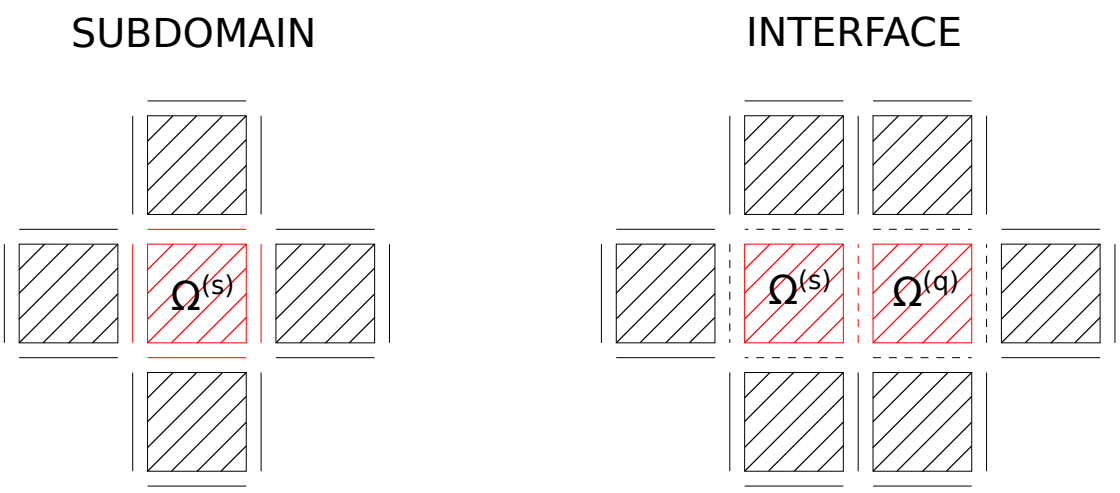

Figure 2.5 –
Subdomain point of view (Left): In red the coarse modes describing Z or G owned by subdomain $\Omega^{(s)}$, in black the interfaces where F times the red modes is also non-null.
Interface point of view (Right): In doted lines the coarse modes describing FZ or FAG owned by local interface $\Gamma^{(sq)}$ between subdomains $\Omega^{(s)}$ and $\Omega^{(q)}$, in black the modes where, due to $\Gamma^{(sq)}$, $Z^T(FZ)$ is non-null.

3. The previous process it is also important in the calculations to obtain FW_p. Let us recall that the orthogonalization process that leads to FW_p is applied to the block FPZ_p, so we need to compute this block first. To do so, we use the matrix α_p previously computed (as shown in the preceding point), then we need to do

$$FPZ_p = FZ_p - (FAG)\alpha_p$$

What we are doing here is a low-rank correction, but this time of FZ_p. The difference is that, to do these corrections, we now use the sparse pattern of (FAG) (each column is non-zero in $\Omega^{(s)}$ and its neighbors) instead of the one in G. The sparse matrix (FAG) is calculated at the initialization once and for all.

4. The last item is the one regarding the orthogonalization and full reorthogonalization procedures for the search directions. As said in the previous

item, once the block PZ_p is computed they must be F-orthogonalized to obtain the search directions W_p and its corresponding block FW_p. This can be achieved by using the standard modified Gram-Schmidt procedure, which will require the use of many communications between subdomains, typically Message Passing Interface (MPI) reductions of dimension 1.

To avoid these many communications we can compute $(PZ_p)^T(FPZ_p)$ by using the local contributions of each subdomain obtained with the BLAS3 and then making only one MPI reduction. The Cholesky factorization is then applied to compute the block of directions W_p as in line [8] of Algorithm 9.

As for the full reorthogonalization process, we use a block modified Gram-Schmidt procedure with the same BLAS3 strategy. Thus, we will have a number of MPI reductions equal to the ones used in classical FETI, but changing from one vector to a matrix and therefore changing the size of the reductions from 1 to at most $N_T \times N_T$.

The evaluation of all these considerations for a good implementation will be shown in the numerical results at the end of this chapter.

2.2 Sorting search directions in S-FETI

The block version of the FETI method has proven to be a significant improvement in comparison with its basic version, on the basis of enlarging the search space of the solution. As a consequence of this extension, we have changes in the algorithm that will impact the execution time. We already named in the previous chapter the main differences between them; we studied the cost difference between FETI and S-FETI where we point out the need to inverse the matrix W^TFW.

This matrix is in general positive semi-definite because of the linear dependencies that may occur in the directions built. We can only aim to compute a pseudo-inverse, but for this same reason, it can also be used to sort the different directions between the useful ones and the rest. Even if, a priori, the computation of this pseudo-inverse is cheap in terms of time consumption its impact on

the final results of the algorithm needs a detailed study.

2.2.1 Linear dependence in block FETI directions

We start this section by recalling the search block built from the preconditioner. At every iteration, it is computed as

$$W_p = PZ_p + \sum_{i=0}^{p-1} W_i \Phi_{ip}$$

This block is orthogonalized with respect to previous blocks W_i already stored. Also, the columns of each of the stored blocks, need to form an F-orthonormal family, which is the norm used in CG algorithm, i.e., for every block they need to fulfill

$$W_p^T F W_p = I \tag{2.19}$$

from this, we know that we also have to find a decomposition of $W_p^T F W_p$ that allows us to modify the basic blocks to build this orthonormal family. Thus the need to study the properties of this matrix.

Since F is symmetric, it is clear that this matrix is also symmetric, but it is not so evident that it will be positive definite. The lack of this property comes from the fact that a linear dependence between the columns of W_p can appear at a certain point of the main iteration. This dependence comes mainly from two issues; the first is the convergence on some of the local interfaces and second is the working precision.

To give a better understanding of the first issue, we have to consider that the global interface is built from the smaller local interfaces that exist between two neighbor subdomains. The characteristics of each of these local interfaces vary from one to another, all being of variable size (as in number of d.o.f.) and also with different mechanical properties along the same. Both characteristics have an impact on the convergence of the method.

For the simpler or smaller local interfaces we expect a faster convergence, for example in a 2D elasticity problem. If the interface consist on one node (e.g., in a corner) the convergence is achieved in two iterations, so the columns created

from this interface (in the subdivided version of S-FETI) will no longer be useful and the linear dependency appears.

For the issue of representation in finite precision arithmetic we know that it depends on the configuration, but it is also closely related to the phenomenon of achieved convergence. Meaning that we can have directions whose differences are so small that it is difficult to distinguish them from numerical noise. In practice what we have is the same direction, so the use of both is redundant.

With both previous issues in consideration, it is clear that the matrix $W_p^T F W_p$ is only positive semi-definite, so no inverse exists. We have two choices from here, either compute a pseudo-inverse and use the complete block with redundancies, or we can sort the directions in order to build a smaller but full rank block with the same information but less computational effort in the operations to follow.

The second choice is in practice the best one. To make this sorting, in [38] a Cholesky factorization is proposed, but we can also add a second way that consists in applying a diagonalization process to the matrix $W_p^T F W_p$. Both procedures will be detailed in the next sections.

2.2.2 Cholesky factorization with complete pivoting

For the process of F-orthonormalization done to the block of search directions, as already said, we start with the decomposition of the $W_p^T F W_p$ matrix. Since this matrix is symmetric, we can use the Cholesky decomposition, instead of a regular QR decomposition, meaning that at each step of the algorithm a lower triangular matrix L_p is built. This matrix is such that

$$W_p^T F W_p = L_p L_p^T$$

This decomposition only works if the previous matrix is of full rank. This is not our case, so we need to use the alternative Cholesky decomposition with complete pivoting (symmetric pivoting) described in [43]. The algorithm for this factorization can also work as a rank revealing procedure. This is achieved by computing a square permutation matrix N, in this case, the decomposition can

be written as

$$N^T W_p^T F W_p N = L_p L_p^T$$

and if we look in detail the matrix L_p we know that it can be described as

$$L_p = \begin{bmatrix} \tilde{L}_p & 0 \\ \times & 0 \end{bmatrix}$$

where

$$\tilde{L}_p \in \mathbb{R}^{r \times r}$$

this smaller matrix is lower triangular with positive elements in the diagonal, and it has a full rank $r \leq N_T$, which is the rank of the $W_p^T F W_p$ matrix.

The existence of the permutation matrix (proved in [45], Thm. 10.9) allows sorting the directions that are useful from the others. This is done merely by taking the first r columns of $W_p N$.

In summary, at each iteration of the method, the directions to be used and stored are defined as

$$W_p \leftarrow (W_p N) \begin{bmatrix} \tilde{L}_p^{-1} & 0 \end{bmatrix}^T$$

this block of sorted directions fulfills the (2.19) condition, in fact

$$\begin{aligned} \begin{bmatrix} \tilde{L}_p^{-1} & 0 \end{bmatrix} (W_p N)^T F (W_p N) \begin{bmatrix} \tilde{L}_p^{-1} & 0 \end{bmatrix}^T &= \begin{bmatrix} \tilde{L}_p^{-1} & 0 \end{bmatrix} L_p L_p^T \begin{bmatrix} \tilde{L}_p^{-1} & 0 \end{bmatrix}^T \\ &= \begin{bmatrix} \tilde{L}_p^{-1} \tilde{L}_p & 0 \end{bmatrix} \begin{bmatrix} \tilde{L}_p^{-1} \tilde{L}_p & 0 \end{bmatrix}^T \\ &= \begin{bmatrix} I \\ 0 \end{bmatrix} \end{aligned}$$

with $I \in \mathbb{R}^{r \times r}$.

The algorithm for the computation of the Cholesky factorization, introduces the use of a new parameter $\varepsilon > 0$ (close to zero), to determine at which point we are in the presence of linear dependence. If this parameter is too small we can be working within the numerical noise and losing precision in the total convergence. On the contrary, using one that is too big can have the impact of losing directions that may be important for the convergence.

Finding the optimal ε is not an easy task and is usually chosen based on

numerical experiences. As a primary consideration, we will try to be closer to the "losing" directions approach. With this, even if we have a slower convergence, eventually we will recover these directions in later iterations. On the other hand, working within the range of the numerical noise of the method can only introduce sources of error harder to control, and may even end in blowups.

Usually, a regular double precision implementation should be enough, but introducing new sources of numerical instabilities needs to be done carefully. Later in numerical tests, we will compare the use of quadruple precision in the computation of the Cholesky decomposition for this matrix, as an alternative for accurately capture the directions needed.

2.2.3 Diagonalization of the search directions block

The second way to treat the sorting of independent search directions can be to use a diagonalization process. At first, we know that this procedure will lead to the decomposition

$$W_p^T F W_p = E_p D_p E_p^{-1}$$

where E_p is the matrix of the eigenvectors in the columns, and D_p is a diagonal matrix with the eigenvalues on the diagonal. In our case, the matrix is real and symmetric, so we have that $E_p^{-1} = E_p^T$. Hence, if we are looking for a block that fulfills the condition (2.19) we can use the previous decomposition to build it. Then, we have

$$W_p^T F W_p = E_p D_p E_p^T$$

this implies that

$$E_p^T W_p^T F W_p E_p = D_p$$

the matrix D_p is diagonal, then we can easily find its inverse, and even more, we can divide it into the multiplication of two diagonal invertible matrices, meaning that we build

$$E_p^T W_p^T F W_p E_p = D_p^{1/2} D_p^{1/2}$$
$$(W_p E_p D_p^{-1/2})^T F (W_p E_p D_p^{-1/2}) = I$$

and so, the F-orthogonalized search directions can be constructed by doing

$$W_p \leftarrow W_p E_p D_p^{-1/2}$$

the previous diagonalization process works if the matrix $W_p^T F W_p$ is invertible, but we only have that is positive semi-definite, so the eigenvalues are greater or equal than zero.

The standard algorithms for the computation of eigenvalues can be used to obtain them in descending (or ascending) order. This fact will allow us to build a smaller F-orthonormal full block of search directions by taking into account the largest of these eigenvalues and their associated eigenvectors. The construction is the same as previous, but we will only consider the r largest eigenvalues to build the block

$$W_p \leftarrow W_p \tilde{E}_p \tilde{D}_p^{-1/2}$$

where

$$rank(W_p^T F W_p) = rank(\tilde{E}_p^T) = rank(\tilde{D}_p) = r$$

meaning that, in theory, no information will be missing from this sorting.

The treatment of the linear dependencies is still limited by some $\varepsilon > 0$ parameter, but now this is done in a more precise way. The algorithm for the computation of the eigenvectors and eigenvalues will give them in decreasing order, so the biggest values, and thus the most important ones, are the first columns of E_p. This means that we can eliminate the smallest eigenvalues and vectors below the threshold ε. In this case, the dependency between this parameter and the final result is less significant than in the Cholesky factorization because we do not eliminate certain directions, but instead, we build a smaller full block containing all the relevant information.

Let us recall that all these computations (Cholesky or Eigendecomposition) are done locally. Therefore, the impact in the time of the diagonalization process is minimal, because of the size of the matrix we are solving depends on the number of subdomains. Also, the use of LAPACK algorithm with BLAS optimization allows a fast computation in every subdomain.

Subdomains	Iterations	Max local memory (Gb)	Directions stored
125	7	3.87	11176

Table 2.2 – Direction stored in a 5x5x5 cube configuration

Remark: Later in the numerical results, we will also show the impact in the use of the Lumped preconditioner, as an economical alternative to complement this new decomposition.

2.3 Memory usage in S-FETI

In previous sections, we stipulate that one of the differences between the S-FETI method and the classical FETI lies in the total memory used in both of them. In FETI-1LM, only two vectors are stored at each iteration; they correspond to the search direction and its multiplication by F. On the other hand, in S-FETI the number of search directions built in every iteration is increased, making mandatory the storage of blocks W_p and FW_p. This difference may be a significant drawback in the implementation of S-FETI.

The total number of directions stored depends and grows linearly as the number of iterations. This value is in every step usually the number of columns in Z_p (in practice, due to the linearity of the directions, we expect a smaller number that corresponds to $Rank(Z_p^T)$). In 2D problems, this is not much of an issue, but in 3D even in small cases, this needs to be considered. For example, if our domain is a cube divided into $5 \times 5 \times 5$ smaller cubes, we see in Table 2.2 that the number of stored directions at each step is of 1600 directions approximately, a fact that comes only from the geometry of the problem. Now, the total memory needed will depend on the size of the triangulation of the mesh, in this case of about 10^5 elements in each subdomain.

To be able to use this method in bigger configurations we will change the usual, direct storage and use a different strategy. We know that the time of a parallel application is limited mostly by the exchange of information between processes. Hence, if we add local extra computations that are not too expensive, we hope to have an implementation as fast as the original one, but without the

memory constraint.

2.3.1 New sparse storage

The idea of this new storage is to, reconstruct the search directions at every iteration, that is, to compute the projection and the full reorthogonalization of the columns in Z_p, by storing matrices that are either sparse or "small" ones.

This is done by saving at each iteration the sparse matrices, Z_p and FZ_p, instead of the full $W_p = PZ_p$ and $FW_p = PFZ_p$. With these sparse matrices, we also compute and store some smaller coefficient matrices that will allow this reconstruction. The size of these new matrices will depend on the number of used search directions at each iteration and will be independent of the size of the mesh. In general, it will be a small number compared to the size of the problem.

The memory needed for the storage will now be limited more importantly on the problem configuration. This will determine the number of search directions, and therefore the memory usage, rather than the size of the discretization. Compared with the total unknowns of the problem, this number (N_T) is usually a small one. The number of neighbors of each subdomain is also limited, so the problems that we solve can increase its mesh size and have a lower impact in the memory used by it, allowing us to use the S-FETI method in more extensive problems.

The fact that we no longer store the complete search directions will also have an impact on the precision achieved with this implementation. However, we hope to keep the accuracy of the S-FETI method or at least be close enough to make this a useful algorithm; we will later discuss this fact in the numerical results.

2.3.2 Reconstruction of search directions

In this section, we will show the details used by the algorithm of the sparse storage for the S-FETI method.

Idea of the reconstruction

We will try to use the definition of the projector and full reorthogonalization process, to find a recurrence for the coefficients that define each search direction. Let us consider the sparse matrices Z_p from (2.16) and recalling the definition of the projector

$$P = I - AG(G^T AG)^{-1} G^T$$

with A a symmetric matrix, that can be the identity, the preconditioner or some scaling matrix. We also know that to apply this projector we use low-rank corrections as in (2.18). Starting from Z_p, we have

$$Z_1 \Longrightarrow PZ_1 = Z_1 + (AG)D_1$$

it is clear that

$$G^T PZ_1 = 0$$

and that the computation of D_1 is done by solving

$$(G^T AG)D_1 = -G^T Z_1$$

meaning that the value of D_1 is

$$D_1 = -(G^T AG)^{-1} G^T Z_1$$

no previous directions were computed, so we have our first search direction block, and the projection coefficients that define it

$$\begin{aligned} W_1 &:= Z_1 \\ \Delta_1 &:= D_1 \\ PW_1 &:= PZ_1 = Z_1 + (AG)\Delta_1 \end{aligned}$$

in the next iteration and from the definition of the S-FETI method, we build the sparse block Z_2; then we apply the same process for this block

$$Z_2 \Longrightarrow PZ_2 = Z_2 + (AG)D_2$$

where

$$G^T P Z_2 = 0$$

and D_2 is computed by solving

$$(G^T A G) D_2 = -G^T Z_2$$

so again, its value is

$$D_2 = -(G^T A G)^{-1} G^T Z_2$$

We need to consider the full reorthogonalization against the directions built in the previous step. Hence we have

$$\begin{aligned} W_2 &= Z_2 + W_1 \Phi_{12} \\ \Longrightarrow P W_2 &= P Z_2 + P W_1 \Phi_{12} \end{aligned} \tag{2.20}$$

and to compute Φ_{12} we use the orthogonality properties of the new directions and the previous ones in the CG algorithm

$$(F P W_1)^T P W_2 = 0 \tag{2.21}$$

replacing $P W_2$ we have

$$(F P W_1)^T (P W_1) \Phi_{12} = -(F P W_1)^T P Z_2$$

this problem is solved to compute Φ_{12}. Its value is

$$\Phi_{12} = -\left[(P W_1)^T (F P W_1) \right]^{-1} (F P W_1)^T P Z_2$$

then, in (2.20) we use the projection definition, to obtain

$$\begin{aligned} P W_2 &= Z_2 + (AG) D_2 + (Z_1 + (AG) D_1) \Phi_{12} \\ &= Z_2 + Z_1 \Phi_{12} + (AG)(D_2 + D_1 \Phi_{12}) \end{aligned}$$

so

$$P W_2 = Z_2 + Z_1 B_{12} + (AG) \Delta_2 \tag{2.22}$$

where

$$B_{12} := \Phi_{12} \text{ and } \Delta_2 := D_2 + D_1\Phi_{12}$$

but also, since

$$W_2 = Z_2 + Z_1 B_{12} \tag{2.23}$$

then we have that

$$PW_2 = W_2 + (AG)\Delta_2$$

from this equation, plus from (2.20) and (2.22) we extend the recurrence to a general case where we have the following equivalent equations

$$PW_p = W_p + (AG)\Delta_p \tag{2.24}$$

$$PW_p = PZ_p + \sum_{j=1}^{p-1} PW_j\Phi_{jp} \tag{2.25}$$

$$PW_p = Z_p + \sum_{j=1}^{p-1} Z_j B_{jp} + (AG)\Delta_p \tag{2.26}$$

With the previous recurrences in mind, we will formalize and explicit the computation of the coefficient matrices B_{jp} and Δ_p that needs to be stored. These matrices along with the sparse ones Z_p define the construction of the search directions.

Remark: The previous construction allows the extension of the sparse multiplication against the operator F. From (2.22), we have

$$FPW_2 = FZ_2 + FZ_1 B_{12} + (FAG)\Delta_2$$

and in the general form, using (2.26) we have

$$FPW_p = FZ_p + \sum_{j=1}^{p-1} FZ_j B_{jp} + (FAG)\Delta_p$$

Formal construction

At the iteration p, we apply to Z_p the low-rank correction definition of P

$$PZ_p = Z_p + (AG)D_p$$

with

$$D_p = -(G^T AG)^{-1} G^T Z_p$$

also from the definition of P, we apply it this time to the reorthogonalized block W_p

$$PW_p = W_p + (AG)\Delta_p \tag{2.27}$$

later we will compute Δ_p, not from the classical definition, but recursively.

On the other hand and to avoid notation problems, let us consider some iteration j. We replace in the previous equation (2.27) the computation of W_j (W_p in there) by the Gram-Schmidt orthogonalization process applied to Z_j

$$PW_j = Z_j + \sum_{i=1}^{j-1} Z_i B_{ij} + (AG)\Delta_j \tag{2.28}$$

with B_{ij} to be computed later, recursively. Then, we replace this in (2.25). Actually it corresponds to the formal construction of the orthogonalized and projected search directions, so we have

$$\begin{aligned} PW_p &= PZ_p + \sum_{j=1}^{p-1} \left(Z_j + \sum_{i=1}^{j-1} Z_i B_{ij} + (AG)\Delta_j \right) \Phi_{jp} \\ &= Z_p + (AG)D_p + \sum_{j=1}^{p-1} Z_j \Phi_{jp} + \sum_{j=1}^{p-1} \sum_{i=1}^{j-1} Z_i B_{ij} \Phi_{jp} + \sum_{j=1}^{p-1} (AG)\Delta_j \Phi_{jp} \\ &= \left\{ Z_p + \sum_{j=1}^{p-1} Z_j \Phi_{jp} + \sum_{j=1}^{p-1} \sum_{i=1}^{j-1} Z_i B_{ij} \Phi_{jp} \right\} + (AG) \left\{ \sum_{j=1}^{p-1} \Delta_j \Phi_{jp} + D_p \right\} \end{aligned}$$

from this equation and (2.27) it is clear that we can define the computation of

W_p and Δ_p as

$$W_p = Z_p + \sum_{j=1}^{p-1} Z_j \Phi_{jp} + \sum_{j=1}^{p-1} \sum_{i=1}^{j-1} Z_i B_{ij} \Phi_{jp} \tag{2.29}$$

$$\Delta_p = \sum_{j=1}^{p-1} \Delta_j \Phi_{jp} + D_p \tag{2.30}$$

developing the computation of W_p

$$\begin{aligned} W_p &= Z_p + \sum_{j=1}^{p-1} Z_j \Phi_{jp} + \sum_{j=1}^{p-1} \sum_{i=1}^{j-1} Z_i B_{ij} \Phi_{jp} \\ &= Z_p + \sum_{j=1}^{p-1} Z_j \Phi_{jp} + \sum_{i=1}^{p-2} Z_i \left(\sum_{j=i+1}^{p-1} B_{ij} \Phi_{jp} \right) \\ &= Z_p + \sum_{i=1}^{p-1} Z_i \Phi_{ip} + \sum_{i=1}^{p-2} Z_i \left(\sum_{j=i+1}^{p-1} B_{ij} \Phi_{jp} \right) \end{aligned}$$

and thus our orthogonalized search direction is

$$W_p = Z_p + \sum_{i=1}^{p-1} Z_i B_{ip} \tag{2.31}$$

with

$$B_{ip} := \Phi_{ip} + \sum_{j=i+1}^{p-1} B_{ij} \Phi_{jp}$$

Finally, using the computation of W_p from (2.31) and Δ_p defined recursively in (2.30), the projected search directions at the iteration p can now be computed as

$$PW_p = W_p + (AG)\Delta_p$$

So we can reconstruct the directions only from the sparse matrices Z_j, $j = 1, \dots p$ and the small matrices B_{ij}, Δ_j and Φ_{ij}, $i = 1, \dots, j$, $j = 1, \dots p$. We store these coefficient matrices instead of the full and larger W_p or FW_p.

At each iteration, we start with the Z_p matrix that is built from previous gradients and with it we can compute the coefficients D_p that do not need to be stored. What we can do, for example, is to use the same memory space that Δ_p and later overwrite it.

To define the computation of the coefficients in Φ_{ip} we need to add some extra calculus. They cannot be based on the storage of any full vector with a size depending on the mesh. Based on the general orthogonal properties of the CG algorithm, let us consider at each iteration the generalization of (2.21).

$$(FPW_j)^T PW_p = 0 \qquad \text{for } j < p$$
$$(FPW_j)^T (PZ_p + \sum_{j=1}^{p-1} PW_j \Phi_{jp}) = 0$$

which implies

$$(FPW_j)^T (PW_j)\Phi_{jp} = -(FPW_j)^T PZ_p \qquad \text{for } j < p$$

the left-hand side corresponds to the matrix symmetric positive semi-definite that needs to be pseudo-inverted at each iteration. To simplify the computations we apply the Cholesky decomposition (or Eigendecomposition) to it, giving

$$L_j L_j^T \Phi_{jp} = -\left(FPW_j\right)^T PZ_p \tag{2.32}$$

The L_j matrix needs to be stored at every iteration, but then again these are small triangular matrices of size $rank(Z_j^T)$.

The right-hand side is the one needing a particular treatment. It depends on full terms that we do not want to store. For the PZ_p term, the definition of P is enough to avoid the full storage, but for the term FPW_j, instead of using it directly, we also need to use the reconstructions that comes from the sparse matrices.

From the construction of PW_j in (2.28) and the fact that F is a linear operator,

we have

$$FPW_j = FZ_j + \sum_{i=1}^{j-1} FZ_i B_{ij} + (FAG)\Delta_j$$

from this, as expected, we also need to store the sparse term FZ_j.

Finally the right term in (2.32) is computed

$$\begin{aligned}\left(FPW_j\right)^T (PZ_p) &= \left[FZ_j + \sum_{i=1}^{j-1} FZ_i B_{ij} + (FAG)\Delta_j\right]^T \left[Z_p + (AG)D_p\right] \\ &= Z_j^T (FZ_p) + \sum_{i=1}^{j-1} B_{ij}^T Z_i^T (FZ_p) + \Delta_j^T (FAG)^T Z_p + \\ &\quad + Z_j^T (FAG)D_p + \sum_{i=1}^{j-1} B_{ij}^T Z_i^T (FAG)D_p + \Delta_j^T (AG)^T (FAG)D_p\end{aligned} \tag{2.33}$$

The six previous terms are computed at every iteration for $j = 1, p-1$. With this, we can now obtain from (2.32) the values of Φ_{jp} and the new implementation is completed.

Next, we present a summary of the computations needed and the matrices stored in this new algorithm for the S-FETI method.

$$\begin{aligned}(AG)^T (FAG) &: \text{ Computed once for all and sparse stored} \\ Z_j^T FZ_p &: \text{ Computed at every iteration for } j = 1, p-1 \text{ and sparse stored} \\ Z_p^T (FAG) &: \text{ Computed at every iteration and sparse stored}\end{aligned} \tag{2.34}$$

$$\begin{aligned}D_p &: \text{ Computed at every iteration and overwritten by } \Delta_p \\ \Delta_p &: \text{ Small matrix stored every iteration} \\ (AG)^T (FAG)\Delta_p &: \text{ Small matrix stored every iteration} \\ \Phi_{jp} &: \text{ Small matrices stored every iteration, for } j = 1, p-1 \\ B_{jp} &: \text{ Small matrices stored every iteration, for } j = 1, p-1\end{aligned}$$

With all these considerations, we can describe the S-FETI method with this new storage for the search directions in Algorithm 10.

Compared with the S-FETI method, we just replaced the projection and full reorthogonalization processes to build W and FW. The rest is analogous, including the use of different preconditioners as well as the rank-revealing strategy.

2.3.3 Implementation details and exploitable parallelism

In the previous section, to reduce the memory usage, we introduced several extra computations to the basic S-FETI algorithm. If we want this method to be as fast as the full version, all these new calculations need to be implemented in the most efficient way possible. The method is shown in Algorithm 10 and it describes a simple presentation for the construction of the search directions. It can be implemented straightforwardly, but here we will give some considerations that can provide an important time boost to the use of this method.

Before explaining this, we will explain the sparsity of the new extra terms (2.34) that we need to compute and store, let us recall them

$$(G)^T(FG),\ Z_j^T F Z_p,\ Z_p^T(FG)$$

with $j = 1,\ldots,p-1$ and for simplicity we consider $A = I$. If we look at these three terms, we note that each one share very similar structures, this comes from the definitions of Z and G

$$G = \left[B^{(1)}t^{(1)}R^{(1)},\ldots,B^{(N_s)}t^{(N_s)}R^{(N_s)}\right]$$
$$Z = \left[B^{(1)}\delta f_{bb}^{(1)},\ldots,B^{(N_s)}\delta f_{bb}^{(N_s)}\right]$$

again for simplicity, no scaling is considered in Z. We consider the complete local interface in Z (without any subdivision) and we think in $R^{(s)} = Ker(\Omega^{(s)})$ as a single vector in $\Omega^{(s)}$. In both cases, the extension to local interface division and kernels with more than one vector is straightforward, because each column associated with a subdomain $\Omega^{(s)}$ can be considered to have the same structure.

With these definitions, we see that G and Z have the same sparse structure

Algorithm 10 S-FETI with Sparse Storage

Initialization
$\lambda_0 = AG[G^T AG]^{-1}(-R^T c)$
$g_0 = F\lambda_0 - d$
loop Iterate $p = 0, 1, 2, \ldots$ until convergence
  $Z_p = [\ldots, \tilde{F}^{(s)^{-1}}_{\Gamma^{(sq)}} P^T g_p, \ldots], \quad s = 1, N_s \quad q = 1, \ldots, n^{(s)}$
  $Q_p = FZ_p$
  $D_p = -(G^T AG)^{-1} G^T Z_p$
  for $j = 0$ to $p-1$ **do**
    $S_{jp} = Z_j^T Q_p$
  end for
  $T_p = Z_p^T (FAG)$
  for $j = 0$ to $p-1$ **do**
    $\Phi = S_{jp}^T + \sum_{i=0}^{j-1} S_{ip}^T B_{ij} + T_p \Delta_j$
    $\Phi = \Phi^T + T_j D_j + \sum_{i=0}^{j-1} B_{ij}^T T_i D_p + U_j^T D_p$
    $\Phi_{jp} = -(L_j L_j^T)^{-1} \Phi$
  end for
  $\Delta_p = D_p + \sum_{j=0}^{p-1} \Phi_{jp}$
  $U_p = (AG)^T (FAG) \Delta_p$
  for $j = 0$ to $p-1$ **do**
    $B_{jp} = \Phi_{jp} + \sum_{i=j+1}^{p-1} B_{ji} \Phi_{ip}$
  end for
  $W = Z_p + \sum_{j=0}^{p-1} Z_j B_{jp} + (AG)\Delta_p$ ▷ Compute projected
  $FW = Q_p + \sum_{j=0}^{p-1} Q_j B_{jp} + (FAG)\Delta_p$ and reorthogonalized blocks
  $N_p L_p L_p^T N_p^T = W^T (FW)$
  $\rho = -(\tilde{L}_p \tilde{L}_p^T)^{-1} (WN_p)^T g_p$ ▷ $\tilde{L}_p \in \mathbb{R}^{r \times r}$ is full ranked
  $\lambda_{p+1} = \lambda_p + (WN_p)\rho$
  $g_{p+1} = g_p + (FWN_p)\rho$
end loop

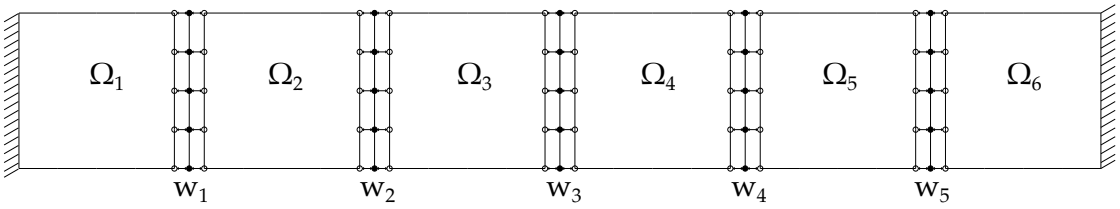

Figure 2.6 – One-way split of six subdomains

given by the definition of the boolean matrices $B^{(s)}$. To see it more clearly, let us take as an example the Figure 2.6 where the domain is divided into six subdomains in a one-way split. Denoting any vector defined in the interface of a subdomain $\Omega^{(s)}$ as $v_b^{(s)}$, and dividing a vector in the total interface w into local interface blocks, we have

$$w = \begin{bmatrix} w^1 \\ w^2 \\ w^3 \\ w^4 \\ w^5 \end{bmatrix}$$

we see that the size and structure of w are the same as $B^{(s)}v_b^{(s)}$, for any $s = 1,\dots 6$. With this, each column of G and Z, defined in the whole interface, have the following structure

$$B^{(1)}v_b^{(1)} = \begin{bmatrix} \times \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad B^{(2)}v_b^{(2)} = \begin{bmatrix} \times \\ \times \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad B^{(3)}v_b^{(3)} = \begin{bmatrix} 0 \\ \times \\ \times \\ 0 \\ 0 \end{bmatrix},$$

$$B^{(4)}v_b^{(4)} = \begin{bmatrix} 0 \\ 0 \\ \times \\ \times \\ 0 \end{bmatrix}, \quad B^{(5)}v_b^{(5)} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \times \\ \times \end{bmatrix}, \quad B^{(6)}v_b^{(6)} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \times \end{bmatrix},$$

Here we see that in each column only a small part of the interface is non zero.

It corresponds to the d.o.f. in the column s shared by the subdomain $\Omega^{(s)}$ ($\Omega^{(s)} \cap \Gamma \neq 0$).

The next structures to consider are the ones of FG and FZ. They correspond to the multiplication of previous blocks by the FETI operator. From the definition of F, we can see that their structure depends on the multiplication of the boolean matrices

$$B^{(s)^T} B^{(q)},\ \forall s, q = 1, \ldots, N_s \tag{2.35}$$

When subdomains s and q are neighbors, this multiplication is non zero, so when multiplying Z or G by F only local interfaces corresponding to the neighbors are non zero. In our example the structures are

$$FB^{(1)}v_b^{(1)} = \begin{bmatrix} \times \\ \times \\ 0 \\ 0 \\ 0 \end{bmatrix},\ FB^{(2)}v_b^{(2)} = \begin{bmatrix} \times \\ \times \\ \times \\ 0 \\ 0 \end{bmatrix},\ FB^{(3)}v_b^{(3)} = \begin{bmatrix} \times \\ \times \\ \times \\ \times \\ 0 \end{bmatrix},$$

$$FB^{(4)}v_b^{(4)} = \begin{bmatrix} 0 \\ \times \\ \times \\ \times \\ \times \end{bmatrix},\ FB^{(5)}v_b^{(5)} = \begin{bmatrix} 0 \\ 0 \\ \times \\ \times \\ \times \end{bmatrix},\ FB^{(6)}v_b^{(6)} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \times \\ \times \end{bmatrix},$$

With previous blocks, the final structure for the terms $(G)^T(FG)$, $Z_j^T FZ_p$, $Z_p^T(FG)$ is given by

$$\begin{bmatrix} \times & 0 & 0 & 0 & 0 \\ \times & \times & 0 & 0 & 0 \\ 0 & \times & \times & 0 & 0 \\ 0 & 0 & \times & \times & 0 \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & \times \end{bmatrix} \begin{bmatrix} \times & \times & \times & 0 & 0 & 0 \\ \times & \times & \times & \times & 0 & 0 \\ 0 & \times & \times & \times & \times & 0 \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times \end{bmatrix} = \begin{bmatrix} \times & \times & \times & 0 & 0 & 0 \\ \times & \times & \times & \times & 0 & 0 \\ \times & \times & \times & \times & \times & 0 \\ 0 & \times & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times \end{bmatrix}$$

here we see that the terms that are non zero correspond in each subdomain to

the subdomain itself plus its neighbors plus the neighbors of these neighbors. This is a general fact that comes from the definitions of Z, G and F

$$(G^T F G)_{ik} = v_b^{(i)^T} B^{(i)^T} \left(\sum_s B^{(s)} S_{bb}^{(s)^{-1}} B^{(s)^T} \right) B^{(k)} v_b^{(k)}$$

in here the term $(G^T F G)_{ik}$ is non zero if $B^{(i)^T} B^{(s)}$ and $B^{(s)^T} B^{(k)}$ are non zero, which occurs when the subdomains i-s and s-j are neighbors.

Even though the last term is less sparse that both Z and FZ (G and FG) its total size is much smaller than previous blocks. They are square matrices of size N_T.

Optimization of the code

With these structures in mind, we can now give the details of several possible optimizations to any basic implementation of Algorithm 10. Let us remember that as part of the new orthogonalization process we have to compute the six terms in (2.33). Then, the improvements come, in one hand, from the fact that some matrix-vector computations can be done in block, using the advantages of BLAS3, and on the other side some of the coefficient matrix computations can be done in parallel.

These changes to optimize the code can be summarized as:

- Concatenated allocation in memory of the terms Z_j with $j = 1, ..., p-1$ so that the computation of $Z_j^T (F Z_p)$ can now be done as a single block for all j.

  To understand this, let us see how the computation of each term $Z_j^T (F Z_p)$ is done locally. Considering a column associated with the subdomain $\Omega^{(s)}$ of Z_p. It is defined in the complete interface, but it takes only local values

  $$z_p^{(s)} = B^{(s)} \tilde{f}_{\Gamma^{(sq)}}^{(s)}$$

  where $\tilde{f}_{\Gamma^{(sq)}}^{(s)}$ is defined in (2.15). Then, each value of $Z_j^T (F Z_p)$ is computed

as

$$\begin{aligned}(Z_j^T F Z_p)_{ik} &= z_j^{(i)^T} F z_p^{(k)} \\ &= z_j^{(i)^T} \left(\sum_s F^{(s)} \right) z_p^{(k)} \\ &= \sum_s z_j^{(i)^T} F^{(s)} z_p^{(k)}\end{aligned}$$

where, as usual $F^{(s)} = B^{(s)} S_{bb}^{(s)^{-1}} B^{(s)^T}$. The values that are non zero are the ones where i-s and s-k are neighbors. This implies that in each subdomain $\Omega^{(s)}$, we need to do an exchange with every neighbor and store in s the vector $z_p^{(k)}$. Then we compute, exchange and store $F^{(s)} z_p^{(k)}$ at every iteration.

In general, each subdomain $\Omega^{(s)}$ will compute and store the sparse vectors $z_p^{(s)}, F^{(s)} z_p^{(s)}, z_p^{(k)}, F^{(s)} z_p^{(k)}$, where $k \in neighbor(s)$.

Finally, each subdomain $\Omega^{(s)}$ will compute the local dot products

$$z_j^{(i)^T} F^{(s)} z_p^{(k)}, \qquad \forall i,k \in neighbor(s) \cup \{s\} \tag{2.36}$$

then an assembly into $(Z_j^T F Z_p)_{ik}$ plus a global reduction are done to compute the total matrix $Z_j^T (F Z_p)$. These are small sparse matrices stored in every subdomain.

The computation is done for each $j = 1, \ldots, p-1$ at the iteration p; meaning that we repeat this process $p-1$ times at each iteration, with the corresponding load of doing the $p-1$ global reductions and $(p-1)$-times the dot products.

To improve the time of these calculations, what we can do is to take advantage of block implementations, as already done in this chapter. To do so, in every subdomain $\Omega^{(s)}$ we do a pseudo-allocation that will concatenate the newly computed and exchanged local vectors $z_p^{(i)}$ into a block of vectors

$$\left[z_1^{(i)}, \ldots, z_{p-1}^{(i)} \right], \qquad \forall i \in neighbor(s) \cup \{s\}$$

with this we can compute the terms in (2.36) as a block

$$\left[z_1^{(i)},\ldots,z_{p-1}^{(i)}\right]^T F^{(s)} z_p^{(k)}, \qquad \forall i,k \in neighbor(s)\cup\{s\}$$

we can use the optimized libraries BLAS3 to obtain a better performance than usual dot products.
Since every Z_j shares the same structure, multiple assemblages can be done into the different $Z_j^T F Z_p$, and for each j at the same time, only a single reduction is made (slightly bigger in data size). We reduce the number of global exchanges from $p-1$ to 1 in each iteration, thus improving the time cost.

- If we look at the following term in the loop to compute Φ_{jp}

$$\sum_{i=1}^{j-1} B_{ij}^T T_i D_p$$

we can see that the matrix D_p is independent of the addition, so it can be treated as

$$\left[\sum_{i=1}^{j-1} B_{ij}^T T_i\right] D_p$$

the term between parentheses is independent of the iteration p and can be computed in previous iterations.

Therefore, instead of doing computations of the type $(FAG)D_p$, we can compute and save

$$V_p \leftarrow \left[\sum_{i=1}^{p-1} B_{ip}^T T_i\right]$$

This computation is added after the matrices B_{ip} are calculated. We store this sum as a single matrix to use it directly in the next iterations. This term is another small matrix, so the memory cost is still controlled.

- Thanks to the first point in here, the matrices needed for the computations of the terms Φ_{jp} for all $j < p$, are global and shared in each subdomain (this includes the matrices coming from the Cholesky factorization). Hence, each process computes the same terms.

  This involves the computation of several small matrix products, that increase in numbers at each iteration. To avoid these increasing computations, the calculation of Φ_{jp} for each $j = 1, p-1$, can be done in parallel.

  Each process j can now be in charge of each matrix Φ_{jp}, up to a limit of N_s matrices (the total number of processes). This means that after the iteration $p > N_s$ the first process will need to do the computation of two matrices Φ_{jp} and then for each extra iteration, another process will be in charge of the new computations. Since the number of iterations of the S-FETI method is expected to be low, there will be some rare cases where this will happen.

  Even if, a priori, this parallelization will speed up the total time, we can not forget that an extra global exchange is needed so that every process can have access to the new computed Φ_{jp} matrices. The impact of this change will be tested in the numerical examples.

2.4 Numerical results

In this section, we want to focus on the S-FETI method, by testing the different implementations and changes presented in this chapter. The objective is to see the comparative advantages or disadvantages that can show each one of them.

To begin with, we will solve the Poisson problem in 3D with Dirichlet condition in a part of the boundary. Let $\Omega \subseteq \mathbb{R}^3$ be a bounded domain, let also $\partial\Omega_D \subseteq \partial\Omega$ be a part of the boundary where Dirichlet conditions will be imposed. Given $f \in L^2(\Omega)$, the problem is

Find $u \in H^1(\Omega)$ *such that*

$$\begin{cases} -\nu\Delta u &= f \quad in \quad \Omega \\ u &= 0 \quad on \quad \partial\Omega_D \\ \dfrac{\partial u}{\partial n} &= 0 \quad on \quad \partial\Omega \setminus \Omega_D \end{cases}$$

with $\nu \in \mathbb{R}^+$, a parameter that characterizes the medium or material and therefore will produce the differences in the local stiffness matrices of the subdomains when a partition of Ω is created.

Let $V := \{v \in H^1(\Omega) : v|_{\partial\Omega_D} = 0\}$. Then, the weak or variational formulation of previous problem is

Find $u \in V$ *such that*

$$\nu \int_\Omega \nabla u \cdot \nabla v dx = \int_\Omega f v, \quad \forall v \in V$$

This problem is discretized using tri-linear $\mathbb{Q}^1$ finite elements functions, leading to the known global system of equations

$$Kx = f$$

2.4.1 S-FETI basics

Using the system arising from the finite element method for the Poisson problem, we will start by revisiting some of the numerical results shown in [38]. We will add some small differences that are considered to be part of our basic starting S-FETI algorithm when doing the comparative performance of the different implementations.

Regarding the method itself, the local interface subdivision will be the default, mainly because we want to reduce to a minimum the number of iterations by augmenting the number of search directions created at each step.

Also, a small variation will be used in the detection of the kernels and the solutions of local problems. Usually, the PARDISO solver ([57],[56]) for the sparse local solutions is used in this type of implementations, but in our case, we will change this to the use of a different solver, the one called DISSECTION Solver. This different solver allows a more robust computation of the kernels and recently it has shown an improved general performance for double and quadruple computations, compared to the usual Intel PARDISO implementations, see [78].

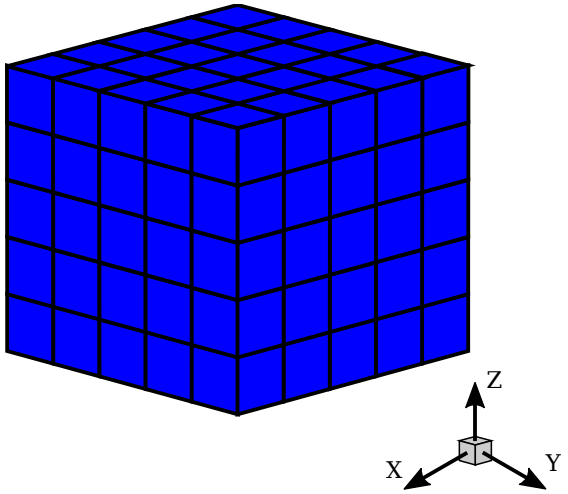

Figure 2.7 – Cube with 125 subdomains.

S-FETI decomposition	Total Search Directions	Iterations
Local subdomain	1125	9
Local interface	6356	4

Table 2.3 – Difference in subdomain division versus local interface division

In this first case, for the 3D Poisson problem, we model the cube $\Omega = [0,1]^3$, with zero Dirichlet boundary in $\partial\Omega_D = \{(x,y,z) \in \partial\Omega : x = 0 \vee x = 1\}$, also $f = 1$.

We divide the cube into 125 smaller cubes, and define $\nu = 10^0$ for the whole domain, see Figure 2.7. Then each subdomain is divided into 75 thousand elements approximately,

We consider the same stopping criteria as the previous chapter, that is, the global relative error and the norm of the jump of relative solutions across the interfaces. In some cases precision measures will be performed, if that is not the case, the criterion will be to stop when both errors are less than 10^{-6}.

We consider the *Superlumped* scaling for the projection, i.e $P = P_A = I - AG(G^TAG)^{-1}G^T$ with A as in Equation (2.17), and the consistent weighted Dirichlet preconditioner as the global preconditioner.

The results are noted in Table 2.3. We can see here that the number of

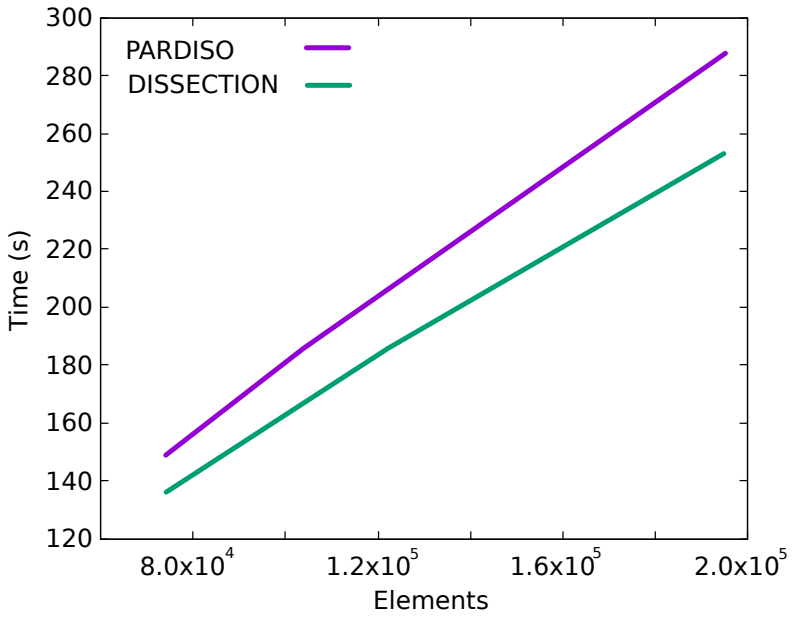

Figure 2.8 – Time of Pardiso vs Dissection for different element number in each subdomain.

iterations is greatly reduced, but at the cost of also adding the extra search directions. This change tries to reduce the impact of communications between processes which is usually the bottleneck in parallel algorithms.

In the next case, if we use the local interface subdivision that generates more search directions per iteration, the difference between solvers can be exposed. The number of local solutions is augmented and therefore the total time cost will depend more on the solver than in the regular S-FETI or simply FETI.

The following tests, as well as all the time performance test of this chapter, were done with a `Fortran-MPI` implementation, on a machine SGI UV 2000 with 32 CPUs Intel Xeon 64 bits EvyBridge E4650 of 10 cores each one, with a frequency of up to 2.4 GHz.

In the graphic of Figure 2.8 we can see that independent of the size problem, the total time is reduced when using the Dissection solver. However, this speedup has an extra memory cost. From now on, both local interface subdivision and Dissection will be the default for the following S-FETI test.

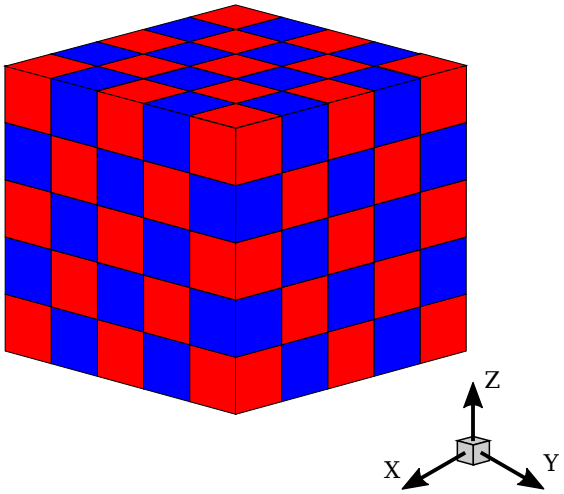

Figure 2.9 – Checkerboard cube with 125 subdomains.

2.4.2 Decomposition of $W^T FW$

Following the propositions of section 2.2, in this part, we will compare the different strategies to sort the search directions generated in S-FETI. We can name both Cholesky and Eigendecomposition of the matrix $W^T FW$.

For this problem, we use the same cube as before, with 125 smaller cubes. We reproduce a slightly heterogeneous case, where we change the parameter ν to be 10^0 and 10^3 in a checkerboard configuration, as seen in Figure 2.9; and we use the local interface subdivision for the S-FETI method.

In this configuration, what we want is to produce a "high enough" number of search directions that can allow us to see the differences between sorting methods and the sensibility to the parameter that represents the zero value ε. In this process are introduced new sources of numerical instability. As a first approach, although very time costly, the use of quadruple precision in the decomposition process may improve the quality of the algorithm and possibly reduce the numerical noise. We test for different values of ε, and we check the number of directions kept as well as the total iterations.

In Table 2.4 we see that as the parameter ε is closer to zero, the number of directions saved augments. Thanks to the full reorthogonalization we also gain in iteration numbers, but the extra directions used are not entirely the ones that we hope to capture with this method. This is shown in the minimal error column,

ε	Implementation	Search directions per iteration	Minimal error	Iterations to error
10^{-4}	QUAD	1247	$6.0537x10^{-5}$	8
10^{-4}	DOUBLE	1245	$9.0064x10^{-5}$	8
10^{-6}	QUAD	1356	$1.7085x10^{-3}$	6
10^{-6}	DOUBLE	1356	$1.7131x10^{-3}$	6
10^{-10}	QUAD	1498	$6.4209x10^{-3}$	5
10^{-10}	DOUBLE	1494	$1.6876x10^{-3}$	5

Table 2.4 – Quadruple vs Double comparison for the Cholesky decomposition of $W^T FW$.

where bigger values are obtained if we use more of them. Also, a small gain in precision is obtained when we use the quadruple precision. However, the behaviour of both implementations is the same, and the cost of the quadruple computations is much more elevated. Hence, we will perform only double implementations with the already named value for ε.

A significant more precise method (smaller minimal error) is obtained when we do not consider values of ε too close to zero. Values that may be within the limits of just noise, for example $\varepsilon = 10^{-4}$ are a much better alternative than $\varepsilon = 10^{-6}$ or any value smaller than that.

Remark: The solver Dissection also comes with a quadruple implementation for solving the local problems and doing the computation of the kernel. However, there were no significant advantages in this context that could suggest the use of this still expensive new implementation. Simple Jacobi one-step improvement (iterative refinement) used to augment the accuracy of the local solver are enough to obtain the same results as a complete quadruple computation performed by Dissection.

Next, we want to compare the performance of the alternative eigenvalue decomposition. From here on, only double precision implementations will be

ε	Implementation	Search directions per iteration	Minimal error	Iterations to error
10^{-4}	CHOLESKY	1245	$9.0064x10^{-5}$	8
10^{-4}	EIGEN	1183	$1.7590x10^{-8}$	11
10^{-6}	CHOLESKY	1356	$1.7131x10^{-3}$	6
10^{-6}	EIGEN	1299	$1.7082x10^{-5}$	8
10^{-10}	CHOLESKY	1494	$1.6876x10^{-3}$	5
10^{-10}	EIGEN	1474	$2.2169x10^{-3}$	5

Table 2.5 – Iterations/Search Directions results for different values of ε.

considered.

One of the advantages expected in changing the decomposition is to make the method less sensitive to threshold selection. In this case, less sensitive to $\varepsilon > 0$. This parameter is used to identify linear dependency and also to diminish the number of saved directions without loosing in performance.

When using the Cholesky decomposition, we can add an extra parameter that can recognize when a gap between the decreasing pivots is too big. This gap is an indicator that we may enter into the limits of the noise directions. As consequence this decomposition is somehow more robust, as we avoid the noise at the cost of eliminating some other maybe useful directions. In any case, this lost vectors are usually recovered in later iterations.

For the Eigenvalue decomposition, we have also added an extra parameter to limit the number of directions saved. It works after the computation with max tolerance (given by the machine precision) of the eigenvalue algorithm. Then it reduces the stored directions to the ones associated to the eigenvalues that are bigger than this new tolerance ε. This way of sorting directions, is a much more stable algorithm than the previous Cholesky decomposition.

In Table 2.5 we show the number of directions stored with each zero value until max precision for each method is achieved.

Here we can see that bigger values of ε reduce the number of directions kept. The convergence is still assured, as the critical information is in the

directions associated with the bigger eigenvalues. The Eigendecomposition, in this case, is less sensitive to variations in the threshold values. In all the cases we see a reduction in both the number of directions kept and the number of total iterations, meaning that at a small cost (in parallel computation is almost negligible) we can have a method with less memory charge, less total iterations, and improved precision.

We also insist on the fact that a much more precise method is obtained when using values of ε not so close to zero. Just $\varepsilon = 10^{-4}$ is more than enough to reduce the minimal error and at the same time reduce the stored directions.

2.4.3 S-FETI with sparse storage

In this section, we want to test the new sparse storage implementation for the S-FETI method, regarding both time and memory usage.

The model problem is again the 3D Poisson equations, solved in the cube $\Omega = [0,1]^3$, with the same boundary conditions and source as in the previous examples.

We also use 125 subdomains with local interface subdivisions, to have at least the same amount of search directions as the previous case. The parameter ν will represent the homogeneous case, meaning that we will have one material as shown in Figure 2.7. The time, memory and accuracy measures will be more important, rather than the number of iterations since this term should not change between the two sparse implementations, and we hope to have similar results between the sparse and the full versions.

We start by testing the speed and precision achieved by the two versions of the SPARSE-S-FETI method. The first with a straightforward implementation when operating with the sparse blocks and the second one with optimizations in the memory allocation. We use BLAS3 routines, and we reduce the bottleneck of doing several matrix-vector operations by doing just one matrix-matrix product. We also change the full reorthogonalization process by making only one process to orthogonalize each saved block and then sending the information to the rest of the processes, see the last point in section 2.3.3.

The results of memory and time are shown in Table 2.6, where we measure

Method	Elements / Subdomain	Time (s)	Max local memory (Gb)
SPARSE-S-FETI	103823	571.1	4.91
SPARSE-OPT-S-FETI	103823	289.4	4.63
SPARSE-S-FETI	148877	648.1	6.57
SPARSE-OPT-S-FETI	148877	339.8	6.13

Table 2.6 – SPARSE vs SPARSE-OPT

the total time used and also the maximum memory used by each process. Here, we can see a speedup of roughly two times for both examples with 100 and 150 thousand elements approximately in each subdomain. We did not add the expected results concerning number of iterations and precision, where there is no difference between the two implementations. We see that no precision is lost, we have improved in memory usage and also the time was reduced, so from here on, the second implementation is the one to be used.

The objective of the sparse storage implementation is to keep a memory usage controlled to be able to use this method for the bigger applications. In this part, we will compare the use of the SPARSE-OPT-S-FETI implementation versus the regular FULL one.

In the Figure 2.10 we can see the maximum memory usage between processes for both methods. We have used two different number of elements per subdomain. Since we are interested in memory performance, we will force to save all the search directions, with no regarding whatsoever on convergence; this is because we want an upper estimate in terms of memory versus the number of iterations. In this example, we are using a number of 2072 directions in each iteration.

For the FULL-S-FETI method, we have a linear relation between the number of iterations (starting from 2) and the memory usage, or in this case, the number of search directions stored (every vector has the same size). For the sparse storage, this iteration/memory relation is in general quadratic. However, this relation is between the number of iterations and the size of the coefficient matrices stored.

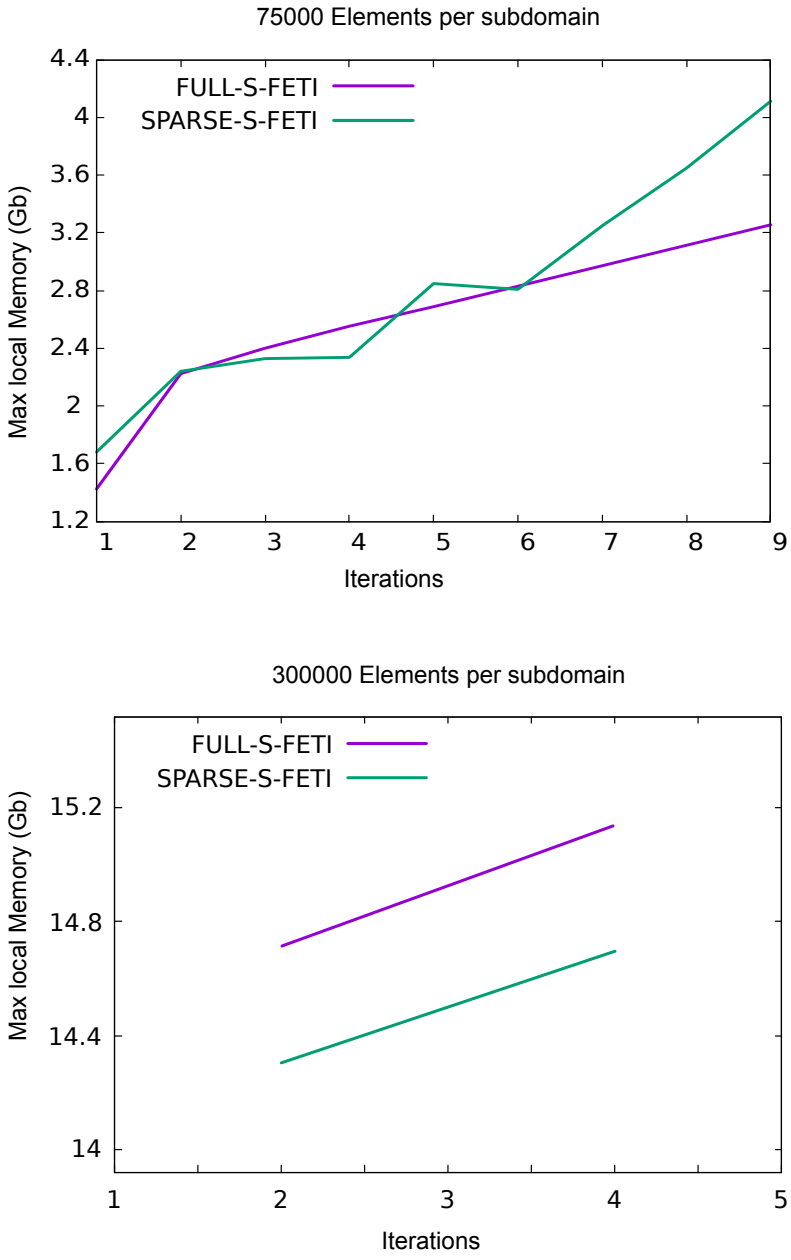

Figure 2.10 – Max local memory usage in cube problem for 75 (up) and 300 (down) thousand elements per subdomain.

Method	$\frac{\text{Elements}}{\text{Subdomain}}$	Iterations	Time (s)	Precision
FULL-S-FETI	103823	11	620.1	$4.3295x10^{-5}$
SPARSE-OPT-S-FETI	103823	11	624.8	$2.0870x10^{-4}$
FULL-S-FETI	148877	11	755.1	$1.9688x10^{-4}$
SPARSE-OPT-S-FETI	148877	11	756.9	$1.8155x10^{-4}$

Table 2.7 – FULL vs SPARSE-OPT

These matrices are dense, but much smaller than a regular full block of search directions and are also independent of the size of the mesh (the size of the complete search vectors). In any case, there is still an increasing linear relation between the sparse stored directions (Z_p to recall) and the iterations, but as explained previously, the size of them is less than the full block, and it depends on the neighbor connectivity of each subdomain.

These results show that the difference between the SPARSE storage and the FULL implementation is more evident as we augment the number of elements in each subdomain. In the first case, the FULL version is a better option, but as we get close to sizes of around 300 thousand elements in each subdomain, the need of the sparse storage is more evident if we want to be able to solve this and other larger problems.

The following test to do, once the memory advantages of the sparse storage were established, is a time consumption test between the SPARSE-OPT-S-FETI and regular method to see the cost of all the extra computations. The problem to solve is the same checkerboard configuration as the previous with two materials. Each subdomain has a parameter of either $\nu_1 = 10^0$ or $\nu_2 = 10^4$. The results are shown in Table 2.7, where we see that the differences in precision and number of iterations are negligible. In the same way, the time difference is small considering the size of the problems, and it is smaller when the size of the problem is increased. We can say then that the new implementation is suitable to test big cases where the memory is a limitation.

2.4.4 General comparison

Finally, we want to test the differences between the two existent preconditioners for S-FETI, the Dirichlet and Lumped. We check for memory, number of iterations and total time of the computations.

In addition to this, we want to compare the sparse and full storage implementations with these preconditioners, as well as the two decompositions for sorting the search direction. The idea is to have a global view that can give us some "rules of 'thumb" of which method should we use in different situations.

- Type of implementation: FULL-S-FETI or SPARSE-OPT-S-FETI
- Preconditioner: Dirichlet or Lumped.
- Decomposition method for $W^T F W$: Cholesky or Eigenvalue.
- Elimination of corner crosspoint interfaces: Yes or no.

We added the last point since in practical engineering applications, the corner interfaces are rare and almost nonexistent. This is due to the fact that the domain division are done automatically by a mesh partitioning algorithm, such as METIS ([46]). This algorithm, with a regular Cartesian mesh, does not create the corner interfaces that we usually see in square or cubes division of a domain. To view the difference in term of number of local solves and total search directions, we can "eliminate" these corners and test the method more realistically, with none of the search directions created by the corner interfaces. In our particular case, we lower the number of total search directions per iteration from 2072 to 600.

One of the crucial differences that we want to test are the ones produced by the different preconditioners available, in this case, the consistent weighted Dirichlet and the also consistent Lumped preconditioner. If we look at both definitions in (2.10) and (2.11), we see that one involves the multiplication by the Schur complement, and that implies the solution of a local problem. Then, we can realize that there are differences between them regarding computational and memory cost. In both characteristics the Dirichlet preconditioner is much more expensive; however, we expect a reduced number of iterations for this

Implementation	Precon-ditioner	Decom-position	Max local memory (Gb)	Iterations	Time (s)
FULL	Dirichlet	Cholesky	5.98	7	429.0
FULL	Dirichlet	Eigen	5.93	7	432.4
FULL	Lumped	Cholesky	4.67	8	507.6
FULL	Lumped	Eigen	5.02	11	825.0
SPARSE-OPT	Dirichlet	Cholesky	6.27	7	340.8
SPARSE-OPT	Dirichlet	Eigen	6.32	7	331.4
SPARSE-OPT	Lumped	Cholesky	5.43	8	410.4
SPARSE-OPT	Lumped	Eigen	7.77	11	812.5

Table 2.8 – Comparative between variations of S-FETI *with* corner interfaces. 125 subdomains and 150 thousand elements per subdomain.

preconditioner, because it considers the nodes in the entire subdomains and not only the nodes in the interface.

The test case will be the same as previous, with 125 subdomains in a checkerboard configuration for the ν parameter and with 150 thousand elements per subdomain. In this case, we will perform several computations and measure the time and memory performance of them. Each result will be a different combination of the four variables previously detailed.

In Table 2.8 we show the results obtained. We note that the results are not in the lines of what we expected. This is due to the characteristic of the problem; meaning, a number of subdomains and size of the mesh not high enough. Therefore, we see a reduction in time for the SPARSE-OPT-S-FETI, but with a higher memory usage. In Table 2.9 a remarkable speedup is obtained for this case when using the SPARSE implementation.

The elimination of corner interfaces also produces a reduction in the total time, making these results closer to a real implementation with automatic subdomain partitioning. In terms of memory usage, there is no significant differences between the methods. As said previously, the size of the problem is not big enough to make the difference more clear. In any case, we still recall that

Implementation	Precon-ditioner	Decom-position	Max local memory (Gb)	Iterations	Time (s)
FULL	Dirichlet	Cholesky	5.21	12	119.6
FULL	Dirichlet	Eigen	5.21	12	123.4
FULL	Lumped	Cholesky	4.46	24	366.9
FULL	Lumped	Eigen	4.58	25	390.0
SPARSE-OPT	Dirichlet	Cholesky	4.43	12	59.7
SPARSE-OPT	Dirichlet	Eigen	4.41	12	60.6
SPARSE-OPT	Lumped	Cholesky	4.31	24	158.3
SPARSE-OPT	Lumped	Eigen	4.09	25	171.4

Table 2.9 – Comparative between variations of S-FETI *without* corner interfaces. 125 subdomains and 150 thousand elements per subdomain.

depending on memory availability the use of one or the other is recommended.

2.5 Conclusion

Several implementation changes have been introduced in this chapter, most of them showing an improved performance compared to the primary S-FETI. We added a new way of sorting search directions. The Eigenvalue decomposition has proved to be a more robust option, and at the same time, it reduces the number of directions stored at each iteration. In terms of memory usage, we developed a new sparse storage that allows using this method in bigger applications without losing significant performance. The use of the Dissection solver with its kernel detection algorithm is a good alternative over the more commonly used PARDISO solver. The Lumped preconditioner for this method is an attractive alternative to reduce memory and computation efforts, even if the results shown here are not definitive to say in which cases should be used. Further testing will be done to this preconditioner as part of the future work.

Finally, other changes and comparatives can still be performed, leaving space for improvements to this method and its implementations. This is all part of the future work in this method.

Chapter 3

Block strategies as a preconditioner

In the development of the domain decomposition methods, in particular of the FETI method, there is always a search for faster and more accurate algorithms that can be used in as many problems as possible. In this line, but using a different approach, we will try to use some of the ideas exposed in the previous chapter, notably, the ones related to the development and implementation of S-FETI.

The main advantage shown by the S-FETI method is the construction of an enlarged search space in the Conjugate Gradient algorithm. With this algorithm, the capture of the larger eigenvalues of the FETI operator, even for some hard problems, can be achieved in a few iterations. The price to pay for this is the use of large amounts of memory that can render impractical its use for some big problems, usually found in real applications. In this context, we already developed a way to circumvent this limitation by using a sparse storage of the directions.

In this chapter, we will try to recover the precision and memory use of the original FETI method, improving the convergence ratio by building a new preconditioner based on the enlarged space generated by the S-FETI method. This new preconditioner is a priori more expensive, but hopefully, the faster convergence expected will compensate this, and we may have a useful method for practical use.

This chapter starts with some preliminaries to understand the idea of the

new preconditioned FETI method; later we will show a natural extension of the first part by mixing it with the S-FETI method. We end up with numerical results of both new methods.

3.1 Introduction and preliminaries

The FETI method is based on finding the solution of an interface problem using the Preconditioned Conjugate Gradient Method. This iterative method (CG) has been largely studied over the years. It has lead to different results in its research, and several generalizations were made to it.

Within these generalizations, we can name the Flexible Conjugate Gradient [54], the Multipreconditioned CG [11] and the Method of Conjugate Directions [44]. Although the last one was developed before the CG method, it can still be considered as one of the generalizations of it.

We will now show in detail, two of these methods and their relation to previous FETI methods. These relations will allow understanding the roots of the new FETI methods proposed in this chapter and justify their convergence.

3.1.1 Method of Conjugate Directions

We consider the solution of the general system

$$F\lambda = d \tag{3.1}$$

with F a symmetric positive definite matrix.

This method, shown in [44] can be considered as a more general method than the CG method.

In here, the search directions

$$w_1, w_2, \ldots$$

are selected to be mutually conjugate, that is

$$(w_i, Fw_j) = 0, \qquad \forall i \neq j$$

and no further restriction is imposed in them. The norm considered here is the one induced by F, because, as already said, is a symmetric positive definitive operator.

The summary is presented in Algorithm 11.

Algorithm 11 Method of Conjugate Directions

Initialization
Choose an approximation vector λ_0
$g_0 = F\lambda_0 - d$
Choose any non-zero vector w_0
loop $p = 0, 1, 2, \ldots$ until convergence
    $\rho_p = -\dfrac{(w_p, g_p)}{(w_p, Fw_p)}$
    $\lambda_{p+1} = \lambda_p + \rho_p w_p$
    $g_{p+1} = g_p + \rho_p F w_p$
    Choose a direction w_{p+1} such that $(w_{p+1}, Fw_j) = 0$, $\forall j = 0, \ldots, p$
end loop

We can see that the CD-method is not precise, in the sense that no formula is given for choosing the search directions. Depending on how are built these directions, different methods can be obtained (including the CG method). Later we will make explicit this definition in order to formulate the new FETI method.

The basic properties of this method are given by the following theorems

Theorem 3.1. *The search directions $w_1, w_2, \ldots$ are mutually conjugate. The residual vector g_p is orthogonal to $w_1, w_2, \ldots, w_{p-1}$. The inner product of w_p with the residuals $g_0, \ldots, g_p$ is the same. All this can be written as*

$$(w_i, Fw_j) = 0, \qquad i \neq j \tag{3.2}$$

$$(w_i, g_p) = 0, \qquad i = 0, 1, \ldots p-1 \tag{3.3}$$

$$(w_p, g_0) = (w_p, g_1) = \cdots = (w_p, g_p) \tag{3.4}$$

The scalar ρ_p can be given by the formula

$$\rho_p = -\frac{(w_p, g_0)}{(w_p, Fw_p)} \tag{3.5}$$

Proof. From the last line in Algorithm 11 we have directly (3.2). Next, using the update of the gradient, we have that

$$(w_i, g_{p+1}) = (w_i, g_p) + \rho_p (w_i, Fw_p)$$

if $i = p$ then, by the definition of ρ_p we know that

$$(w_i, g_{p+1}) = 0$$

Moreover, from (3.2) we have

$$(w_i, g_{p+1}) = (w_i, g_p)$$

Equations (3.3) and (3.4) follow from the last two relations. The last equation (3.5) follows from (3.4) and (3.2). □

As for the convergence of the method we have the following theorem

Theorem 3.2. *The CD-method is a p-step method ($p \leq n$) in the sense that at the p-iteration the estimate λ_p is the solution λ*

Proof. Let p be the first integer, such that

$$y_0 = \lambda - \lambda_0 \in span\{w_0, \ldots, w_{p-1}\}$$

Clearly $m \leq n$ since the vectors $w_0, w_1, \ldots$ are linearly independent. On the other hand, we may choose scalars $\alpha_0, \ldots, \alpha_{p-1}$ such that

$$y_0 = \alpha_0 w_0 + \cdots + \alpha_{p-1} w_{p-1}$$

Hence,

$$\lambda = \lambda_0 + \alpha_0 w_0 + \cdots + \alpha_{p-1} w_{p-1}$$

Moreover

$$g_0 = F\lambda_0 - d = F(\lambda_0 - \lambda) = -\alpha_0 F w_0 - \cdots - \alpha_{p-1} F w_{p-1}$$

Computing the inner product

$$\begin{aligned}(w_i, g_0) &= -\alpha_0(w_i, Fw_0) - \cdots - \alpha_{p-1}(w_i, Fw_{p-1}) \\ &= -\alpha_i(w_i, Fw_i)\end{aligned}$$

an using (3.5) we imply that $\alpha_i = \rho_i$ and hence $\lambda = \lambda_m$. □

Finally, we can see that the approximated solution improves with the number of iterations and the error is minimized at each step.

Theorem 3.3. *Let λ be the solution of (3.1) and f be the error function*

$$f(v) = \|\lambda - v\|_F = (\lambda - v, F(\lambda - v)) = (v, Fv) - 2(v, d) + (\lambda, d)$$

then, the point λ_p minimizes $f(v)$ on the line $v = \lambda_{p-1} + \alpha w_{p-1}$. In fact, the point λ_p minimizes $f(v)$ on the space

$$\lambda_0 + span\{w_0, \ldots, w_{p-1}\}$$

and this space contains the vectors, $\lambda_0, \ldots, \lambda_p$

Proof. If w is a search direction, then we have

$$f(v + \alpha w) = f(v) - 2\alpha(w, g) + \alpha^2(v, Fv)$$

where

$$g = d - Fv = F(\lambda - v)$$

Considered as a function of α, the function $f(v + \alpha w)$ has a minimum value at $\alpha = \rho$, where

$$\rho = \frac{(w, g)}{(w, Fw)}$$

Comparing this with the definition of ρ_p in the algorithm, we have the minimization result.

For the last part, let us consider the point $v \in \lambda + span\{w_O, \ldots, w_{p-1}\}$, then we have

$$v = \lambda_0 + \alpha_0 w_0 + \cdots + \alpha_{p-1} w_{p-1}$$

applying f

$$f(v) = f(\lambda_0) - \sum_{i=0}^{p-1} [2\alpha_i(w_i, g_0) - \alpha_i^2(w_i, Fw_i)]$$

At its minimum we have

$$\alpha_i = \frac{(w_i, g_0)}{(w_i, Fw_i)}$$

and so $\alpha_i = -\rho_i$, for all $i = 0, \ldots, p-1$. Then is clear that the minimum point is λ_p. □

Considering the previous theorems, as long as we do the full reconjugation, we have the liberty to build the search directions in any form we want.

3.1.2 Flexible Conjugate Gradient

As a continuation of the previous Conjugate Directions method, and within the Conjugate Gradient framework, we can recall the General CG or Flexible CG (as denoted in [54]). We consider the problem (3.1), solved with the Preconditioned CG method. The idea is to add at each iteration of the CG method a preconditioner step with the aim of accelerating the convergence by using suitable search directions. Usually this is achieved by finding the solution of a problem of the type

$$Bx = y$$

where we want that the conditioning of the matrix $B^{-1}F$ to be smaller than the conditioning of F.

In PCG, the preconditioner B^{-1} is symmetric positive-definite and fixed. This allows keeping the convergence properties of CG. In the Flexible-CG we relax this conditions and change the preconditioning step to a general mapping

$$g \to \mathcal{B}[g]$$

this mapping also tries to approximate the inverse of F in order to improve the convergence. This more general mapping does not need to have an explicit form as it can be a recursive non-linear mapping. We will just consider it as a general procedure to approximate the inverse of F.

With this preconditioning procedure and using a full reorthogonalization process, we can define the algorithm for the Flexible CG method in Algorithm 12. This algorithm makes explicit the construction of the F-conjugate search direc-

Algorithm 12 Flexible Conjugate Gradient Method

Initialization
Choose an approximation vector λ_0
$g_0 = F\lambda_0 - d$
$w_0 = \mathcal{B}[g_0]$
loop $p = 0, 1, 2, \ldots$ until convergence
    $\rho_p = -\dfrac{(w_p, g_p)}{(w_p, Fw_p)}$
    $\lambda_{p+1} = \lambda_p + \rho_p w_p$
    $g_{p+1} = g_p + \rho_p F w_p$
    $w_{p+1} = \mathcal{B}[g_p]$
    for $i = 0$ to p **do**
        $\gamma = -\dfrac{(w_i, Fw_{p+1})}{(w_i, Fw_i)}$
        $w_{p+1} = w_{p+1} + \gamma w_i$
    end for
end loop

tions by introducing a preconditioning procedure and applying the Modified Gram-Schmidt procedure to build the new directions.

It is also essentially the same CG algorithm used by FETI, the main difference lies in the preconditioner. If we consider $\mathcal{B}$ as the Dirichlet or Lumped preconditioner, we can recover the usual FETI method.

Remark: The detailed convergence properties of this algorithm, including bounds for the error, depend on the definition of the mapping $\mathcal{B}$, where conditions on coercivity and boundness are assumed [7]. In our case, and to keep the mapping as general as possible, we will only consider the previously shown convergence of the CD-method to validate our new FETI method with recursive preconditioner.

3.2 FETI with recursive preconditioner

The Simultaneous-FETI method can be considered as a Multipreconditioned CG method [38], this is because several directions are created in the preconditioning step thanks to the additive nature of the Dirichlet preconditioner. Closely related to this method, we have the Flexible CG in which recursive or non-linear preconditioners can be used at the cost of doing a full reorthogonalization of the search directions (truncated orthogonalization are also possible.). In the FETI framework, there are no practical problems in the use of these extra computations, since we are forced to do them to keep a good convergence (See [66]).

Following this idea of being forced to do a full reorthogonalization. Let us consider the general method of Conjugate Directions. With it, we have total liberty in the creation of the search directions, as long as they are conjugate between them, more precisely F-conjugate (F being the FETI operator, which is symmetric positive definite).

This last part made us think of "mixing" these methods. We consider some special preconditioner that changes from iteration to iteration, and that gives us a better approximation than the usual Dirichlet in the FETI method. It is built straightforwardly from the one in S-FETI where a more accurate approximation is obtained at every iteration. Even if we are not exactly in the conditions to consider this new method as a Flexible CG method, we can see it as a Conjugate Directions method, where the convergence is assured.

Now we will show the details of this new method, and later its extension to an "in between" method, if we consider the S-FETI method and this new one.

3.2.1 One direction from S-FETI block

We derive this new method starting with the FETI basic algorithm, and modifying the preconditioning step

$$w_p = \tilde{F}^{-1} g_p \tag{3.6}$$

with $\tilde{F}^{-1}$ being any suited preconditioner (mainly the Dirichlet or Lumped one). Even though the multiplication for this precondition matrix is not computed

directly, we can still consider this step as simple and not so expensive (details about this have been discussed in previous chapters), the main difference will be in this step, where a more elaborated preconditioner will be constructed.

We change this multiplication by the Dirichlet or Lumped operator for the definition of a new procedure (or mapping) $\mathcal{B}$, based on the block construction done in S-FETI.

The idea behind this is to use the minimization process of the CG method applied in the S-FETI method, where we construct a larger search space, based on the combination of local terms. Let us recall this process in Algorithm 13 where after the computation of the search directions block, we obtain the optimization parameters and update the solution In this new method, we will consider

Algorithm 13 Minimization process in S-FETI

1: ⋮
2: $\rho_p = -W_p^T g_p$
3: $\lambda_{p+1} = \lambda_p + W_p \rho_p$
4: ⋮

the update vector $W_p\rho_p$ as our unique search direction. It contains all the information from the non local preconditioner, and it can help speed the process of capture of the eigenvalues. Meaning that at every iteration of the Conjugate Gradient method, we will define the search direction as

$$w_p = W_p\rho_p = -W_p(W_p^T g_p)$$

The construction of W_p was already explained when we define the S-FETI method in chapter 2 and it involves the solution of several Neumann and Dirichlet problems (if the Dirichlet preconditioner is used), so it is more costly than the regular preconditioner. This extra cost compared to the gain in the convergence will be tested in the numerical examples.

Adding the construction of the block W_p to our new search direction allows to explicit the definition of the procedure $\mathcal{B}[g]$ used as a preconditioner. This mapping is described in Algorithm 14. The notation in this procedure is the same as in chapter 2 so $D^{(s)^{-1}}_{\Gamma^{(sq)}}$ is the local preconditioner partitioned into neighbor

Algorithm 14 Preconditioning procedure $\mathcal{B}$

1: **Input** Residual vector g
2: $Z = [\dots, D^{(s)^{-1}}_{\Gamma^{(sq)}} g, \dots]$ ▷ Local Dirichlet preconditioner
3: $W = PZ$ ▷ FETI Projection
4: $NLL^T N^T = W^T F W$ ▷ Cholesky decomposition
5: $W = WNL^{-T}$
6: $\rho = -W^T g$
7: $w = W\rho$
8: **Return** $\mathcal{B}[g] \leftarrow w$

interfaces and the matrix N is the permutation matrix coming from the Cholesky factorization applied to $W^T F W$.

Basically what we are doing is similar to what we do in any preconditioned CG algorithm, that is, a process from which we take the gradient, and we apply a particular operator to build a search direction that approximates to the actual error $F^{-1} g_p$, at the p iteration. The difference, in this case, is that the search direction do not come from a direct multiplication by a fixed matrix, but from a more complicated process that in the end makes the new direction to depend on the previous gradient.

The final method is shown in Algorithm 15. What we describe is a modification of the FETI method (using the Flexible CG instead of classical CG), in which we change the computation of the preconditioner with the procedure coming from S-FETI that allows defining $\mathcal{B}[\cdot]$ in Algorithm 14. The definition of the projection operator P, the matrix A, the matrices G of the FETI coarse space and the block of vectors of the kernel R are defined in the same way as previous methods (FETI-1LM or S-FETI).

We hope that with this new method we are going to recover the accuracy and memory usage of the basic FETI method. We store one direction per iteration and at the same time we diminish the roundoff effects of the extra sources of numerical noise that can be found in the S-FETI method. As for the advantages, the number of iterations will be reduced, due to the improved way to capture the largest eigenvalues, and we expect this will be enough to compensate for the extra cost on the mapping of the preconditioner.

Algorithm 15 FETI block to one direction Algorithm

Initialization
$\lambda_0 = AG[G^T AG]^{-1}(-R^T c)$
$g_0 = (F\lambda_0 - d)$
$w_0 = \mathcal{B}[P^T g_0]$
loop Iterate $p = 0, 1, 2, \ldots$ until convergence
    $\alpha_p = -\dfrac{(g_p, w_p)}{(w_p, Fw_p)}$
    $\lambda_{p+1} = \lambda_p + \alpha_p w_p$
    $g_{p+1} = g_p + \alpha_p F w_p$
    $w_{p+1} = \mathcal{B}[P^T g_{p+1}]$
    for $i = 0$ to p **do**
        $\gamma_i = -\dfrac{(w_i, Fw_{p+1})}{(w_i, Fw_i)}$
        $w_{p+1} = w_{p+1} + \gamma_i w_i$
        $Fw_{p+1} = Fw_{p+1} + \gamma_i Fw_i$
    end for
end loop

Cost of the method

Considering the cost of the original FETI method and the one of the S-FETI method, we can give a comparative estimate of the performance of this method against the original FETI at each iteration.

We know by its definition that all we need to compare is the preconditioning step in FETI versus the preconditioning procedure defined in Algorithm 14. We note that this is because the procedure $\mathcal{B}[\cdot]$ will also give the vector Fw_p at the end of the procedure since its value corresponds to $FW\rho$. Hence no extra multiplication by F is needed after this part.

Based on the analysis of the cost in S-FETI, we can say that the most expensive part here is the computation of the block FW. This includes a projection and several forward-backward computations, but we can use the same strategies to minimize the computations as done in S-FETI. For each iteration we only add to each process the workload of $\mathcal{N}$ Neumann solves, with $\mathcal{N}$ being either $neighbors + 1$ or $2 \times neighbors$ depending on the construction of the blocks. The projection is also done using low-rank corrections. For details in the cost of

S-FETI and thus this preconditioning procedure, see subsection 2.1.5 in the previous chapter.

Finally, we can say that the cost of each iteration is approximately an iteration of S-FETI (which at the same time has a few extra cost compared to FETI) plus an iteration of FETI without the Neumann solution. We can not estimate the number of iterations needed for convergence (hopefully much less than FETI), so we will analyze in the numerical examples the real performance of this method. We will test different ways to compute the block W used in the mapping $\mathcal{B}[\cdot]$, namely the diagonalization process in the treatment of the term $W^T F W$ and the use of the Lumped preconditioner.

3.2.2 Linear combination of directions from block

In this section, we will extend the previous idea of using the added combination $W\rho$ built in the S-FETI method as a single direction, to an "in between" method. We will save in addition to the unique previous directions a reduced number of vectors containing the information of the block of search directions generated by S-FETI. In every iteration, this number will be limited between 1 and the total number of linear independent search directions N_S.

In order to achieve this, we will proceed similarly as before, only this time, instead of using the FETI method as a base, we will use the S-FETI method and we will change the directions stored after the update of the gradient at the end of each iteration.

First, we will recall the construction of the block of directions W in S-FETI, but this time computed with the eigenvalues decomposition. This will allow to sort the total independent search directions in decreasing order of importance. In this case, starting from the gradient we have that

Algorithm 16 Block construction in S-FETI

1: ⋮
2: $W_p = [\dots, D_{\Gamma^{sq}}^{(s)^{-1}} g_p, \dots]$
3: $EDE^T = W_p^T F W_p$
4: $W_p = W_p E D^{-1/2^T}$
5: ⋮

From here we know that the matrix W_p is formed by columns of normalized vectors that are sorted in decreasing order from the ones that capture the bigger eigenvalues of the FETI operator to the ones that represent the smaller ones.

Is from this characteristic that we can use and save the more relevant directions. In the previous case, we added each column to build a unique search direction

$$w_p = W_p \rho = \left[\begin{array}{c|c|c} & & \\ W_p^1 & \dots & W_p^{N_S} \\ & & \end{array}\right] \begin{bmatrix} \rho_1 \\ \vdots \\ \rho_{N_S} \end{bmatrix} = \sum_i \rho_i W_p^i$$

with ρ the vector of minimization parameters $\rho = -W_p g_p$. This time we split this sum, and we recover a new block of search directions. This block is smaller than the previous one, but to try to keep all the information we add an extra direction, the vector $W_p \rho$. This implies the construction of yet another block $\tilde{W}_p$ defined by

$$\tilde{W}_p = \left[\begin{array}{c|c|c|c} & & & \\ W_p \rho & W_p^1 & \dots & W_p^M \\ & & & \end{array}\right]$$

Where M is a number that depends on the number of columns, fixed by the user, that we will study in detail in the numerical tests. What we try to accomplish here is the reduction of the relevant information to a limited number of search directions, improving convergence but with controlled memory usage, at expenses of the extra computations needed to build the original W_p.

Like the previous method, the definition of the FETI projector P is as usual. In Algorithm 17 we described the final algorithm, built from the basic S-FETI, but changing the construction of the stored block directions. The definition of

the projection and all the matrices to compute λ_0 are defined as in previous FETI methods (see section 1.1.2).

Algorithm 17 FETI block with reduced saved directions algorithm

Initialization
$\lambda_0 = AG[G^T AG]^{-1}(-R^T c)$
$g_0 = (F\lambda_0 - d)$
loop Iterate $p = 0, 1, 2, ...$ until convergence
  $Z = \left[\ldots, D^{(s)^{-1}}_{\Gamma^{(sq)}} g_p, \ldots\right]$ ▷ Local Dirichlet preconditioner
  $W = PZ$ ▷ FETI Projection
  for $i = 0$ to p **do**
    $\Gamma_i = -W_i^T FW$
    $W = W + \Gamma_i W_i$
    $FW = FW + \Gamma_i FW_i$
  end for
  $EDE^T = W^T FW$
  $W = WED^{-1/2}$
  $\rho = -W^T g_p$
  $\lambda_{p+1} = \lambda_p + W\rho$
  $g_{p+1} = g_p + FW\rho$
  $W_p = \left[W\rho, \rho^1 W^1, \ldots, \rho^N W^M\right]$
  $FW_p = \left[FW\rho, \rho^1 FW^1, \ldots, \rho^N FW^M\right]$
end loop

Since the base of this algorithm is S-FETI, its convergence is assured. This fact comes from the Theorem 2.1 where the result is independent of the preconditioner or the construction of the block of search directions, as long as they are F-conjugate, which clearly is our case.

Cost of the method

The cost of this algorithm, at every step, is derived directly from the cost of S-FETI in subsection 2.1.5. We do the same construction to obtain the block of search directions. The only difference is the size of the stored vectors that may be reduced as much as we want, however in the next section we will try to find the optimal value of needed stored directions (if there is one).

3.3 Numerical results

The methods shown in this chapter can be considered as variations of the S-FETI algorithm, so the characteristics of them in terms of problems that it can solve are very similar to the original S-FETI. Is for this reason that a simple test case will be considered next. We want to make a first comparative between these methods.

The Poisson problem or any other simple case, solved in 3D will allow us to do this first comparative of both methods presented in this chapter.

We recall the variational formulation of this problem for some parameter $\nu \in \mathbb{R}^+$ and some bounded $\Omega \in \mathbb{R}^3$.

Find $u \in V$ *such that*

$$\nu \int_\Omega \nabla u \cdot \nabla v dx = \int_\Omega f v, \quad \forall v \in V$$

where

$$V := \left\{ v \in H^1(\Omega) : v|_{\partial\Omega_D} = 0 \right\}$$

and $\partial\Omega_D$ is the section of the boundary where we impose $u = 0$.

A finite element discretization, using tri-linear $\mathbb{Q}^1$ functions is used to approximate the solution of this problem.

We finish the problem by setting the characteristics of our model, which in this case are a cube domain $\Omega = [0,1]^3$ with boundary conditions in $\partial\Omega_D = \{x = 0\} \cup \{x = 1\}$, the value of the source is $f = 1$ and the parameter ν will have an homogeneous value along the domain.

The cube will be divided in 125 smaller cubes of equal size, with variable number of elements in each one, and the stopping criteria are the same as previous chapters.

3.3.1 Storage of single direction

The first method to test will be the one where all the search directions are reduced to a single one. In simpler terms, we reduce the S-FETI approach to a FETI method with a more expensive preconditioner.

Method	$\frac{\text{Elements}}{\text{Subdomains}}$	Iterations
FETI-1LM	74088	17
S-FETI	74088	7
SINGLE_DIR-S-FETI	74088	14
FETI-1LM	195112	18
S-FETI	195112	7
SINGLE_DIR-S-FETI	195112	15

Table 3.1 – Convergence comparative.

In terms of memory, we recover the same usage as the original FETI-1LM, so no test related to memory will be performed.

Our first and main approach will be to asset the number of iterations needed for convergence in this method and compare it to the original FETI-1LM. Using the already described configuration, with $\nu = 10^{-3}$. we can see the results in Table 3.1.

Regarding iteration numbers, we can see a reduction in the total number of them compared to FETI-1LM, but no reduction against S-FETI. This goes with the lines of what we expected, meaning that an expensive preconditioner can improve the FETI-1LM method, but the information lost when storing the constructed directions as a single vector will reduce the convergence compared to S-FETI.

In any case, the excessive cost of this approach does not compensate for the good results shown when reducing iterations of FETI-1LM.

3.3.2 Storage of reduced directions

We continue with the extension of the previous method, so in this case, we change the reduction of the search directions from a single vector to a small number of them. These vectors are computed in descending order of importance, so we want to keep a minimal number of search directions without losing convergence.

We continue to use the same test case, that is, the Poisson problem in the

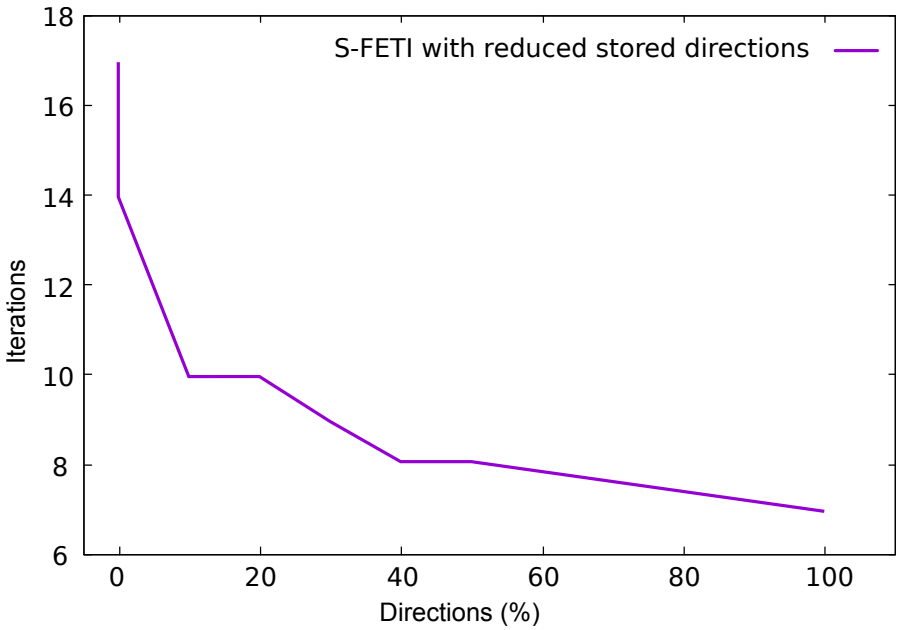

Figure 3.1 – Iterations versus percentage of computed directions. Example 1

cube with 125 subdomains and fixed 75 thousand elements per subdomain. This is complex enough, mainly due to the number of directions produced in each iteration, so we can expose the differences between this method and the original S-FETI or FETI.

In the definition of this method, we have to choose a fixed number of search directions to keep at every iteration. In that sense, a fixed number M was introduced, but what we actually do in practice is to save a percentage of the independent search directions. This value has a great impact on the method, so we need to do variations of it to try in find a possible optimal value.

In Figure 3.1 we show the behaviour of the method in terms of total iterations versus the percentage of directions stored. Let us recall that the total number of search directions generated, without considering linear independence is 2072. From this we do the first reduction, to later keep a part of this reduced amount.

In this image, we see the behaviour of the method, with a fast initial reduction in the number of iterations, even if we store only a small percentage of directions, such as 10%. Values close to this can be considered as "optimal" for this method if we are looking for memory reduction and higher precision. Depending on

the problem to solve and the machine used, we choose the best suited value for every case.

3.4 Conclusion

The variations of the S-FETI method shown in this chapter, are a priori ideas worth testing, since the storage of the information produced by S-FETI can be an issue. Numerical results show that in the actual conditions of the first variation, even if we obtain an improvement regarding iterations to convergence, precision and memory, there is no gain in total time due to the expensive cost of the implementation used. The second method arises as an alternative to certain particular cases, where at the price of more iterations we reduce the memory usage. In any case, we found important to state these results, to give a source for new development in the S-FETI and other FETI-like domain decomposition methods.

Chapter 4

FETI-2LM with enlarged search space

In this final chapter, we will show another application of the main idea in the S-FETI method, that is, enriching the search space of the FETI algorithm. This time we apply the enhancement to another FETI method shown in this thesis, in particular, the FETI-2LM method.

The FETI-1LM and FETI-2LM methods share similarities in the iterative algorithms used, both of them are Krylov based, the Conjugate Gradient and the ORTHODIR method. This fact gives us the idea of enriching the search space in FETI-2LM, but without expecting the same results as in S-FETI. This is because a big part of the convergence properties of S-FETI come from splitting the preconditioner to isolate the modes that make the convergence slower. Also, the connection of this method with the FETI-Geneo method makes the fast convergence to be expected. In the FETI-2LM case, we do not have these characteristics, so the advantages are not clear.

In the FETI-2LM method we no longer have an optimal preconditioner, nor a method to compare, so we do not expect to have the same speedup as in S-FETI. However, we will have an improvement in the convergence, due to the construction of directions using local properties that are lost in the FETI-2LM method. We will also improve the convergence ratio because the size of the new enriched space is bigger compared to the original one. Finally, in terms of time spent, we will be applying the same implementation strategies as in S-FETI. Hence, we also hope to have a time speedup.

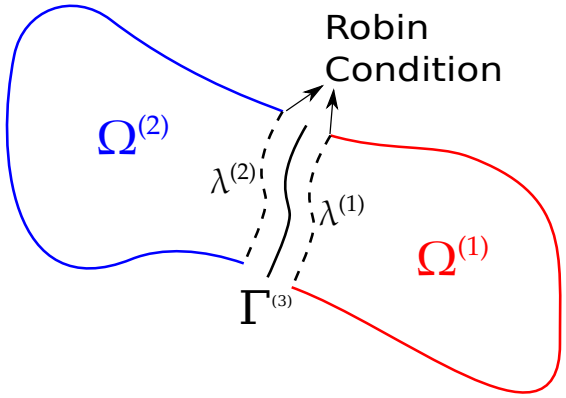

Figure 4.1 – Two subdomains with Robin Interface Condition

We start by recalling the FETI-2LM method, to then pass directly to the formulation of the new Block-2LM method. We end this chapter by showing some numerical examples to test this new method and compared it to the original FETI-2LM.

4.1 Introduction

Most of the elements needed in this chapter were already described throughout the previous chapters. Nevertheless, we want to recapitulate some of the more important ones, to keep clarity on the new method to be constructed.

4.1.1 The FETI-2LM method

This method started as a solver for acoustic problems, first shown in [31], then it was generalized in [67] and [68] as a robust solver for problems coming from any Finite Element discretization of an elliptical partial differential equation.

It is based on the imposition of a Robin condition on the interface, see Figure 4.1, in order to "glue" the solutions of each subdomain. We will recall the basic formulation for a two-subdomain division. The general formulation comes from using the two-subdomain case in each pair of interface edges between neighbor subdomains.

Consider the linear problem, arising from a Finite Element discretization of

a PDE

$$Kx = f$$

We divide the global problem into two subdomains $\Omega^{(1)}$, $\Omega^{(2)}$ and their interface $\Gamma^{(3)}$. Then, the global problem has the structure

$$\begin{bmatrix} K_{ii}^{(1)} & 0 & K_{ib}^{(1)} \\ 0 & K_{ii}^{(2)} & K_{ib}^{(2)} \\ K_{bi}^{(1)} & K_{bi}^{(2)} & K_{bb}^{(1)} + K_{bb}^{(2)} \end{bmatrix} \begin{bmatrix} x_i^{(1)} \\ x_i^{(2)} \\ x_b \end{bmatrix} = \begin{bmatrix} f_i^{(1)} \\ f_i^{(2)} \\ f_b^{(1)} + f_b^{(2)} \end{bmatrix}$$

the subdomain stiffness matrices and right-hand sides are

$$K^{(1)} = \begin{bmatrix} K_{ii}^{(1)} & K_{ib}^{(1)} \\ K_{bi}^{(1)} & K_{bb}^{(1)} \end{bmatrix}, f^{(1)} = \begin{bmatrix} f_i^{(1)} \\ f_b^{(1)} \end{bmatrix} \qquad K^{(2)} = \begin{bmatrix} K_{ii}^{(2)} & K_{ib}^{(2)} \\ K_{bi}^{(2)} & K_{bb}^{(2)} \end{bmatrix}, f^{(2)} = \begin{bmatrix} f_i^{(2)} \\ f_b^{(2)} \end{bmatrix}$$

The FETI-2LM method comes from the imposition of independent generalized Robin condition on the interface $\Gamma^{(3)}$.

The discrete local problems are

$$\begin{bmatrix} K_{ii}^{(s)} & K_{ib}^{(s)} \\ K_{bi}^{(s)} & K_{bb}^{(s)} + A_{bb}^{(s)} \end{bmatrix} \begin{bmatrix} x_i^{(s)} \\ x_b^{(s)} \end{bmatrix} = \begin{bmatrix} f_i^{(s)} \\ f_b^{(s)} + \lambda_b^{(s)} \end{bmatrix}, \qquad s = 1, 2$$

the augmentation matrix $A_{bb}^{(s)}$ will be considered as a sparse approximation of the neighbor Schur complement, as seen in subsection 1.2.3.

To be the restrictions of the global problem, each local solution must satisfy two conditions, first the continuity condition

$$x_b^{(1)} - x_b^{(2)} = 0$$

the second condition is the interface equilibrium, which is the last line in the global block formulation

$$K_{bi}^{(1)} x_i^{(1)} + K_{bi}^{(2)} x_i^{(2)} + (K_{bb}^{(1)} + K_{bb}^{(2)}) x_b = f_b^{(1)} + f_b^{(2)}$$

if we use the continuity condition, we have

$$K_{bi}^{(1)}x_i^{(1)} + K_{bi}^{(2)}x_i^{(2)} + K_{bb}^{(1)}x_b^{(1)} + K_{bb}^{(2)}x_b^{(2)} = f_b^{(1)} + f_b^{(2)}$$

now using the last line of the local problems, we can write the equilibrium condition as

$$A_{bb}^{(1)}x_b^{(1)} + A_{bb}^{(2)}x_b^{(2)} = \lambda_b^{(1)} + \lambda_b^{(2)}$$

we combine both continuity and equilibrium conditions to obtain the interface mixed equations

$$\begin{aligned} A_{bb}^{(1)}x_b^{(2)} + A_{bb}^{(2)}x_b^{(2)} &= \lambda_b^{(1)} + \lambda_b^{(2)} \\ A_{bb}^{(1)}x_b^{(1)} + A_{bb}^{(2)}x_b^{(1)} &= \lambda_b^{(1)} + \lambda_b^{(2)} \end{aligned} \tag{4.1}$$

additionally, by eliminating the inner unknowns on the Robin local problem, we can obtain the explicit relation between the trace of the solution on the interface $x_b^{(s)}$ and the discrete augmented flux $\lambda_b^{(s)}$

$$(K_{bb}^{(s)} - K_{bi}^{(s)}K_{ii}^{(s)^{-1}}K_{ib}^{(s)} + A_{bb}^{(s)})x_b^{(s)} = \lambda_b^{(s)} + f_b^{(s)} - K_{bi}^{(s)}K_{ii}^{(s)^{-1}}f_i^{(s)}$$

we denote by $S_{bb}^{(s)} := K_{bb}^{(s)} - K_{bi}^{(s)}K_{ii}^{(s)^{-1}}K_{ib}^{(s)}$ the Schur complement matrix and by $c_b^{(s)} = f_b^{(s)} - K_{bi}^{(s)}K_{ii}^{(s)^{-1}}f_i^{(s)}$ the condensed right-hand side.

Replacing $x_b^{(s)}$ with their values as functions of $\lambda_b^{(s)}$ coming from previous explicit relation, and using the mixed equations (4.1), we can write the condensed interface problem of the form $F\lambda = d$ associated to the FETI-2LM method

$$\begin{bmatrix} I & I - (A_{bb}^{(1)} + A_{bb}^{(2)})(S_{bb}^{(2)} + A_{bb}^{(2)})^{-1} \\ I - (A_{bb}^{(1)} + A_{bb}^{(2)})(S_{bb}^{(1)} + A_{bb}^{(1)})^{-1} & I \end{bmatrix} \begin{bmatrix} \lambda_b^{(1)} \\ \lambda_b^{(2)} \end{bmatrix} = \begin{bmatrix} (A_{bb}^{(1)} + A_{bb}^{(2)})(S_{bb}^{(2)} + A_{bb}^{(2)})^{-1}c_b^{(2)} \\ (A_{bb}^{(1)} + A_{bb}^{(2)})(S_{bb}^{(1)} + A_{bb}^{(1)})^{-1}c_b^{(1)} \end{bmatrix}$$

The definition and implementation of this method for general problems, come from the computation of this operator between every two neighbor subdomains in a general configuration.

The previous condensed non-symmetric problem is solved using the ORTHODIR iterative method. This is because in the general case, the operator is dense, so its assembling is computationally inefficient. On the contrary, to multiply some vector by this operator (denoted F as usual FETI methods), we only need local data and exchanges between neighbors; hence an iterative method such as ORTHODIR is more suited.

We present in Algorithm 18 the ORTHODIR method used in FETI-2LM, as it will be the base to construct its generalized block version.

Algorithm 18 FETI-2LM algorithm

Initialization
$\lambda_0 = 0$
$g_0 = (F\lambda_0 - d)$
$w_0 = g_0$
loop ORTHODIR Iteration from $p = 0, 1, \ldots$ until convergence
    $\rho_p = -\dfrac{(Fw_p)^T g_p}{(Fw_p)^T (Fw_p)}$
    $\lambda_{p+1} = \lambda_p + \rho_p w_p$
    $g_{p+1} = g_p + \rho_p F w_p$
    loop Construction of the $p+1$ vector of the base $F^T F$-orthonormal
        $w_{p+1} = g_{p+1}$ ▷ $w_{p+1} = Fw_p$ is replaced by the gradient
        for $i = 0$ to p **do**
            $\gamma_i = -\dfrac{(Fw_i)^T (Fw_{p+1})}{(Fw_i)^T (Fw_i)}$
            $w_{p+1} = w_{p+1} + \gamma_i w_i$
            $Fw_{p+1} = Fw_{p+1} + \gamma_i Fw_i$
        end for
    end loop
end loop

In the next section, we will extend this algorithm to its block version. The idea is to use the decomposition of the gradient to build a block of search directions to update the solution in every iteration.

4.2 The Block-2LM Algorithm

Following the work made in the development of the S-FETI method, we can apply the same strategy to the FETI-2LM method. In this case, the method called Block-2LM will build a search space based on the separation of the gradient. The difference lies that the gradient will no longer be preconditioned as in S-FETI. However, it can also be considered as a sum of local gradients that when separated will improve the capture of the local subdomains behaviour. In any case, in FETI-2LM we have two independent Lagrange multipliers instead of one in each the interface. Therefore, we can create as many directions as the S-FETI method in both of its version (subdomain or interface divisions).

Following the notation in chapter 2, the gradient $g \in \mathbb{R}^n$ of the FETI-2LM method is

$$g = \begin{bmatrix} g^{(1)} \\ \vdots \\ g^{(N_s)} \end{bmatrix}$$

with N_s being the number of subdomains. Also, $g^{(i)} \in \mathbb{R}^{m_i}$, $m_i \leq n$ are the values of the global gradient in the subdomain $\Omega^{(s)}$. These local gradients are independent of each other because there are two Lagrange multipliers on the interface. This subdivision is enough to form a new block of directions, all by considering that g can be written as

$$g = \sum_{s=1}^{N_s} \bar{g}^{(s)}$$

where $\bar{g}^{(s)} \in \mathbb{R}^n$ is the extension by zero of each local vector, i.e.

$$\bar{g}^{(s)} = \begin{bmatrix} 0 & \dots & 0 & g^{(s)} & 0 & \dots & 0 \end{bmatrix}^T$$

with this division, our block of search directions will be defined by

$$Z = \left[\bar{g}^{(1)}, \bar{g}^{(2)} \dots, \bar{g}^{(N_s)} \right]$$

which is similar to the subdomain division in S-FETI.

Analogously, to form the interface subdivisions, we can consider at the local level that $g^{(s)}$ can also be decomposed into smaller vectors, one for each of the subdomain neighbors

$$g^{(s)} = \begin{bmatrix} g_{loc}^{(s),1} \\ \vdots \\ g_{loc}^{(s),n^{(s)}} \end{bmatrix} \tag{4.2}$$

with $n^{(s)}$ the number of neighbors of subdomain $\Omega^{(s)}$. Hence, the general gradient can also be written as

$$g = \sum_{s=1}^{N_s} \sum_{i=1}^{n^{(s)}} g^{(s),i} \tag{4.3}$$

where $g^{(s),i} \in \mathbb{R}^n$ is an extension by zero, defined as

$$g^{(s),i} := \begin{bmatrix} 0 & \dots & 0 & g_{loc}^{(s),i} & 0 & \dots & 0 \end{bmatrix}^T, \quad s = 1,\dots,N_s, \quad i = 1,\dots,n^{(s)}$$

From Equation (4.3), we can use the additive form of the gradient to define the new search space for the Block-2LM method. This space will be spanned at each iteration by the column vectors in

$$Z = \left[g^{(1),1},\dots,g^{(1),n^{(1)}}, \quad \dots \quad ,g^{(N_s),1},\dots,g^{(N_s),n^{(N_s)}} \right]$$

The iterative algorithm used for the FETI-2LM method is the ORTHODIR method, so the Krylov space built is formed from the successive matrix-gradient product, orthonormalized using the F^TF-norm. We are now spanning a different search space, but the use of this bigger space still implies the convergence of the method since it contains the original Krylov space. This result comes from the definition of Z and the fact that

$$Span\{g^{(1)} + \dots + g^{(m)}\} \subseteq Span\{g^{(1)},\dots,g^{(m)}\}$$

for any vectors $g^{(i)} \in \mathbb{R}^m$, $m \in \mathbb{N}$, $i = 1,\dots,m$.

The norm used in this method changes from F to F^TF-norm. The matrix that needs to be inverted also changes. This matrix is used to obtain the block of

orthonormal vectors and also to obtain the optimal descent coefficients. In this case, this matrix correspond to $(FW)^T FW$, where W is the block after applying the full reorthogonalization, i.e., at the p-iteration we have

$$W_p = Z_p + \sum_{i=0}^{p} W_i \Phi_i$$

The same linear dependency issue occurs for this matrix as in the S-FETI, see Section 2.2. Let us recall that this is due to the phenomenon of working near the computer precision or because we achieve convergence in some of the interfaces before the other. This dependency between columns in Z lead to a positive semi-definite $(FW)^T(FW)$ matrix, so as previous treatment, the Cholesky decomposition with partial pivoting will be used to select the independent directions (The Eigendecomposition is also valid). The algorithm is described in (19).

4.3 Implementation and cost of the method

The difference in cost between the FETI-2LM and Block-2LM is similar to the one from FETI and S-FETI. We can name the augmentation on the size of the information shared between neighbors, with no incremental number of exchanges. We also have the pseudo-inversion of the matrix $(FW)^T(FW)$, both of these changes are relatively cheap ones.

The most expensive part is the multiplication by the FETI operator. This can be done with a simple, straightforward implementation that solves a local problem for each column in Z. However, this can only serve for academical purposes to know the precision and convergence behaviour of it. If we want a practical method, we need to exploit the sparse structure of the search directions matrix Z.

We know from the definition of the columns $g^{(s),i}$ in Z that they are non-null only on the interface of subdomain $\Omega^{(s)}$. The multiplication by the FETI operator only requires, at the subdomain level, the solution of local problems equal to the number of neighbors, which can be done using a block implementation. With this consideration, we can have the speedup to make this method useful in more

Algorithm 19 Block FETI-2LM method

Initialization
$\lambda_0 = 0$
$g_0 = F\lambda_0 - b$
$Z_0 = [\dots, g_0^{(s),i}, \dots], \quad s = 1, N_s,\ i = 1, n^{(s)}$
$W_0 = Z_0$
loop Block ORTHODIR Iteration from $p = 0, 1\dots$ convergence
  $N_p L_p L_p^T N_p^T = (FW_p)^T (FW_p)$ ▷ Cholesky factorization
  $W_p = W_p N_p L_p^{-T}$ ▷ Eliminates useless directions and
  $FW_p = FW_p N_p L_p^{-T}$ ▷ $F^T F$-orthogonalizes blocks
  $\rho_p = -(FW_p)^T g_p$
  $\lambda_{p+1} = \lambda_p + W_p \rho_p$
  $g_{p+1} = g_p + FW_p \rho_p$
  loop Construction of the $p+1$ vector of the base $F^T F$-orthonormal
    $Z_{p+1} = [\dots, g_{p+1}^{(s),i}, \dots], \quad s = 1, N_s,\ i = 1, n^{(s)}$
    for $i = 0$ to p **do**
      $\Phi_i = -(FW_i)^T (FZ_{p+1})$
      $W_{p+1} = Z_{p+1} + W_i \Phi_i$
      $FW_{p+1} = FZ_{p+1} + FW_i \Phi_i$
    end for
  end loop
end loop

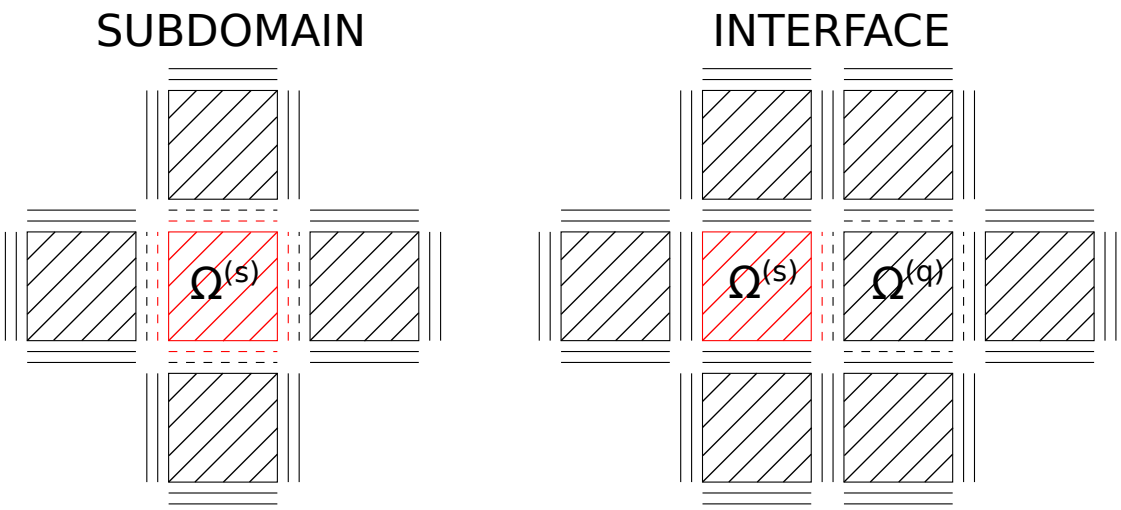

Figure 4.2 –
Left: In red, the coarse modes describing Z owned by subdomain $\Omega^{(s)}$. In dotted lines, the modes shared between $\Omega^{(s)}$ and the subdomains involved in multiplying by the FETI-2LM operator.
Right: In dotted lines, the coarse modes describing FZ owned by the local interface of $\Omega^{(s)}$. In red, the non-null modes in the interface.

real problems, because the time spent during the resolutions will be recovered by reducing the iterations needed for convergence.

To understand the implementation of this method, we will again use the so-called *Coarse Modes*, which correspond to the non-null vectors in each column of Z or FZ. These modes are stored locally in each subdomain. In this case, we have one mode for each neighbor; they are the vectors $g_{loc}^{(s),i}$ defined in Equation (4.2). At the same time, they are shared with every neighbor for further computations, e.g., multiplication by the FETI operator.

In Figure 4.2 we can see in the left, the described modes owned by the subdomains and in dotted lines are the ones owned by a subdomain; also, we see the subdomains involved in the computing of FZ. In the right, the dotted modes are the ones held by the red interface, at the same time these modes are non-null modes of FZ in the red interface, this comes from the definition of the operator.

To better understand this, let us look at the definition of the operator F for a single interface

$$F\begin{bmatrix}\lambda_1^{(s)}\\ \lambda_1^{(q)}\end{bmatrix}=\begin{bmatrix}\lambda_1^{(s)}+\lambda_1^{(q)}-(A^{(s)}+A^{(q)})(S^{(q)}+A^{(q)})^{-1}\lambda_1^{(q)}\\ \lambda_1^{(s)}+\lambda_1^{(q)}-(A^{(s)}+A^{(q)})(S^{(s)}+A^{(s)})^{-1}\lambda_1^{(s)}\end{bmatrix}$$

this implies that for the case of the Block-2LM, at the interface level, we have to compute

$$F\begin{bmatrix}\lambda_1^{(s)}\\ 0\end{bmatrix}=\begin{bmatrix}\lambda_1^{(s)}\\ \lambda_1^{(s)}-(A^{(s)}+A^{(q)})(S^{(s)}+A^{(s)})^{-1}\lambda_1^{(s)}\end{bmatrix}$$
$$F\begin{bmatrix}0\\ \lambda_1^{(q)}\end{bmatrix}=\begin{bmatrix}\lambda_1^{(q)}-(A^{(s)}+A^{(q)})(S^{(q)}+A^{(q)})^{-1}\lambda_1^{(q)}\\ \lambda_1^{(q)}\end{bmatrix}$$

We note that each subdomain solves one local problem for each local interface (number of neighbors). All the local solutions, plus the multiplication by local augmentation matrix are sent together to each of the neighbors; meaning we send the vectors

$$(S^{(s)}+A^{(s)})^{-1}\lambda_1^{(s)}$$
$$A^{(s)}(S^{(s)}+A^{(s)})^{-1}\lambda_1^{(s)}$$

this for every local computation performed. Thus, we can multiply previous vectors by the local augmentation matrix associated with the interface. After these multiplications, we can compute and store the local contribution to FZ. With this, each local interface stores the contributions to FZ done by the neighbor subdomain.

Finally, in terms of memory use, the same issue described in chapter 2 is expected with this implementation. Nevertheless, we can use the same strategy to store the sparse directions and reconstruct them later, although we will leave it as a future work.

Remark: We want to point out some of the differences between this method and S-FETI. The fact that no projection is needed in this case, so no extra cost is added in this part, and also the connectivity is more limited than in the S-FETI method, thanks to the definition of the FETI-2LM operator. Hence, we need

fewer modes stored in each interface.

4.4 Numerical results

In this section, we will be performing another basic comparative testing, that will include the FETI-2LM and Block-2LM methods. Classical comparison between iteration number before convergence, as well as a full `Fortran-MPI` implementation to measure for time performance, will be analyzed.

To perform this benchmark, we do not need for a special complex case, as one of the main properties of both methods is the robustness and is expected to be similar. We will leave the tests for complex cases as future work. From now on, we will focus on the study of the basic Poisson problem in 3D, with tri-linear $\mathbb{Q}^1$ functions for the finite element discretization. For more details, see the numerical results in chapter 3 and chapter 2. We just recall the existence of the ν parameter, that also, in this case, will change between neighbor subdomains, meaning that a heterogeneous checkerboard type of configuration will be tested, and blocks of two different materials will be used.

4.4.1 Block-2LM vs 2LM

The global domain is again the cube $\Omega = [0,1]^3$. Several subdomain divisions will be done, all characterized by being smaller cubes of equal size. We want to see the impact of augmenting the generated search directions in each iteration.

As usual, the global error and solution jump across the interfaces will be our stopping criteria. We stop when both of them are less to 10^{-4}. We have diminished this value to be able to use the Block version in cases where the memory is an important constraint.

In Table 4.1 and Table 4.2 we can see a general comparative of both methods. We have set the values for the parameters $patch\_size = 2$ and $patch\_depth = 3$, see Algorithm 7. Also, the number of elements per subdomain changes, as well as the total number of subdomains (cubes). The values of the parameter ν are 10^0 and 10^5 in a checkerboard configuration Figure 2.9.

Method	$\frac{Elements}{subdomain}$	Iterations	Time (s)
FETI-2LM	46656	87	8.1
BLOCK-2LM	46656	50	2115.1
FETI-2LM	74088	91	14.5
BLOCK-2LM	74088	57	3516.4
FETI-2LM	103823	93	21.7
BLOCK-2LM	103823	62	7919.6

Table 4.1 – 64 subdomains

Method	$\frac{Elements}{subdomain}$	Iterations	Time (s)
FETI-2LM	1000	39	0.1
BLOCK-2LM	1000	24	375.7
FETI-2LM	27000	45	3.3
BLOCK-2LM	27000	41	7377.0

Table 4.2 – 125 subdomains

In the results, we can see the reduction in the number of iteration when comparing the two methods. However, its values are not small enough to compensate for the time spent in each iteration. This slowdown in the total time is mainly due to the full reconjugation, where after a certain number of total iterations the number of matrix-matrix products needed is too costly.

These effects are less present in the S-FETI methods. The computations in this part can be reduced by 2 because the values of the Lagrange multiplier are shared by neighbor subdomains. In Block-2LM we are forced to work with full-size blocks of search directions since two independent Lagrange multipliers are used in this case. Also, the number of iterations goes up as high as 62 and the quadratic effects of full reorthogonalization produce the notorious slowdown.

We consider this implementation as a first step before doing the sparse storage implementation in the same way that was done in chapter 2. In that case, the performance of this method, specially when doing the full reconjugation, should be several times faster, and hence a method to be used in practical applications.

Remark: As part of the future work, we leave the test of a variation of this method, with even more directions per iterations. The idea is to use the neighbor local gradient as a new direction doubling the number of directions built. The normal block of directions, using the notation of this chapter, is defined as

$$Z = \left[\ldots, g^{(s),1}, \ldots, g^{(s),n^{(s)}}, \ldots\right], \qquad s = 1, \ldots, N_s$$

and when doubling the directions we will have

$$Z = \left[\ldots, g^{(s),1}, g^{vec\{(s),1\}}, \ldots, g^{(s),n^{(s)}}, g^{vec\{(s),n^{(s)}\}}, \ldots\right], \qquad s = 1, \ldots, N_s$$

where $g^{vec\{(s),i\}}$ is the extension by zero (using the same position as the non zero values of $g^{(s),i}$) of the local gradient coming from the neighbor subdomain of $\Omega^{(s)}$ through his local interface i.

4.5 Conclusion

Following the ideas used in S-FETI, a new Block method has been developed for FETI-2LM. The properties of FETI-2LM, such as the lack of an optimal preconditioner or the fact that two Lagrange multipliers are already imposed in the interfaces, make this method a less appropriate candidate for a block version than the FETI-1LM was. However, the enlargement the search space, still predicts a better convergence in terms of iterations for this method.

The numerical results validate the reduction in the iterations needed by this method to converge. Nevertheless, this reduction is not compensated as in the S-FETI case, mainly because the total number of iterations achieved is not sufficiently small to keep the time spent in the full reconjugation as a factor controlled. This issue produces a major slowdown in the Block-2LM method.

Alternatives to the reconjugation must be found. The sparse storage proposed in chapter 2 appears as a good option, but we leave that as a future work.

Conclusion and perspectives

Throughout the work exposed in the four chapters of this thesis, we have detailed several improvements to existing classic FETI methods, that are currently used in real life applications, such as structure, electromagnetism models among others.

The first part consisted in the development of Hybrid-FETI, born from the mixing of the FETI-1LM and FETI-2LM methods. This new FETI algorithm consist in the imposition of either Neumann or Robin conditions in the local interfaces, as done in FETI-1LM and FETI-2LM respectively, as a way of "gluing" them.

Due to the particularities of both base methods, the Hybrid-FETI is well suited for solving problems where a small number of interfaces present bad numerical features beyond the heterogeneities. This is the case of contact problems, where the contact area presents an ill-conditioning in which the FETI-1LM even with the consistent version of the Dirichlet preconditioner does not assure the convergence. These problems are usually solved using the FETI-2LM due to the formulation with two independent Lagrange multipliers provided by this method. FETI-2LM can handle these issues in the contact area in a more robust way. However, its use is only required for the contact area, so we can improve the global convergence by treating the non-contact interfaces with the regular and faster FETI-1LM method. This global liberty of choice for the interfaces is the new Hybrid-FETI method.

When imposing one or the other type of interfaces, we form subdomains where both methods are active, forcing us to create a preconditioner optimal for these subdomains. In the subdomains where only the Neumann boundary

condition is imposed, we use the regular Dirichlet preconditioner to boost the global convergence. In this context, as an extension of the regular Dirichlet preconditioner, the Robin preconditioner was developed. This preconditioner differs from the Dirichlet only in the subdomains with mixed interfaces. In these subdomains the augmentation matrix present in the local problem is added to the Schur complement usually computed in the Dirichlet preconditioner.

Although we see an improvement with the Robin preconditioner, we leave as future work for this part the development of an additional preconditioner for the 2LM interfaces, where no preconditioner is applied or in general a new preconditioner for the global operator of the Hybrid-FETI.

Looking at numerical results, the best ones obtained by this new method are seen in the contact case where we reduced by almost two times the number of iterations of the only working base method, which is the FETI-2LM. These reductions should be directly applied to a reduction in terms of total time, but since we did not have the chance to test bigger cases, this will rest as a developing work. In this same line, we want to extend the application of the Hybrid-FETI to other problems solved only by the FETI-2LM method, for example, the ones arising from the use of mortar elements to treat non-conforming meshes, where also an ill-conditioning is present in a localized zone, see [65].

In the second part, we present the development of the S-FETI starting from its first version presented in [60] to its more precise general formulation, shown in [38]. The development of the consistent Dirichlet preconditioner is also exposed as this is a crucial part of the good behaviour of S-FETI. Details of the implementation were also explained, also needed to understand the new sparse storage developed in this chapter.

Following the presentation of the method, we introduced a variation to the original method. This change comes from the problem of treating the linear dependency usually present when constructing a block of search directions. This dependency comes from different elements, which are present in almost all problems. A good recognition of them can have a large impact on the final results. The Eigenvalue decomposition is proposed as an alternative to the basic Cholesky factorization that uses the basic S-FETI.

Confirmed by the numerical results we can see that with the Eigendecomposition the same convergence can be achieved, but using fewer directions in each iteration. This reduction comes from the fact that the new decomposition can arrange the orthogonal search directions in order of relevance an reduce the complete information contained in a block of directions. On the contrary, the Cholesky decomposition only eliminates the ones considered problematic, a process that leads to possible loss of information at each iteration.

One of the goals of the Eigenvalue decomposition is trying to reduce the number of directions used in each iteration, and therefore make the usage of memory S-FETI method less expensive. Even though there is an improvement present in here, the memory cost of this method makes it limited if we want to solve bigger cases. To surpass this, a new sparse storage was developed. This implementation is based on the sparsity present in the block of search directions, where at a local level each block of directions contains non-null columns in the ones representing the same subdomain and its neighbors. Using this information we can reconstruct the directions from a series of smaller coefficient matrices and these non-null vectors from the blocks, stored as *Coarse Modes*. The reduction in memory use is more clear when working on problems with a great number of unknowns.

In the numerical results, we show that the impact in total time induced by the extra computations is negligible if we do a correct implementation. Several improvements to the basic implementation were introduced, leading to a new S-FETI with sparse storage as fast as the original one. However, there is still room for new improvements as we have augmented the complexity of the algorithm by introducing several new sources of computations, that can also be revised. We leave that for future work.

The third chapter is dedicated to testing different ideas regarding variations of the S-FETI. We can name the two main ones, which consist of using the block of search directions build by S-FETI as a preconditioner for the original FETI method. The second idea is to add to the search direction created in the first method a small part of the vectors associated with the largest eigenvalues when doing the decomposition of $W^T F W$.

Both methods show an improvement in terms of iterations needed for convergence, but due to the expensive cost of the first one of them, it can only be considered for academical purposes and future development of methods derived from S-FETI. The second one needs further tests to asses its value.

Finally, the Block-2LM method was developed as an extension of FETI-2LM. We use the same ideas that lead to the S-FETI method, i.e., the construction of a larger search space (in this case, containing the original Krylov space to assure the convergence). The sparsity of the search directions helps to speed up this method, reducing drastically the number of forward-backward reductions done in every iteration.

The results confirm a reduction in the number of iterations, although not as significant as the one seen in S-FETI, and with no reduction in the total time spent. However, this method is considered as a first step, because a sparse storage version may also be needed in this case. We justify this necessity in the existence of problems in electromagnetism, like the ones seen in [8],[64].

Bibliography

[1] J. Alberty, C. Carstensen, S. A. Funken, and R. Klose. "Matlab Implementation of the Finite Element Method in Elasticity". In: *Computing* 69.3 (2002), pp. 239–263.

[2] Jochen Alberty, Carsten Carstensen, and Stefan A. Funken. "Remarks around 50 lines of Matlab: short finite element implementation". In: *Numerical Algorithms* 20.2 (1999), pp. 117–137.

[3] P.R. Amestoy, I.S. Duff, and J.-Y. L'Excellent. "Multifrontal parallel distributed symmetric and unsymmetric solvers". In: *Computer Methods in Applied Mechanics and Engineering* 184.2 (2000), pp. 501–520.

[4] Philip Avery, Gert Rebel, Michel Lesoinne, and Charbel Farhat. "A numerically scalable dual-primal substructuring method for the solution of contact problems—-part I: the frictionless case". In: *Computer Methods in Applied Mechanics and Engineering* 193.23–26 (2004), pp. 2403–2426.

[5] O. Axelsson and P. S. Vassilevski. "A Black Box Generalized Conjugate Gradient Solver with Inner Iterations and Variable-Step Preconditioning". In: *SIAM Journal on Matrix Analysis and Applications* 12.4 (1991), pp. 625–644.

[6] O. Axelsson and P. S. Vassilevski. "Variable-step multilevel preconditioning methods, I: Self-adjoint and positive definite elliptic problems". In: *Numerical Linear Algebra with Applications* 1.1 (1994), pp. 75–101.

[7] Owe Axelsson. "Generalized Conjugate Gradient Methods". In: *Iterative Solution Methods*. Cambridge University Press, 1994, pp. 504–557.

[8] A. Barka and F. X. Roux. "Scalability of FETI-2LM methods on HPC clusters for antenna RCS applications". In: *2014 International Symposium on Antennas and Propagation Conference Proceedings*. Dec. 2014, pp. 19–20.

[9] C. Bernardi, Y. Maday, and A. T. Patera. "Domain Decomposition by the Mortar Element Method". In: *Asymptotic and Numerical Methods for Partial Differential Equations with Critical Parameters*. Ed. by Hans G. Kaper, Marc Garbey, and Gail W. Pieper. Dordrecht: Springer Netherlands, 1993, pp. 269–286.

[10] Ligia Cristina Brezeanu. "Contact Stresses: Analysis by Finite Element Method (FEM)". In: *Procedia Technology* 12 (2014). The 7th International Conference Interdisciplinarity in Engineering, INTER-ENG 2013, 10-11 October 2013, Petru Maior University of Tirgu Mures, Romania, pp. 401–410.

[11] Robert Bridson and Chen Greif. "A Multipreconditioned Conjugate Gradient Algorithm". In: *SIAM Journal on Matrix Analysis and Applications* 27.4 (2006), pp. 1056–1068.

[12] Ernie Chan, Marcel Heimlich, Avi Purkayastha, and Robert van de Geijn. "Collective communication: theory, practice, and experience". In: *Concurrency and Computation: Practice and Experience* 19.13 (2007), pp. 1749–1783.

[13] Ernie Chan, Marcel Heimlich, Avi Purkayastha, and Robert van de Geijn. "Collective Communication: Theory, Practice, and Experience: Research Articles". In: *Concurr. Comput. : Pract. Exper.* 19.13 (Sept. 2007), pp. 1749–1783.

[14] Andrew Chapman and Yousef Saad. "Deflated and augmented Krylov subspace techniques". In: *Numerical Linear Algebra with Applications* 4 (1996), pp. 43–66.

[15] Edmond Chow and Yousef Saad. "Approximate Inverse Techniques for Block-Partitioned Matrices". In: *SIAM Journal on Scientific Computing* 18.6 (1997), pp. 1657–1675.

[16] T. Coleman and C. Van Loan. *Handbook for Matrix Computations*. Society for Industrial and Applied Mathematics, 1988.

[17] Ibrahima Dione and José.M. Urquiza. "Finite element approximations of the Lamé system with penalized ideal contact boundary conditions". In: *Applied Mathematics and Computation* 223 (2013), pp. 115–126.

[18] V. Dolean, P. Jolivet, and F. Nataf. *An Introduction to Domain Decomposition Methods: Algorithms, Theory, and Parallel Implementation:* Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics, 2015.

[19] Jack Dongarra, Jeremy Du Croz, Iain Duff, and Sven Hammarling. "A Proposal for a Set of Level 3 Basic Linear Algebra Subprograms". In: *SIGNUM Newsl.* 22.3 (July 1987), pp. 2–14.

[20] Z. Dostál, T. Kozubek, V. Vondrák, T. Brzobohatý, and A. Markopoulos. "Scalable TFETI algorithm for the solution of multibody contact problems of elasticity". In: *International Journal for Numerical Methods in Engineering* 82.11 (2010), pp. 1384–1405.

[21] Z. Dostál, Francisco A.M. Gomes Neto, and Sandra A. Santos. "Solution of contact problems by FETI domain decomposition with natural coarse space projections". In: *Computer Methods in Applied Mechanics and Engineering* 190.13 (2000), pp. 1611–1627.

[22] Zdeněk Dostál, David Horák, Radek Kučera, Vít Vondrák, Jaroslav Haslinger, Jiří Dobiáš, and Svatopluk Pták. "FETI based algorithms for contact problems: scalability, large displacements and 3D Coulomb friction". In: *Computer Methods in Applied Mechanics and Engineering* 194.2 (2005). Selected papers from the 11th Conference on The Mathematics of Finite Elements and Applications, pp. 395–409.

[23] Zdeněk Dostál, Vít Vondrák, David Horák, Charbel Farhat, and Philip Avery. "Scalable FETI Algorithms for Frictionless Contact Problems". In: *Domain Decomposition Methods in Science and Engineering XVII*. Ed. by Ulrich Langer, Marco Discacciati, David E. Keyes, Olof B. Widlund, and Walter Zulehner. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 263–270.

[24] Iain S. Duff. "Parallel implementation of multifrontal schemes". In: *Parallel Computing* 3.3 (1986), pp. 193–204.

[25] D. Dureisseix and C. Farhat. "A numerically scalable domain decomposition method for the solution of frictionless contact problems". In: *International Journal for Numerical Methods in Engineering* 50.12 (2001), pp. 2643–2666.

[26] C. Farhat, F.X. Roux, International Association for Computational Mechanics, and J.T. Oden. *Implicit Parallel Processing in Structural Mechanics*. Computational mechanics advances. North-Holland, 1994.

[27] Charbel Farhat, Po-Shu Chen, Jan Mandel, and Francois Xavier Roux. "The two-level FETI method Part II: Extension to shell problems, parallel implementation and performance results". In: *Computer Methods in Applied Mechanics and Engineering* 155.1 (1998), pp. 153–179.

[28] Charbel Farhat, Po-Shu Chen, Franck Risler, and Francois-Xavier Roux. "A unified framework for accelerating the convergence of iterative substructuring methods with Lagrange multipliers". In: *International Journal for Numerical Methods in Engineering* 42.2 (1998), pp. 257–288.

[29] Charbel Farhat, Michael Lesoinne, and Kendall Pierson. "A scalable dual-primal domain decomposition method". In: *Numerical Linear Algebra with Applications* 7.7-8 (2000), pp. 687–714.

[30] 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". In: *International Journal for Numerical Methods in Engineering* 50.7 (2001), pp. 1523–1544.

[31] Charbel Farhat, Antonini Macedo, Michel Lesoinne, Francois-Xavier Roux, Frédéric Magoulès, and Armel de La Bourdonnaie. "Two-level domain decomposition methods with Lagrange multipliers for the fast iterative solution of acoustic scattering problems". In: *Computer Methods in Applied Mechanics and Engineering* 184.2–4 (2000), pp. 213–239.

[32] Charbel Farhat and Jan Mandel. "The two-level FETI method for static and dynamic plate problems Part I: An optimal iterative solver for biharmonic systems". In: *Computer Methods in Applied Mechanics and Engineering* 155.1 (1998), pp. 129–151.

[33] Charbel Farhat, Jan Mandel, and Francois Xavier Roux. "Optimal convergence properties of the FETI domain decomposition method". In: *Computer Methods in Applied Mechanics and Engineering* 115.3 (1994), pp. 365–385.

[34] Charbel Farhat and Francois-Xavier Roux. "A method of finite element tearing and interconnecting and its parallel solution algorithm". In: *International Journal for Numerical Methods in Engineering* 32.6 (1991), pp. 1205–1227.

[35] Charbel Farhat and Francois-Xavier Roux. "An Unconventional Domain Decomposition Method for an Efficient Parallel Solution of Large-Scale Finite Element Systems". In: *SIAM Journal on Scientific and Statistical Computing* 13.1 (1992), pp. 379–396.

[36] P. Gosselet, D. J. Rixen, and C. Rey. "A domain decomposition strategy to efficiently solve structures containing repeated patterns". In: *International Journal for Numerical Methods in Engineering* 78.7 (2009), pp. 828–842.

[37] Pierre Gosselet, Christian Rey, and Daniel J Rixen. "On the initial estimate of interface forces in FETI methods". In: *Computer Methods in Applied Mechanics and Engineering* 192.25 (2003), pp. 2749–2764.

[38] Pierre Gosselet, Daniel Rixen, François-Xavier Roux, and Nicole Spillane. "Simultaneous FETI and block FETI: Robust domain decomposition with multiple search directions". In: *International Journal for Numerical Methods in Engineering* 104.10 (2015), pp. 905–927.

[39] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. "A high-performance, portable implementation of the MPI message passing interface standard". In: *Parallel Computing* 22.6 (1996), pp. 789–828.

[40] Tongxiang Gu, Xingping Liu, Zeyao Mo, and Xuebin Chi. "Multiple search direction conjugate gradient method I: methods and their propositions". In: *International Journal of Computer Mathematics* 81.9 (2004), pp. 1133–1143.

[41] Tongxiang Gu, Xingping Liu, Zeyao Mo, and Xuebin Chi. "Multiple search direction conjugate gradient method II: theory and numerical experiments". In: *International Journal of Computer Mathematics* 81.10 (2004), pp. 1289–1307.

[42] A. Gupta and V. Kumar. "Optimally scalable parallel sparse Cholesky factorization". In: Society for Industrial and Applied Mathematics, Philadelphia, PA (United States), Dec. 1995.

[43] Sven Hammarling, Nicholas J. Higham, and Craig Lucas. "LAPACK-Style Codes for Pivoted Cholesky and QR Updating". In: *Applied Parallel Computing. State of the Art in Scientific Computing: 8th International Workshop, PARA 2006, Umeå, Sweden, June 18-21, 2006, Revised Selected Papers*. Ed. by Bo Kågström, Erik Elmroth, Jack Dongarra, and Jerzy Waśniewski. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 137–146.

[44] M. R. Hestenes and E. Stiefel. "Methods of conjugate gradients for solving linear systems". In: *Journal of research of the National Bureau of Standards* 49 (1952), pp. 409–436.

[45] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Second. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2002.

[46] George Karypis and Vipin Kumar. "METIS, a Software Package for Partitioning Unstructured Graphs and Computing Fill-Reduced Orderings of Sparse Matrices". In: *University of Minnesota, Department of Computer Science* 180 (1998).

[47] N. Kikuchi and J. Oden. *Contact Problems in Elasticity*. Society for Industrial and Applied Mathematics, 1988.

[48] Armel de La Bourdonnaye, Charbel Farhat, Antonini Macedo, Frédéric Magoules, and François-Xavier Roux. "A Non-Overlapping Domain Decomposition Method for the Exterior Helmholtz Problem". In: *Contemporary Mathematics* 218 (1998), pp. 42–66.

[49] F. Magoules, F.X. Roux, and G. Houzeaux. *Parallel Scientific Computing*. ISTE. Wiley, 2015.

[50] Frédéric Magoulès, François-Xavier Roux, and Laurent Series. "Algebraic approximation of Dirichlet-to-Neumann maps for the equations of linear elasticity". In: *Computer Methods in Applied Mechanics and Engineering* 195.29 (2006). Absorbing Boundary Conditions, pp. 3742–3759.

[51] Jan Mandel. "Balancing domain decomposition". In: *Communications in Numerical Methods in Engineering* 9.3 (1993), pp. 233–241. ISSN: 1099-0887.

[52] Jan Mandel and Radek Tezaur. "Convergence of a substructuring method with Lagrange multipliers". In: *Numerische Mathematik* 73.4 (1996), pp. 473–487.

[53] Jan Mandel, Radek Tezaur, and Charbel Farhat. "A Scalable Substructuring Method by Lagrange Multipliers for Plate Bending Problems". In: *SIAM Journal on Numerical Analysis* 36.5 (1999), pp. 1370–1391.

[54] Yvan Notay. "Flexible Conjugate Gradients". In: *SIAM Journal on Scientific Computing* 22.4 (2000), pp. 1444–1460.

[55] Dianne P O'Leary. "Parallel implementation of the block conjugate gradient algorithm". In: *Parallel Computing* 5.1 (1987). Proceedings of the International Conference on Vector and Parallel Computing-Issues in Applied Research and Development, pp. 127–139.

[56] Cosmin G. Petra, Olaf Schenk, and Mihai Anitescu. "Real-time stochastic optimization of complex energy systems on high-performance computers". In: *IEEE Computing in Science & Engineering* 16.5 (2014), pp. 32–42.

[57] Cosmin G. Petra, Olaf Schenk, Miles Lubin, and Klaus Gärtner. "An augmented incomplete factorization approach for computing the Schur complement in stochastic optimization". In: *SIAM Journal on Scientific Computing* 36.2 (2014), pp. C139–C162.

[58] G. Prathap. "Brick Elements". In: *The Finite Element Method in Structural Mechanics: Principles and Practice of Design of Field-consistent Elements for Structural and Solid Mechanics*. Dordrecht: Springer Netherlands, 1993, pp. 287–336.

[59] A. Quarteroni and A. Valli. *Domain Decomposition Methods for Partial Differential Equations*. Numerical Mathematics and Scie. Clarendon Press, 1999.

[60] Daniel Rixen. "Substructuring and dual methods in structural analysis". PhD thesis. Université de Liège, Belgium, Collection des Publications de la Faculté des Sciences appliquées, n.175, 1997.

[61] Daniel J. Rixen and Charbel Farhat. "A simple and efficient extension of a class of substructure based preconditioners to heterogeneous structural mechanics problems". In: *International Journal for Numerical Methods in Engineering* 44.4 (1999), pp. 489–516.

[62] Daniel J. Rixen, Charbel Farhat, Radek Tezaur, and Jan Mandel. "Theoretical comparison of the FETI and algebraically partitioned FETI methods, and performance comparisons with a direct sparse solver". In: *International Journal for Numerical Methods in Engineering* 46.4 (1999), pp. 501–533.

[63] F. X. Roux. "Spectral analysis of the interface operators associated with the preconditioned saddle-point principle domain decomposition method". In: *Fifth International Symposium on Domain Decomposition Methods for Partial Differential Equations*. SIAM, 1992, pp. 73–90.

[64] F. X. Roux and A. Barka. "Block Krylov Recycling Algorithms for FETI-2LM Applied to 3-D Electromagnetic Wave Scattering and Radiation". In: *IEEE Transactions on Antennas and Propagation* 65.4 (Apr. 2017), pp. 1886–1895.

[65] François-Xavier Roux. "A FETI-2LM Method for Non-Matching Grids". In: *Domain Decomposition Methods in Science and Engineering XVIII*. Ed. by Michel Bercovier, Martin J. Gander, Ralf Kornhuber, and Olof Widlund. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 121–128.

[66] François-Xavier Roux. "Acceleration of the Outer Conjugate Gradient by Reorthogonalization for a Domain Decomposition Method for Structural Analysis Problems". In: *Proceedings of the 3rd International Conference on Supercomputing*. ICS '89. Crete, Greece: ACM, 1989, pp. 471–476.

[67] François-Xavier Roux, Frédéric Magoulès, Stéphanie Salmon, and Laurent Series. "29. Optimization of Interface Operator Based on Algebraic Approach". In: *Domain Decomposition Methods in Science and Engineering* (2002), p. 297.

[68] F.-X. Roux, F. Magoulès, L. Series, and Y. Boubendir. "Approximation of Optimal Interface Boundary Conditions for Two-Lagrange Multiplier FETI Method". In: *Domain Decomposition Methods in Science and Engineering*. Ed. by Timothy J. Barth et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 283–290.

[69] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Second. Society for Industrial and Applied Mathematics, 2003.

[70] A. H. Sameh and D. J. Kuck. "On Stable Parallel Linear System Solvers". In: *J. ACM* 25.1 (Jan. 1978), pp. 81–91.

[71] Olaf Schenk and Klaus Gärtner. "Solving unsymmetric sparse systems of linear equations with PARDISO". In: *Future Generation Computer Systems* 20.3 (2004), pp. 475–487.

[72] V. Simoncini and E. Gallopoulos. "Convergence properties of block GMRES and matrix polynomials". In: *Linear Algebra and its Applications* 247 (1996), pp. 97–119.

[73] Valeria Simoncini and Daniel B. Szyld. "Flexible Inner-Outer Krylov Subspace Methods". In: *SIAM Journal on Numerical Analysis* 40.6 (2002), pp. 2219–2239.

[74] A. van der Sluis and H. A. van der Vorst. "The rate of convergence of Conjugate Gradients". In: *Numerische Mathematik* 48.5 (1986), pp. 543–560.

[75] N. Spillane and D.J. Rixen. "Automatic spectral coarse spaces for robust finite element tearing and interconnecting and balanced domain decomposition algorithms". In: *International Journal for Numerical Methods in Engineering* 95.11 (2013), pp. 953–990.

[76] Nicole Spillane. "An Adaptive MultiPreconditioned Conjugate Gradient Algorithm". In: *SIAM Journal on Scientific Computing* 38.3 (2016), A1896–A1918.

[77] Dan Stefanica. "FETI and FETI-DP Methods for Spectral and Mortar Spectral Elements: A Performance Comparison". In: *Journal of Scientific Computing* 17.1 (2002), pp. 629–638.

[78] A. Suzuki and F.-X. Roux. "A dissection solver with kernel detection for symmetric finite element matrices on shared memory computers". In: *International Journal for Numerical Methods in Engineering* 100.2 (2014), pp. 136–164.

[79] A. Toselli and O. Widlund. *Domain Decomposition Methods - Algorithms and Theory*. Springer Series in Computational Mathematics. Springer Berlin Heidelberg, 2006.

[80] Peter Wriggers. "Discretization, Large Deformation Contact". In: *Computational Contact Mechanics*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 225–307.

[81] Alexander J. Zaslavski. "Penalty Methods". In: *Numerical Optimization with Computational Errors*. Cham: Springer International Publishing, 2016, pp. 239–264.

[82] Zhi-Hua Zhong and Jaroslav Mackerle. "Static Contact Problems - A review". In: *Engineering Computations* 9.1 (1992), pp. 3–37.

[83] Yunkai Zhou and Yousef Saad. "Block Krylov–Schur method for large symmetric eigenvalue problems". In: *Numerical Algorithms* 47.4 (2008), pp. 341–359.

Hybridization of FETI Methods

Abstract

In the domain decomposition framework, classic non-overlapping methods such as FETI and its variations are often used in industrial applications due to better performance and properties shown by them. Several improvements have been developed in order to achieve these good results. Following this line of work we present new methods based on FETI and new implementations of the recently developed S-FETI method.
The classics FETI-1LM and FETI-2LM methods are used to build a new hybrid FETI method as a fusion of both of them. These methods differ from one another in the boundary condition imposed on the interface. The basic idea is to develop a new algorithm that can use both methods at the same time by choosing in each interface the most suited condition depending on the characteristics of the problem. We search to have a faster and more robust code that can work with configurations that the base methods will not handle optimally by themselves. The performance of this method is tested on a problem with a contact surface, where most of the numerical issues are focused.
In the following part, we present a new implementation for the S-FETI method. As seen in the original presentation, the full reorthogonalization needed by S-FETI, and all FETI methods in general, impose the storage of all the directions created, making the memory cost somehow a relevant issue. The objective is to reduce the memory usage of this method, allowing this method to work on larger problems. We achieve this by using the sparsity properties of the search directions. We also propose other variations to lower the storage, but this time by reducing the directions saved at each iteration. Finally, an extension of the FETI-2LM method to his block version as in S-FETI is developed. Numerical results for the different algorithms are presented.

Keywords: numerical analysis, domain decomposition methods, algebra, scientific computation

---

LJLL Laboratoire Jacques-Louis Lions
Laboratoire Jacques-Louis Lions – 4 place Jussieu – Université Pierre et Marie Curie – Boîte courrier 187 – 75252 Paris Cedex 05 – France