Hybridization of FETI Methods

Roberto Molina-Sepulveda

UPMC SORBONNE UNIVERSITÉS

Laboratoire Jacques-Louis Lions

Doctoral School **ED Sciences Mathématiques Paris Centre**

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

Thesis defended by **Roberto Molina**

Defended on **1st September, 2017**

In order to become Doctor from Laboratoire Jacques-Louis Lions

Academic Field **Applied Mathematics**

Speciality **Numerical Analysis**

Thesis Title

Hybridization of FETI Methods

Thesis supervised by François-Xavier Roux

Committee members

Supervisor	François-Xavier Roux	Professor at LJLL

UPMC SORBONNE UNIVERSITÉS

Laboratoire Jacques-Louis Lions

Doctoral School **ED Sciences Mathématiques Paris Centre**

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

Thesis defended by **Roberto Molina**

Defended on **1st September, 2017**

In order to become Doctor from Laboratoire Jacques-Louis Lions

Academic Field **Applied Mathematics**

Speciality **Numerical Analysis**

Thesis Title

Hybridization of FETI Methods

Thesis supervised by François-Xavier Roux

Committee members

Supervisor	François-Xavier Roux	Professor at LJLL

UPMC SORBONNE UNIVERSITÉS

Laboratoire Jacques-Louis Lions

École doctorale **ED Sciences Mathématiques Paris Centre**

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

Thèse présentée par **Roberto Molina**

Soutenue le **1er septembre 2017**

En vue de l'obtention du grade de docteur de l'Laboratoire Jacques-Louis Lions

Discipline **Mathématiques Appliquées**

Spécialité **Analyse Numérique**

Titre de la thèse

Hybridation de mèthodes FETI

Thèse dirigée par François-Xavier Roux

Composition du jury

Directeur de thèse	François-Xavier Roux	professeur au LJLL

The Laboratoire Jacques-Louis Lions neither endorse nor censure authors' opinions expressed in the theses: these opinions must be considered to be those of their authors.

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 this work new domain decomposition methods and new implementations for existing methods are developed. A new method based on previous domain decomposition methods is formulated. The classic FETI [30] plus FETI-2LM [35] methods are used to build the new Hybrid-FETI. 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. By doing this we search to have a faster and more robust code that can work with configurations that the base methods will not handle it optimally by himself. The performance is tested on a contact problem.
The following part involves the development of a new implementation for the S-FETI method [39], the idea is to reduce the memory usage of this method, to make it able to work in larger problem. Different variation for this method are also proposed, all searching the reduction of directions stored each iteration of the iterative method. 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

Thanks to my family, friends and people from the University.

Contents

List of Tables

List of Figures

Introduction

In the last decades and thanks to the increasing computational power, faster 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 of strategies in parallelism that allows to profit of this new computers architectures. Different parallel iterative and direct methods for solving linear systems had been developed [70],[25],[3], both with positive and negative features in terms of speed, memory and accuracy. Results of iterative methods based on Krylov spaces usually depend on the condition number of the matrix representing the system, and the memory requirements are usually not an issue, on the other hand direct methods are more robust but the use of memory can be a problem for some large systems.

For problems coming from discretization of Finite Element Methods we have properties that allow us to use a different approach, that is the Domain Decomposition Methods (DDM), this methods can be considered as an hybrid between iterative and direct methods [60],[19]. They are based on the partition of the domain of the problem into subdomains, where smaller system of equations are defined. From this division, this methods can be categorized into two large groups, the overlapping and non overlapping methods.

In this work we will focus in some DDM with non overlapping interfaces, mainly the method of Finite Elements Tearing and Interconnecting or FETI and other related methods [30],[33],[35].

The objective of our work is to develop new FETI methods, that apply to particular cases, improving the results shown by the existing methods. Also we will expand the results on one of the existing FETI methods to extend his

applications to problems in which the current formulations does not allows it.

Contributions of this thesis

The following work is based in one of the most known of the non overlapping domain decomposition methods, the Finite Element Tearing and Interconnecting method (FETI) [30],[29].

The works on this method since its first formulation have produced several improvements to the same [34],[61], but also it has permitted the formulation of new FETI methods [33],[35]. Within this context, and using the similarities on the constructions of two of the most used FETI methods, namely the original FETI-1LM and the later developed FETI-2LM. We will formulate a new algorithm based in this two methods, that tries to take advantage of the good properties of one and the other. After the development of this new method, and after a better understanding of the same we can exhibit his advantages in contact cases where it outperforms the FETI-1LM and FETI-2LM methods.

In a different line of work, this time following the development of the new S-FETI method [39], we will continue the analysis within, in order to find new, faster or more robust forms of the same. Different variations of it will be formulated and tested, all of them trying to improve the existing results shown by the method. We will also extend the application of S-FETI to a larger class of problems, with a new implementation, base on a sparse storage, that reduces the memory limitations of this method.

Next we will try to use the ideas presented in the formulation of S-FETI to develop new FETI algorithms with certain improvements, however since they are in a basic stage they present several issues that leads to new sources of research.

Finally, we use the same idea that led to the S-FETI formulation but this time is applied to the FETI-2LM method, in order to develop a new Block version of this last one.

Before giving more details about FETI and the other FETI-like methods, we want to recall some of the basic linear algebra tools needed in this thesis to understand this type of algorithms, we refer to the iterative solvers for linear

systems which are one of the main elements in the different FETI methods.

Iterative methods

We start by showing the basic properties of the iterative $Krylov$ methods used in the solution of the FETI problems.

Krylov methods

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

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

With $A \in \mathcal{M}_{n\times n}(\mathbb{R})$ square real matrix, x the unknown, b the right hand side (rhs) known term, both $\mathbb{R}^n$ vectors.

Big part of this work is based on the resolution of a linear system of this type via an iterative method. First we will consider the more general case where A is invertible and our system has a unique solution. The most used methods to solve this kind of problems are the ones based on projections in a particular type of space, the $Krylov\ Space$.

From this we can build the Krylov methods, that consist on building some adequate subspace and project our solution in this space, all by just using simple operations such as matrix-vector products, dot products or linear combination of vectors.

This way the $Krylov\ Space$ can be defined by

Definition 0.1. *Lets 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*

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

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 with this definition 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 therfore

$$\begin{aligned} A^{p+q+1} g_0 &= \sum_{k=0}^{p-2} \alpha_k A^{k+1} g_0 + \alpha_{p-1} A^p g_0 \\ &= \sum_{k=0}^{p-2} \alpha_k A^{k+1} g_0 + \alpha_{p-1} \sum_{k=0}^{p-1} \beta_k A^k g_0 \\ &= \sum_{k=0}^{p-1} \gamma_k A^k g_0 \end{aligned} \tag{3}$$

□

Lemma 0.3. *The Krylov space succession its 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 $q > 0$.

We then 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

$$A^{p_{max}} g_0 = \sum_{k=0}^{p_{max}-1} \alpha_k A^k g_0 \tag{4}$$

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

$$A^{p_{max}-1} g_0 = \sum_{k=0}^{p_{max}-1} \alpha_k A^{k-1} g_0 \tag{5}$$

which is contradictory with the linear dependency of the vectors.

If we divide both terms in Equation 4 by α_0 and passing all terms to one side, we have

$$\begin{aligned} g_0 + \sum_{k=1}^{p_{max}-1} \frac{\alpha_k}{\alpha_0} A^k g_0 - \frac{1}{\alpha_0} A^{p_{max}} g_0 = 0 \Leftrightarrow \\ Ax_0 - b + \sum_{k=1}^{p_{max}-1} \frac{\alpha_k}{\alpha_0} A^k g_0 - \frac{1}{\alpha_0} A^{p_{max}} g_0 = 0 \Leftrightarrow \\ A\left(x_0 + \sum_{k=1}^{p_{max}-1} \frac{\alpha_k}{\alpha_0} A^{k-1} g_0 - \frac{1}{\alpha_0} A^{p_{max}-1} g_0 \right) = b \Leftrightarrow \end{aligned} \tag{6}$$

□

In practice to build this 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 about 16 iteration the new values of the succession $A^p g_0$ start to be linearly dependent, and depending on the matrix A some values are way too small or too big to be represented.

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

With the construction of the *Krylov space p* basis, that we call V_p, we can now attack the problem of finding an approximate solution x_p. We know, as shown previously that the real solution is in $x_0 + \mathcal{K}_{p_{max}}$ but since we are working in limited arithmetic we search a projected solution in the space $x_0 + \mathcal{K}_p$. The

Algorithm 1 Arnoldi iteration algorithm

Initialization
$g_0 = Ax_0 - b$
$v_1 = \frac{g_0}{\|g_0\|}$
loop Construction of the $j+1$ vector of the base
    $w = Av_j$
    for $i = 1$ to j **do**
        $\alpha_i = (w, v_i)$
        $w = w - \alpha_i v_i$
    end for
    $v_{j+1} = \frac{w}{\|w\|}$
end loop

solution can be written as

$$x_p = x_0 + V_p z_p \tag{7}$$

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

$$\begin{aligned} e_p &:= x_p - x = x_0 - x + V_p z_p = e_0 + V_p z_p \\ g_p &:= Ax_p - b = Ae_p = Ae_0 + AV_p z_p = g_0 + AV_p z_p \end{aligned} \tag{8}$$

A Krylov method consist, in one hand, of an algorithm to compute the base of the Krylov Space and on the other hand an optimal criteria to determine the approximate solution x_p. This is made, a priori, by minimizing the error or residual using an adapted norm.

Lanczos method

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

$$\begin{aligned} V_p^t V_p &= I_p \\ AV_p &= V_{p+1} H_{p+1,p} \end{aligned} \tag{9}$$

where

$$H_{p+1,p} := \begin{bmatrix} h_{11} & h_{12} & \dots & \dots & h_{1p} \\ h_{21} & h_{22} & \dots & \dots & h_{2p} \\ 0 & h_{32} & h_{33} & \dots & h_{3p} \\ \vdots & \ddots & \ddots & \dots & \vdots \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & 0 & h_{p+1,p} \end{bmatrix} \tag{10}$$

and so

$$H_p := H_{pp} = V_p^t A V_p \tag{11}$$

For the case of a symmetric matrix the matrix H_p is also symmetric an thus tridiagonal, so the algorithm for the construction of the Base of Arnoldi is simplified.

Algorithm 2 Algorithm of Lanczos

Initialization
$g_0 = Ax_0 - b$
$v_1 = \frac{g_0}{\|g_0\|}$
loop Construction of the $j+1$ vector of the base of Lanczos
    $w = Av_j$
    $h_{j,j-1} = h_{j-1,j}$
    $w = w - h_{j-1,j} v_{j-1}$
    $h_{jj} = (w \cdot v_j)$
    $w = w - h_{jj} v_j$
    $h_{j+1,j} = \|w\|$
    $v_{j+1} = \frac{w}{\|h_{j+1,j}\|}$
end loop

The algorithm described in 2 shows the construction of the base of Arnoldi

for symmetric matrices, now called *Basis of Lanczos*. This algorithm has the property of only use short recurrences in the computation so his cost is constant in every iteration.

If the matrix is also positive definite, the *Lanczos method* consist on minimize the error in the norm defined by A.

$$\mathcal{E}(x_p) = \|x_p - x\|_A^2 = (A(x_p - x) \cdot (x_p - x)) = (g_p \cdot e_p) \tag{12}$$

This approximate solution has the next 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 12 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 it is missing is the practical computation of x_p. To do so, of 12 and 8 we have that

$$\mathcal{E}(x_p) = (A(e_0 + V_p z_p) \cdot (e_0 + V_p z_p)) = (AV_p z_p \cdot V_p z_p) + 2(g_0 \cdot V_p z_p) + (g_0 \cdot e_0) \tag{13}$$

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

$$\begin{aligned} \mathcal{J}_p(z_p) &= \frac{1}{2}(V_p^t A V_p z_p \cdot z_p) + (V_p^t g_0 \cdot z_p) \\ &= \frac{1}{2}(T_p z_p \cdot z_p) + (y_p \cdot z_p) \end{aligned} \tag{14}$$

where $T_p = H_p$ is the matrix of the orthonormalization coefficients. This is a classical finite dimension minimization problem from were we have the next 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.

$$\begin{aligned}
&= \mathcal{J}(\alpha x + (1-\alpha)y) = \\
&= \frac{1}{2}\alpha^2(Ax \cdot x) + \alpha(1-\alpha)(Ax \cdot y) + \frac{1}{2}(1-\alpha)^2(Ay \cdot y) - \alpha(b \cdot x) - (1-\alpha)(b \cdot y) \\
&= \alpha\mathcal{J}(x) + (1-\alpha)\mathcal{J}(y) \\
&+ \frac{1}{2}\Big[(\alpha^2 - \alpha)(Ax \cdot x) + 2\alpha(1-\alpha)(Ax \cdot y) + ((1-\alpha)^2 - (1-\alpha))(Ay \cdot y)\Big] \\
&= \alpha\mathcal{J}(x) + (1-\alpha)\mathcal{J}(y) + \frac{1}{2}\alpha(\alpha-1)\left[(Ax \cdot x) - 2(Ax \cdot y) + (Ay \cdot y)\right]
\end{aligned} \tag{15}$$

Since A is positive definite, we have

$$(Ax \cdot x) - 2(Ax \cdot y) + (Ay \cdot y) = (A(x-y) \cdot (x-y)) > 0 \tag{16}$$

whenever $x \neq y$.

Now, if $\alpha \in]0,1[$, then $\alpha(1-\alpha) < 0$, hence

$$\frac{1}{2}\alpha(\alpha-1)\left[(Ax \cdot x) - 2(Ax \cdot y) + (Ay \cdot y)\right] < 0 \tag{17}$$

□

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

$$\mathcal{D}\mathcal{J}(x) \cdot y = (Ax \cdot y) - (b \cdot y) = ((Ax - b) \cdot y) \tag{18}$$

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

Corollary 0.9. *The minimum of $\mathcal{E}(x_p)$ defined in 13 is the point $x_p = x_0 + V_p z_p$, z_p being the solution of the system*

$$T_p z_p = -y_p \tag{19}$$

Proof. From previous theorem and lemma all we have to prove is that T_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 consist on building T_p and y_p then find z_p and replace it in 7 to find the approach solution x_p who minimizes the error in the A-norm.

One of the good things of this method is that the vectors of the base 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.

Conjugate gradient

The Lanczos method will be a much better algorithm if it could use a short recurrence for the calculation of the approach solution x_p. This can be done if the first component of z_p are the ones of z_{p-1} which will give a formula of the type

$$x_p = x_{p-1} + \alpha_p v_p \tag{20}$$

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. But we will not be able to compute the error $\mathcal{E}(x_p)$ because e_0 will be unknown, the only way to test the convergence of the method is by using the dimensionless residual

$$\frac{\|Ax_p - b\|}{\|b\|} < \epsilon \tag{21}$$

So we are forced to compute the successive gradients in order to control the method. Since $g_p \in \mathcal{K}_{p+1} \cap \mathcal{K}_p^{\perp}$ 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 it goes to zero, because we will stop before have any representation problem.

Let $G_p = (g_0, g_1, \dots, g_{p-1})$ as we just saw, $G_p = V_p \Delta_p$ where Δ_p is a diagonal matrix. The projection matrix is also tridiagonal symmetric positive definite

$$G_p^t A G_p = \Delta_p^t V_p^t A V_p \Delta_p = \Delta_p T_p \Delta_p = \tilde{T}_p \tag{22}$$

it admits the factorization $\tilde{T}_p = \tilde{L}_p \tilde{D}_p \tilde{L}_p^t$ and so

$$G_p^t A G_p = \tilde{L}_p \tilde{D}_p \tilde{L}_p^t \Leftrightarrow \tilde{L}_p^{-1} G_p^t A G_p \tilde{L}_p^{-t} = \tilde{D}_p \tag{23}$$

The previous 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 $\mathcal{K}_p$.

Since the projection matrix $W_p^t A W_p$ is diagonal, this base is ideal to use a short recurrence in the computation of the solution of the optimization problem 12, as it can be built using the relation

$$W_p \tilde{L}_p^t = G_p \tag{24}$$

Let $(w_0, w_1, \dots, w_{p-1})$ be column vectors of W_p and $(\gamma_0, \gamma_1, \dots)$ the sub-diagonal elements of $\tilde{L}_p^{-t}$, the equation 24 implies

$$w_0 = g_0 \text{ and } \gamma_{j-1} w_{j-1} + w_j = g_j \, \forall j > 0 \tag{25}$$

With the different relations between x_p, g_p and w_p we can formulate a new method without only using short recurrences in the construction of every new vector. Actually, from the previous equation we have

$$g_0 = A x_0 - b \text{ and } w_0 = g_0 \tag{26}$$

From the properties of the base W_p we have

$$x_p = x_{p-1} + \rho_{p-1} w_{p-1} \Leftrightarrow g_p = g_{p-1} + \rho_{p-1} A w_{p-1} \tag{27}$$

We know that g_p is orthogonal to $\mathcal{K}_p$ and so to w_{p-1}, this allows us to obtain

ρ_{p-1}

$$(g_p \dot{w}_{p-1}) = (g_{p-1} \cdot w_{p-1}) + \rho_{p-1}(Aw_{p-1} \cdot w_{p-1}) = 0 \Leftrightarrow \rho_{p-1} = -\frac{(g_{p-1} \cdot w_{p-1})}{(Aw_{p-1} \cdot w_{p-1})} \quad (28)$$

From the equation 25 we can build the new w_p with the previous w_{p-1} and g_p

$$w_p = g_p - \gamma_{p-1} w_{p-1} \quad (29)$$

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

$$(w_p \cdot Aw_{p-1}) = (g_p \cdot Aw_{p-1}) - \gamma_{p-1}(w_{p-1} \cdot Aw_{p-1}) = 0 \Leftrightarrow \gamma_{p-1} = \frac{(g_p \cdot Aw_{p-1})}{(Aw_{p-1} \cdot w_{p-1})} \quad (30)$$

The method defined this way i called the $Conjugate gradient$ method and it can be resumed in 3.

Algorithm 3 Conjugate gradient method

1: **Initialization**
2: $g_0 = Ax_0 - b$
3: $w_0 = g_0$
4: **loop** Iteration of the CG method
5: $\rho_{p-1} = -(g_{p-1} \cdot w_{p-1})/(Aw_{p-1} \cdot w_{p-1})$
6: $x_p = x_{p-1} + \rho_{p-1} w_{p-1}$
7: $g_p = g_{p-1} + \rho_{p-1} Aw_{p-1}$
8: **if** $(g_p \cdot g_p)/(b \cdot b) < \epsilon^2$ **then**
9: End
10: **end if**
11: $\gamma_{p-1} = (g_p \cdot Aw_{p-1})/(Aw_{p-1} \cdot w_{p-1})$
12: $w_p = g_p - \gamma_{p-1} w_{p-1}$
13: **end loop**

From a theoretical point of view, we do not need any orthogonalization

to build the new w_p descent direction, but in limited arithmetic, errors are transmitted from each actualization, so 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 this 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

$$\gamma_i = \frac{(g_p \cdot Aw_i)}{(Aw_i \cdot Aw_i)}, \qquad i = 0, \ldots, p-1 \tag{31}$$

and

$$w_p = g_p - \sum_{i=0}^{p-1} \gamma_i w_i \tag{32}$$

ORTHODIR

In the case if a non symmetric matrix, the H_p matrix is no longer tridiagonal. We can not expect to have short recurrence to build an orthogonal basis of the Krylov space $\mathcal{K}_p$. Also, A does not define an inner product and the criteria of optimality $\mathcal{E}(x_p)$ may not be used. The logical choice for a stopping criteria is the computation of the square norm of the residual

$$\mathcal{R}(x_p) = (A(x_p - x) \cdot A(x_p - x)) = (g_p \cdot g_p) = \|x_p - x\|^2_{A^tA} = (A^tA(x_p - x) \cdot (x_p - x)) \tag{33}$$

We have similar properties as 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 to A^tA.*

Proof. Direct from equation 33. □

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

Proof. From the properties of projection in an affine space

$$(A^tA(x_p - x) \cdot w_p) = (A(x_p - x) \cdot Aw_p) = (g_p \cdot Aw_p) = 0, \forall w_p \in \mathcal{K}_p \tag{34}$$

□

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

$$A^tAx = A^tb \tag{35}$$

This equation is call the "$Normal\,equation$" and it does not have a very good conditioning since it can be as much as the square of the original conditioning of A.

The best is to compute the solution using a short recurrence, so from the theorem 0.10 we know that the any A^tA-orthogonal base of $\mathcal{K}_p$, W_p implies that $W_p^tA^tAW_p$ should be diagonal.

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

$$\begin{aligned} x_p &= x_{p-1} + \rho_p w_p \\ g_p &= g_{p-1} + \rho_p A w_p \end{aligned} \tag{36}$$

and from the A^tA-orthonormal properties we have

$$(g_p \cdot Aw_p) = 0 \Leftrightarrow (g_{p-1} \cdot Aw_p) + \rho_p(Aw_p \cdot Aw_p) = 0 \Leftrightarrow \rho_p = -(g_{p-1} \cdot Aw_p) \tag{37}$$

and so the algorithm is described in 4

This algorithm forces to save the vectors of the base, but is numerically stable,

Algorithm 4 ORTHODIR method

1: **Initialization**
2: $g_0 = Ax_0 - b$
3: $w_0 = g_0$
4: **loop** Iterate $p = 1, \ldots,$ 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: $\quad g_p = g_{p-1} + \rho A w_{p-1}$
8: $\quad w_p = A w_{p-1}$
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**

in any case we will deal with this issue in the next chapters.

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 direction, i.e

$$w_p = g_p \tag{38}$$

Parallelization of Krylov methods

The codes presented previously, represent a sequential algorithm, however they can be easily transformed into its respective parallel version.

The implementation of the parallel version of this type of methods is base on a message-passing standard, in practical terms this implies the use of libraries that use the most common parallel computing architectures, the most common communication protocol is the so-called Message Passing Interface (MPI) [40].

Using this protocol we only add two changes to the usual sequential algorithm

1. Exchange of data between processes (or subdomain in the domain decomposition framework) to assemble the matrix-vector products. The details

of how to compute this products will be given when presenting the first domain decomposition method used in this work.

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

Using the MPI libraries, or in general any communication protocol, needed to do the exchanges in 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.

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

Chapter 1

Hybrid FETI method

In the domain decomposition framework, the Finite Element Tearing and Interconnecting (FETI) methods have proven to be very effective in the solution of real engineering applications during the last years. This is one of the reason why its current development keeps up to this day, always searching for faster, precise and robust new versions. In this context we have developed a new method built from the base of two existing FETI methods, namely the FETI-1LM and FETI-2LM. This new method tries to recover the good properties of each method, in configurations where the use of one or the other is not so clear. For the development of this new method, first we need to understand the basics of both basic FETI methods from a theoretical point of view, but also the implementation which will be crucial to show how the new method works.

This chapter is presented first introducing the FETI method in his classic version, including the preconditioners used to achieve good performance and some implementation considerations. Then the FETI method with two lagrange multipliers is explained to finally show the new hybrid method that arises from mixing the two previous FETI methods. The chapter ends with numerical results for all the methods.

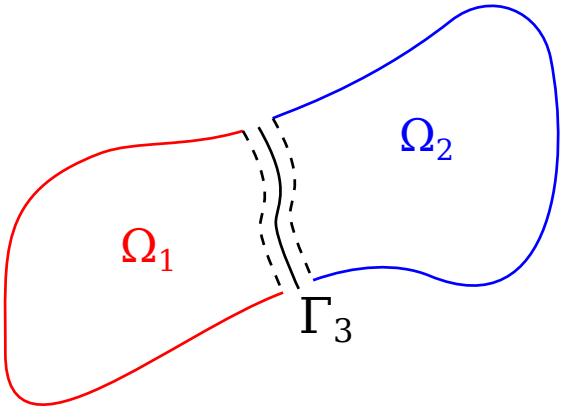

Figure 1.1 – Two subdomain splitting

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 later be extended 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. To find his variational form, the Stokes formula is used in 1.1, so 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. This 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 this equations, but the contrary is not always true, due to the differences that occur in $\Gamma^{(3)}$.

We continue by using again the Stokes formula 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, his restriction in the subdomains $\Omega^{(s)}$ will be continuous on the interface $\Gamma^{(3)}$. On the other hand two functions $u^{(s)}$ that satisfies 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 shows that a new condition is necessary to have an equivalence between the solution of the global problem and the solution of the local ones. For u to be in $H^1_0(\Omega)$ this admissibility condition imposes the continuity on the interface

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

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 consist in introducing boundary conditions on $\Gamma^{(3)}$ to complement the local equations 1.3 and to iteratively find the values of these boundary conditions for which both continuity 1.6 and equilibrium 1.7 interface conditions are satisfied, meaning that this solution will be the exact same as the global searched one.

Depending on the condition imposed, two 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 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 reduce the computation 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 to find u^3 whose residual is equal to $\frac{\partial u^{(1)}}{\partial n^{(1)}} + \frac{\partial u^{(2)}}{\partial n^{(2)}}$.

The FETI method is based in 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, $u^{(1)}|_{\Gamma^{(3)}}$ 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 $u^{(1)}|_{\Gamma^{(3)}} = u^{(2)}|_{\Gamma^{(3)}}$. This condition can be written under the weak form

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

Now, consider the Lagrangian

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

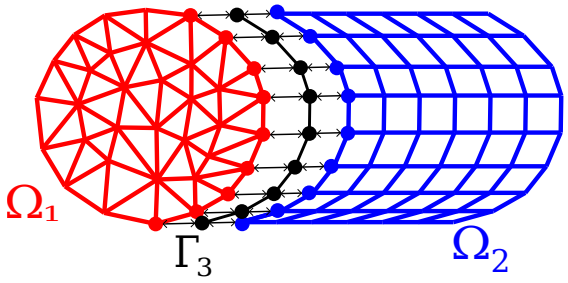

Figure 1.2 – Two subdomain divisions with duplicated nodes.

and 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

Lets consider a discretization of variational equation 1.2 using a finite element method. This process works for different elliptic partial differential equations and different finite element discretizations. So from now on we can consider this as a more general work, as long as we have this type of discretization 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, where subscripts 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 $\Gamma^{(3)} = \partial\Omega^{(1)} \cap \Omega^{(2)}$

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

The formulation of each local discretization matrix is made considering that each subdomain has its own mesh and also that the nodes of the interface $\Gamma^{(3)}$ are shared by both meshes as in 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 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 systems 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 and 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 linking the trace and the flux of a vector satisfying the inner subset is derived

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

$S_{bb}^{(s)} = K_{bb}^{(s)} - K_{bi}^{(s)} K_{ii}^{(s)^{-1}} K_{ib}^{(s)}$ 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

$$\begin{cases} K_{ii}^{(1)} x_i^{(1)} + K_{ib}^{(1)} x_b^{(1)} &= f_i^{(1)} \\ K_{ii}^{(2)} x_i^{(2)} + K_{ib}^{(2)} x_b^{(2)} &= f_i^{(2)} \end{cases} \tag{1.25}$$

- 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 we have the continuity relation 1.27 and 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 and 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 PDEs.

The inner equations 1.25 are common solution vectors of local problems for any kind of boundary conditions on $\Gamma^{(3)}$. Equations 1.27 and 1.26 are the actual condensed interface problem since the inner equations 1.25 establish that $x_i^{(1)}$ and $x_i^{(2)}$ can be derived from $x_b^{(1)}$ and $x_b^{(2)}$.

1.1.2 FETI one lagrange multiplier

The previous ideas are now formalized for the extended case where we have $N_s > 2$ subdomains and and interface Γ, defined as

$$\Gamma = \bigcup_{1 \leq s,q \leq N_s} (\partial\Omega^{(s)} \cap \partial\Omega^{(q)}) \tag{1.28}$$

In the FETI method the discrete flux, noted λ, is the unknown defined in the nodes along the interface and the jump of the solutions of the local Neumann problems are the gradient of the condensed interface problem. The discretization of the local Neumann problems in some subdomain s, can also be written as

$$K^{(s)}x^{(s)} = f^{(s)} + t^{(s)^T} B^{(s)^T} \lambda \tag{1.29}$$

where $K^{(s)}$ is the local stiffness matrix, $f^{(s)}$ the right-hand side vector, $t^{(s)} \in \mathcal{M}_{\#(\partial\Omega^{(s)})\times\#(\Omega^{(s)})}$ are trace operators which extracts boundary degrees of freedom from subdomain $\Omega^{(s)}$ and $B^{(s)} \in \mathcal{M}_{\#(\Gamma)\times\#(\partial\Omega^{(s)})}$ are discrete assembling matrices which connects pairwise degrees of freedom on the interface. In a general case, the stiffness matrix $K^{(s)}$, the local solution vector $x^{(s)}$ and the local right-hand side vector $f^{(s)}$ are defined as

$$K^{(s)} = \begin{bmatrix} K_{ii}^{(s)} & K_{ib}^{(s)} \\ K_{bi}^{(s)} & K_{bb}^{(s)} \end{bmatrix} \tag{1.30}$$

$$x^{(s)} = \begin{bmatrix} x_i^{(s)} \\ x_b^{(s)} \end{bmatrix}, f^{(s)} = \begin{bmatrix} f_i^{(s)} \\ f_b^{(s)} \end{bmatrix} \tag{1.31}$$

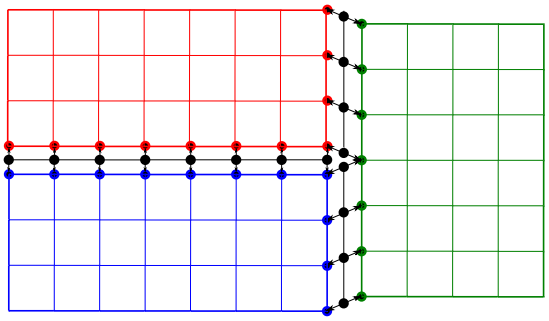

Figure 1.3 – Multiple interface node.

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

$$x_b^{(s)} = t^{(s)} x^{(s)} \tag{1.32}$$

The discrete operator $B^{(s)}$ is a defined as the mapping of a vector in local interface $\partial\Omega^{(s)}$ on the complete interface, so applied to the solution on the local interface $x_b^{(s)}$ we can write the continuity condition across the total interface, degree of freedom per degree of freedom

$$\sum_s B^{(s)} t^{(s)} x^{(s)} = 0 \tag{1.33}$$

the restriction of $B^{(s)}$ on interface Γ^{ij}, 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 that it's in more than two subdomains, Figure 1.3, will generate the same number of continuity conditions and flux as the number of interfaces who shares it.

With the definition of $B^{(s)}$ and $t^{(s)}$, the solutions $x^{(s)}$ of 1.29 and 1.33 are the searched restrictions in every subdomain of the global discrete solution of 1.17. The vectors $x^{(s)}$ are actually continuous and the definition of $B^{(s)}$ is that ${t^{(s)}}^T {B^{(s)}}^T \lambda$ is zero for the inner nodes of $\Omega^{(s)}$ and in the interface we have ${B^{(ij)}}^T \lambda + {B^{(ji)}}^T \lambda = 0$ thanks to the opposite sign. So, again, the assembly of local discrete Neumann equations 1.29 gives exactly global discrete equation 1.16.

The gradient of the condensed interface problem is defined as

$$g = \sum_s B^{(s)} t^{(s)} x^{(s)} \tag{1.34}$$

where $x^{(s)}$ is solution of the local discrete Neumann problem 1.29. Continuity relation 1.33 defines 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 in $\Omega^{(s)}$, so the local discrete Neumann equations 1.29 are ill posed. If $K^{(s)}$ comes from the Laplace equation, its kernel are the constant fields 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 in the matrix $K^{(s)}$ to achieve this, and also because it allows to compute 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 1.29 can be written as a particular solution plus an undefined element of the kernel of $K^{(s)}$

$$x^{(s)} = K^{(s)^+}(f^{(s)} + t^{(s)^T} B^{(s)^T} \lambda) + R^{(s)} \alpha^{(s)} \tag{1.35}$$

From equation 1.29 we see that the right-hand side belongs to the range space of matrix $K^{(s)}$, and so is orthogonal to the kernel. This orthogonality constraint can be written

$$R^{(s)^T} \left(f^{(s)} + t^{(s)^T} B^{(s)^T} \lambda \right) = 0 \tag{1.36}$$

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

Condensed interface problem

Replacing $x^{(s)}$ from equation 1.35 in the continuity condition 1.33 we have

$$\sum_s B^{(s)} t^{(s)} K^{(s)^+} t^{(s)^T} B^{(s)^T} \lambda + \sum_s B^{(s)} t^{(s)} R^{(s)} \alpha^{(s)} = -\sum_s B^{(s)} t^{(s)} K^{(s)^+} f^{(s)} \tag{1.37}$$

To build the condensed interface problem this previous equation is used, and the equation 1.36, leading to the problem to be satisfied by λ and the vector of coefficients of the kernel components α

$$\begin{bmatrix} F & G \\ G^T & 0 \end{bmatrix} \begin{bmatrix} \lambda \\ \alpha \end{bmatrix} = \begin{bmatrix} d \\ c \end{bmatrix} \tag{1.38}$$

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 $\Omega^{(s)}$
- $G^T \lambda = \left(\ldots, B^{(s)} t^{(s)} R^{(s)}, \ldots \right)^T \lambda$
- $d = -\sum_s B^{(s)} t^{(s)} K^{(s)^+} f^{(s)}$
- $c = \left(\ldots, -b^{(s)^T} R^{(s)}, \ldots \right)^T$

The condensed interface system 1.38 is an hybrid system. It's solution λ satisfies the following orthogonality condition

$$\mu^T F \lambda = \mu^T d, \forall \mu / G^T \mu = 0 \tag{1.39}$$

Now consider λ_0, for example

$$\lambda_0 = AG(G^T AG)^{-1} c \tag{1.40}$$

where A is a matrix symmetric positive definite that is usually taken as being the identity, but it can also be defined as the preconditioner, later to be defined, or some scaling matrix. For details, see [62].

Then this λ_0 satisfies the admissibility constraint of equation 1.36 and $G^T(\lambda - \lambda_0) = 0$. So, if P is any projector in the kernel of G^T, then from equation 1.39 we have that λ is the solution of the following projected problem

$$P^T F P(\lambda - \lambda_0) = P^T(d - F\lambda_0) \tag{1.41}$$

The FETI method solves iteratively via a conjugate gradient algorithm the previous projected condensed interface problem 1.41, using the orthogonal projector in the kernel of G^T.

Interpretation of projector P

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

$$P = I - AG\left(G^T AG\right)^{-1} G^T \tag{1.42}$$

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

$$\left(G^T AG\right)\alpha = -G^T g \tag{1.43}$$

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

$$g^p = \sum_s B^{(s)} S_{bb}^{(s)^+} B^{(s)^T} \lambda^p + \sum_s B^{(s)} t^{(s)} K^{(s)^+} f^{(s)} = \sum_s B^{(s)} t^{(s)} x^{(s)^{p+}} \tag{1.44}$$

where u_i^{p+} is the solution of the local Neumann problems, computed using the pseudo-inverse matrices

$$x^{(s)^{p+}} = K^{(s)^+}\left(f^{(s)} + t^{(s)^T} B^{(s)^T} \lambda^p\right) \tag{1.45}$$

so the gradient is equal to the jump of this particular solutions. From equations

1.42 and 1.43, the projected gradient Pg^p is

$$Pg^p = g^p + AG\alpha^p = \sum_s B^{(s)} t^{(s)} x^{(s)p+} + \sum_s B^{(s)} t^{(s)} R^{(s)} \alpha^{(s)p} \tag{1.46}$$

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

$$x^{(s)p} = x^{(s)p+} + R^{(s)} \alpha^{(s)p} \tag{1.47}$$

The definition of the constraint 1.36 associated with the orthogonal projector 1.43 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

$$G^T Pg^p = 0 \Leftrightarrow \left(B^{(s)} t^{(s)} R^{(s)} \right)^T Pg^p = 0 \qquad \forall s \tag{1.48}$$

Computing the projected gradient Pg^p consists in fact in 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 solution that ensures a convergence rate for the overall FETI process asymptotically independent upon the number of subdomains. 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. It's mandatory 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 subdomain s to the condensed interface

operator, defined in1.37, is

$$B^{(s)}t^{(s)}K^{(s)^+}t^{(s)^T}B^{(s)^T} \tag{1.49}$$

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

$$t^{(s)}K^{(s)^+}t^{(s)^T} = (K^{(s)}_{bb} - K^{(s)}_{bi}K^{(s)^{-1}}_{ii}K^{(s)}_{ib})^+ =: S^{(s)^+}_{bb} \tag{1.50}$$

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

$$D^{-1} = \sum_s B^{(s)}S^{(s)}_{bb}B^{(s)^T} \tag{1.51}$$

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

$$\left(\sum_s B^{(s)}S^{(s)^+}_{bb}B^{(s)^T}\right)^+ \simeq \sum_s B^{(s)}S^{(s)}_{bb}B^{(s)^T} \tag{1.52}$$

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

$$\begin{bmatrix} K^{(s)}_{ii} & K^{(s)}_{ib} \\ 0 & I \end{bmatrix}\begin{bmatrix} \tilde{w}^{(s)}_i \\ \tilde{w}^{(s)}_b \end{bmatrix} = t^{(s)^T}B^{(s)^T}w = \begin{bmatrix} 0 \\ w^{(s)}_b \end{bmatrix} \tag{1.53}$$

whose solution is

$$\tilde{w}^{(s)}_i = -K^{(s)^{-1}}_{ii}K^{(s)}_{ib}w^{(s)}_b \tag{1.54}$$

and multiplying by the stiffness matrix to obtain

$$\begin{bmatrix} K^{(s)}_{ii} & K^{(s)}_{ib} \\ K^{(s)}_{bi} & K^{(s)}_{bb} \end{bmatrix}\begin{bmatrix} \tilde{w}^{(s)}_i \\ \tilde{w}^{(s)}_b \end{bmatrix} = \begin{bmatrix} 0 \\ (K^{(s)}_{bb} - K^{(s)}_{bi}K^{(s)^{-1}}_{ii}K^{(s)}_{ib})\tilde{w}^{(s)}_b \end{bmatrix} = \begin{bmatrix} 0 \\ S^{(s)}_{bb}w^{(s)}_b \end{bmatrix} \tag{1.55}$$

With this preconditioner it has been proved [53],[29], that the FETI method is asymptotically independent upon the mesh size. With a condition number for the projected condensed operator bounded by

$$C(1+\log(\frac{H}{h}))^2 \tag{1.56}$$

where h is the mesh size and H is the characteristic subdomain size. Meaning, that when decreasing H the number of subdomains increases and when decreasing h the finite element mesh is refined (more elements).

So, with both local and global preconditioners, the number of iterations does not depend anymore on both the number of subdomain nor the mesh size. A second preconditioner, not mathematically optimal can be introduced, as an alternative to the cost of implementation of the Dirichlet one, the so called "Lumped" preconditioner that can de defined as

$$L^{-1} = \sum_s B^{(s)} K_{bb}^{(s)} B^{(s)^T} \tag{1.57}$$

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

$$\tilde{B}^{(s)} := \beta^{(s)} B^{(s)} \tag{1.58}$$

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 mechanical consistent combination of the interface reaction forces from the Dirichlet problem in each subdomain. With

this new scaled assembling operators, the preconditioners are written as

$$D^{-1} = \sum_s \tilde{B}^{(s)} S_{bb}^{(s)} \tilde{B}^{(s)^T} \tag{1.59}$$

$$L^{-1} = \sum_s \tilde{B}^{(s)} K_{bb}^{(s)} \tilde{B}^{(s)^T} \tag{1.60}$$

The behaviour of both preconditioner can be seen [29], [61] and will be evocated in later sections.

1.1.4 FETI Algorithms

Using the definitions in 1.38 and the projector in 1.42, the CG algorithm to solve the FETI problem 1.41 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), because in limited arithmetic the orthogonal properties of the CG method are lost, specially in the context

of the FETI methods, where the multiplication by the operator is not totally 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
    $\rho_p = -\dfrac{(Fw_p)^T g_p}{(Fw_p)^T (Fw_p)}$
    $\lambda_{p+1} = \lambda_p + \rho_p w_p$
    $g_{p+1} = g_p + \rho_p Fw_p$
    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 works when the matrix F is symmetric positive definite, for a more general case when our FETI operator F is no longer symmetric or non positive definite, as we will see in later sections, the CG algorithm is no longer appropriate, we use instead the ORTHODIR method with left preconditioner for unsymmetric matrices, mainly for its implementation, simplicity and good properties, as is equivalent to the GMRES algorithm [50, Chapter 12]. One of the main theoretical differences between them two is the storage of previous computed directions which, in any case, it is done to perform

a full reorthogonalization that allows the loss of orthogonality of the search directions and keeps a good convergence ratio.

The full description is done in Algorithm 6.

1.2 FETI with two lagrange multipliers

1.2.1 FETI-2LM

We now present the second FETI method that is used as a base to build our method. Originally introduced in [35] as a solver for acoustic problems then extended in [67] and [68] as a robust solver for more general problems. The basic idea of this method is to impose Robin boundary conditions in the interface to "glue" the local solutions in order for them to be the restrictions of the global solution. To have a method useful for different approximations of elliptic PDE, it will as in FETI, be derived from linear algebra.

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

$$K^{(s)} = \begin{bmatrix} K_{ii}^{(s)} & K_{ib}^{(s)} \\ K_{bi}^{(s)} & K_{bb}^{(s)} \end{bmatrix}, \qquad f^{(s)} = \begin{bmatrix} f_i^{(s)} \\ f_b^{(s)} \end{bmatrix} \tag{1.61}$$

where $K_{bb}^{(s)}$ represents the interaction matrices between the nodes on the interface and those from 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

$$\begin{bmatrix} K_{ii}^{(1)} & 0 & K_{ib}^{(1)} \\ 0 & K_{ii}^{(2)} & K_{ib}^{(2)} \\ K_{bi}^{(1)} & K_{bi}^{(2)} & K_{bb} \end{bmatrix} \begin{bmatrix} x_i^{(1)} \\ x_i^{(2)} \\ x_b \end{bmatrix} = \begin{bmatrix} f_i^{(1)} \\ f_i^{(2)} \\ f_b \end{bmatrix} \tag{1.62}$$

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

$$\begin{bmatrix} K_{ii}^{(1)} & K_{ib}^{(1)} \\ K_{bi}^{(1)} & K_{bb}^{(1)} + (K_{bb}^{(2)} - K_{bi}^{(2)} K_{ii}^{(2)^{-1}} K_{ib}^{(2)}) \end{bmatrix} \begin{bmatrix} x_i^{(1)} \\ x_b \end{bmatrix} = \begin{bmatrix} f_i^{(1)} \\ f_b^{(1)} + (f_b^{(2)} - K_{bi}^{(2)} K_{ii}^{(2)^{-1}} f_i^{(2)}) \end{bmatrix} \tag{1.63}$$

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

$$\begin{bmatrix} K_{ii}^{(s)} & K_{ib}^{(s)} \\ K_{bi}^{(s)} & K_{bb}^{(s)} + S_{bb}^{(q)} \end{bmatrix} \begin{bmatrix} x_i^{(s)} \\ x_b \end{bmatrix} = \begin{bmatrix} f_i^{(s)} \\ f_b^{(s)} + c_b^{(q)} \end{bmatrix} \tag{1.64}$$

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 in the subdomain q, opposite to s.

Previous equation means that the restriction of the global solution is, in each subdomain, a local solution of a problem with generalized Robin boundary condition in 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 behaviour 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 as the rest of the domain forces it to do. Then, the local problem formulation takes exactly 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 interfaced should be set up independently. This leads to introduce two

independent interface variables, one for each side.

$$\begin{bmatrix} K_{ii}^{(s)} & K_{ib}^{(s)} \\ K_{bi}^{(s)} & K_{bb}^{(s)} + A_b^{(s)} \end{bmatrix} \begin{bmatrix} x_i^{(s)} \\ x_b^{(s)} \end{bmatrix} = \begin{bmatrix} f_i^{(s)} \\ f_b^{(s)} + \lambda^{(s)} \end{bmatrix} \tag{1.65}$$

from this local equations, inner equations can be eliminated to get the equivalent local condensed problem

$$\begin{aligned} (K_{bb}^{(s)} - K_{bi}^{(s)} K_{ii}^{(s)^{-1}} K_{ib}^{(s)} + A_b^{(s)}) + x_b^{(s)} &= f_b^{(s)} - K_{bi}^{(s)} K_{ii}^{(s)^{-1}} f_i^{(s)} + \lambda^{(s)} \\ (S_{bb}^{(s)} + A_b^{(s)}) x_b^{(s)} &= c_b^{(s)} + \lambda^{(s)} \end{aligned} \tag{1.66}$$

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

$$x_b^{(s)} = (S_{bb}^{(s)} + A_b^{(s)})^{-1} \lambda^{(s)} + (S_{bb}^{(s)} + A_b^{(s)})^{-1} c_b^{(s)} \tag{1.67}$$

Since the boundary conditions on both sides are not consistent, the two interface conditions 1.27 and 1.26 must be enforced to make the solutions of the local problems 1.65 to be the restrictions of the solution of the global problem 1.17. Thanks to the second line of local equation 1.65, the interface equilibrium condition gives

$$A_b^{(1)} x_b^{(1)} + A_b^{(2)} x_b^{(2)} = \lambda^{(1)} + \lambda^{(2)} \tag{1.68}$$

The residual of the condensed interface problem is then

$$\begin{aligned} x_b^{(1)} - x_b^{(2)} &= 0 \\ \lambda^{(1)} + \lambda^{(2)} - A_b^{(1)} x_b^{(1)} - A_b^{(2)} x_b^{(2)} &= 0 \end{aligned} \tag{1.69}$$

where $x_b^{(s)}$ is a function of $\lambda^{(s)}$ according to equation (1.67). This two equations, combined with the local problem 1.65 are equivalent to the global block problem 1.62. In this form, equations 1.69 have a mixed form, to avoid this, a new combination of both equations is done, leading to a new condensed interface

problem

$$\begin{aligned} \lambda^{(1)} + \lambda^{(2)} - (A_b^{(1)} + A_b^{(2)})x_b^{(2)} = 0 \\ \lambda^{(2)} + \lambda^{(1)} - (A_b^{(2)} + A_b^{(1)})x_b^{(1)} = 0 \end{aligned} \tag{1.70}$$

This two forms are equivalent if the sum of the augmentation matrices $A_b^{(1)} + A_b^{(2)}$ is non singular. This is of course satisfied when 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, computing $x_b^{(s)}$ from 1.67 and using it in (1.70), the condensed interface problem used to compute $\lambda^{(1)}$ and $\lambda^{(2)}$, can be explicitly defined

$$\begin{aligned} \begin{bmatrix} I & I - (A_b^{(1)} + A_b^{(2)})(S_{bb}^{(2)} + A_b^{(2)})^{-1} \\ I - (A_b^{(2)} + A_b^{(1)})(S_{bb}^{(1)} + A_b^{(1)})^{-1} & I \end{bmatrix} \begin{bmatrix} \lambda^{(1)} \\ \lambda^{(2)} \end{bmatrix} \\ = \begin{bmatrix} (A_b^{(1)} + A_b^{(2)})(S_{bb}^{(2)} + A_b^{(2)})^{-1} c_b^{(2)} \\ (A_b^{(2)} + A_b^{(1)})(S_{bb}^{(1)} + A_b^{(1)})^{-1} c_b^{(1)} \end{bmatrix} \end{aligned} \tag{1.71}$$

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

1.2.2 Arbitrary mesh partition

Lets 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 2 subdomain case and apply it to the different interfaces created between neighbours 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

$$\Gamma = \bigcup_{1 \le s,q \le N_s} (\partial\Omega^{(s)} \cap \partial\Omega^{(q)}) \tag{1.72}$$

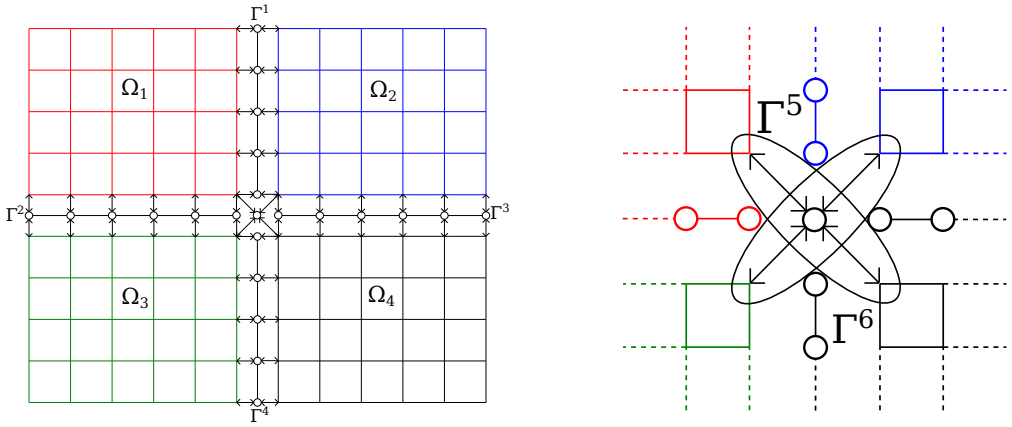

Figure 1.4 – Γ^j division example and crosspoint detail.

and we can define more precisely Γ^j as

$$\Gamma^j := \Gamma^{(sq)} = \partial\Omega^{(s)} \cap \partial\Omega^{(q)}, \qquad \forall s,q : \partial\Omega^{(s)} \cap \partial\Omega^{(q)} \neq \varnothing \tag{1.73}$$

The crosspoint 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 two edges were created in the crosspoint, as diagonal subdomains are neighbours. For each interface edge Γ^j and for each subdomain $\Omega^{(s)}$ that intersects it, lets define a boolean matrix $B^{(s)}_{\Gamma^j} \in \mathcal{M}_{dim(\Gamma^j)\times dim(\Omega^{(s)})}(\mathbb{R})$ by

$$B^{(s)}_{\Gamma^j} v^{(s)} = v^{(s)}_b|_{\Gamma^j} \tag{1.74}$$

where the vector $v^{(s)}$ defined in all the nodes of subdomain 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 can produce redundancies in the formulation of the problem. In practice this 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 previous definitions we can rewrite the local problem 1.65 and the

continuity conditions 1.69 for the general case

$$\begin{aligned}
\left(K^{(s)} + \sum_{\Gamma^j \subseteq \Omega^{(s)}} B_{\Gamma^j}^{(s)^T} A_{\Gamma^j}^{(s)} B_{\Gamma^j}^{(s)}\right) x^{(s)} &= f^{(s)} - \sum_{\Gamma^l \subseteq \Omega^{(s)}} B_{\Gamma^l}^{(s)^T} \lambda_{\Gamma^l}^{(s)}, \qquad s = 1, \ldots, N_s \\
B_{\Gamma^j}^{(s)} x^{(s)} - x_b|_{\Gamma^j} &= 0 \qquad \forall \Gamma^j, \quad \Omega^{(s)} \supseteq \Gamma^j \\
\sum_{\Omega^{(q)} \supseteq \Gamma^j} \left(\lambda_{\Gamma^j}^{(q)} - A_{\Gamma^j}^{(q)} x_b|_{\Gamma^j}\right) &= 0 \qquad \forall \Gamma^j
\end{aligned} \tag{1.75}$$

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 multipliers in the interface edge Γ^j on the side of the subdomain s and the vector $x_b|_{\Gamma^j}$ is the restriction of the global unknown x to Γ^j. Lets define also, for every $\Omega^{(s)}$ the total augmentation matrix

$$A^{(s)} := \sum_{\Gamma^j \subseteq \Omega^{(s)}} B_{\Gamma^j}^{(s)^T} A_{\Gamma^j}^{(s)} B_{\Gamma^j}^{(s)} \tag{1.76}$$

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

$$\begin{aligned}
\sum_{\Omega^{(q)} \supseteq \Gamma^j} &\left(\lambda_{\Gamma^j}^{(q)} - A_{\Gamma^j}^{(q)} B_{\Gamma^j}^{(s)} \left(K^{(s)} + A^{(s)}\right)^{-1} \sum_{\Gamma^l \subseteq \Omega^{(s)}} B_{\Gamma^l}^{(s)^T} \lambda_{\Gamma^l}^{(s)}\right) \\
&= - \sum_{\Omega^{(q)} \supseteq \Gamma^j} A_{\Gamma^j}^{(q)} B_{\Gamma^j}^{(s)} \left(K^{(s)} + A^{(s)}\right)^{-1} f^{(s)}, \qquad \forall \Gamma^j, \forall \Omega^{(s)} \supseteq \Gamma^j
\end{aligned} \tag{1.77}$$

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 were 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, but the use of two independent lagrange multipliers grant a good handle of this problems and

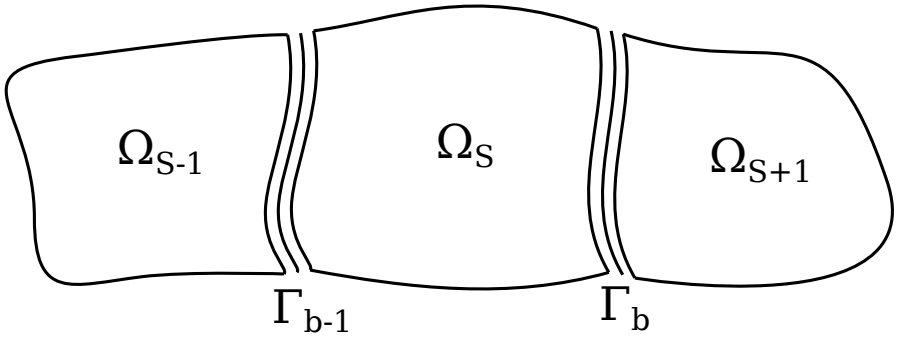

Figure 1.5 – One way splitting.

convergence is achieved, making the FETI-2LM a more robust method.

1.2.3 Optimal Interface Boundary Conditions

As seen in the definition of the FETI-2LM method, a big part of it, comes from the definition of the augmentation matrix $A^{(s)}$. Different work as been done to achieve the best numerical choice, next we will show some of them.

From a theoretical point of view, in the case of 2 subdomain the optimal augmentation matrices are defined as

$$\begin{aligned} A_b^{(1)} &:= S_{bb}^{(2)} \\ A_b^{(2)} &:= S_{bb}^{(1)} \end{aligned} \tag{1.78}$$

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 1.78 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 1.71 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

$$K^{(s)} = \begin{bmatrix} K_{ii}^{(s)} & K_{ib-1}^{(s)} & K_{ib}^{(s)} \\ K_{b-1i}^{(s)} & K_{b-1b-1}^{(s)} & 0 \\ K_{bi}^{(s)} & 0 & K_{bb}^{(s)} \end{bmatrix} \quad , \quad f^{(s)} = \begin{bmatrix} f_i^{(s)} \\ f_{b-1}^{(s)} \\ f_b^{(s)} \end{bmatrix} \tag{1.79}$$

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

$$\begin{bmatrix} S_{b-1b-1}^{(s)} & S_{b-1b}^{(s)} \\ S_{bb-1}^{(s)} & S_{bb}^{(s)} \end{bmatrix} = \begin{bmatrix} K_{b-1b-1}^{(s)} - K_{b-1i}^{(s)} {K_{ii}^{(s)}}^{-1} K_{ib-1}^{(s)} & -K_{b-1i}^{(s)} {K_{ii}^{(s)}}^{-1} K_{ib}^{(s)} \\ -K_{bi}^{(s)} {K_{ii}^{(s)}}^{-1} K_{ib-1}^{(s)} & K_{bb}^{(s)} - K_{bi}^{(s)} {K_{ii}^{(s)}}^{-1} K_{ib}^{(s)} \end{bmatrix} \tag{1.80}$$

$$\begin{bmatrix} c_{b-1}^{(s)} \\ c_b^{(s)} \end{bmatrix} = \begin{bmatrix} f_{b-1}^{(s)} - K_{b-1i}^{(s)} {K_{ii}^{(s)}}^{-1} f_i^{(s)} \\ f_b^{(s)} - K_{bi}^{(s)} {K_{ii}^{(s)}}^{-1} f_i^{(s)} \end{bmatrix} \tag{1.81}$$

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

$$\begin{bmatrix} \cdots & \cdots & 0 & 0 \\ S_{b-1b-2}^{(s-1)} & S_{b-1b-1}^{(s-1)} + S_{b-1b-1}^{(s)} & S_{b-1b}^{(s)} & 0 \\ 0 & S_{bb-1}^{(s)} & S_{bb}^{(s)} + S_{bb}^{(s+1)} & S_{bb+1}^{(s+1)} \\ 0 & 0 & \cdots & \cdots \end{bmatrix} \quad , \quad \begin{bmatrix} \cdots \\ c_{b-1}^{(s-1)} + c_{b-1}^{(s)} \\ c_b^{(s)} + c_b^{(s+1)} \\ \cdots \end{bmatrix} \tag{1.82}$$

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

$$\begin{bmatrix} S_{b-1b-1}^{(-)} + S_{b-1b-1}^{(s)} & S_{b-1b}^{(s)} \\ S_{bb-1}^{(s)} & S_{bb}^{(s)} + S_{bb}^{(+)} \end{bmatrix} \begin{bmatrix} x_{b-1}^{(s)} \\ x_b^{(s)} \end{bmatrix} = \begin{bmatrix} c_{b-1}^{(-)} + c_{b-1}^{(s)} \\ c_b^{(s)} + c_b^{(+)} \end{bmatrix} \tag{1.83}$$

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

$$\begin{aligned}
S^{(-)}_{b-1b-1} &= S^{(s)}_{b-1b-1} - S^{(s)}_{b-1b}\left[S^{(s)}_{bb}+S^{(+)}_{bb}\right]^{-1} S^{(s)}_{bb-1} \\
S^{(+)}_{bb} &= S^{(s)}_{bb} - S^{(s)}_{bb-1}\left[S^{(-)}_{b-1b-1}+S^{(s)}_{b-1b-1}\right]^{-1} S^{(s)}_{b-1b} \\
c^{(-)}_{b-1} &= c^{(s)}_{b} - S^{(s)}_{bb-1}\left[S^{(-)}_{b-1b-1}+S^{(s)}_{b-1b-1}\right]^{-1}\left[c^{(-)}_{b-1}+c^{(s)}_{b-1}\right] \\
c^{(+)}_{b} &= c^{(s)}_{b-1} - S^{(s)}_{b-1b}\left[S^{(s)}_{bb}+S^{(+)}_{bb}\right]^{-1}\left[c^{(s)}_{b}+c^{(+)}_{b}\right]
\end{aligned} \tag{1.84}$$

Previous system suggest 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, since $\Omega^{(s)}$ is the only subdomain with right hand side non null, then $c^{(-)}_{b-1} = 0$ and $c^{(+)}_{b} = 0$ and the system is the exact condensation of local augmented problem

$$\begin{bmatrix} K^{(s)}_{ii} & K^{(s)}_{ib-1} & K^{(s)}_{ib} \\ K^{(s)}_{b-1i} & K^{(s)}_{b-1b-1}+S^{(-)}_{b-1b-1} & 0 \\ K^{(s)}_{bi} & 0 & K^{(s)}_{bb}+S^{(+)}_{bb} \end{bmatrix} \begin{bmatrix} x^{(s)}_{i} \\ x^{(s)}_{b-1} \\ x^{(s)}_{b} \end{bmatrix} = \begin{bmatrix} f^{(s)}_{i} \\ f^{(s)}_{b-1} \\ f^{(s)}_{b} \end{bmatrix} \tag{1.85}$$

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 1.85) converges in a number of iteration equal to the number of subdomains minus one.*

Proof. See Roux et al. [67]. □

With this theorem we know that in every subdomain the optimal augmentation term, in a one way split, 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 had been done [68], from which we can name the following:

Schur neighbour: In a physical context, the complete outer Schur complement matrix represent 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 neighbour Schur complement. In this case the cost of computation and the exchange of data are reduced to the neighboring subdomains only. Even though this is a good theoretical approximation, in practice it still has a very expensive cost. From this first approach defines the next two approximations, that try to mimic the behavior of the neighbour Schur complement, but with an sparse and not expensive approximation of it, that gives good convergence for the FETI-2LM method.

Lumped approximation: It has been shown in [67] that an approximation of 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 good results. This way to approximate, is very easy to implement since 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 later in the numerical results section. The goal is emulate the spectral density of the neighbour Schur complement matrix as close as possible.

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

$$\tilde{S}^{(s)} = K_{bb}^{(s)} - K_{bj}^{(s)} {K_{jj}^{(s)}}^{-1} K_{jb}^{(s)}, \qquad s = 1, 2, \forall j \in V_d \tag{1.86}$$

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

the set

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

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 its from this idea that the sparse approximation is build.

We start again with the 2 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]. Lets define the following subsets of indexes for the nodes of $\Omega^{(s)}, s = 1, 2$:

$$\begin{aligned}
V_{\Omega^{(s)}} &= \{ \text{ indexes of nodes inside the subdomain } \Omega^{(s)}\} \\
V_\Gamma &= \{ \text{ indexes of nodes inside the interface } \Gamma\} \\
V_p^j &= \{ \text{ indexes of the nodes in} V_\Gamma \text{ such that the minimum connectivity distance between each of these nodes and the node labelled } j \text{ is lower or equal than } p, p \in \mathbb{N}\} \\
V_{p,d}^j &= \{ \text{ indexes of the nodes in } V_{\Omega^{(s)}} \text{ such that the minimum connectivity distance between each of these nodes and the nodes in } V_p^j \text{ is lower or equal than } d, d \in \mathbb{N}\}
\end{aligned} \tag{1.87}$$

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

This approximation consists on defining an sparse augmentation matrix, in order to avoid the increase on the bandwidth of the local problem matrix. This augmentation matrix is build 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 7.

Algorithm 7 Sparse approximation of neighbour 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 build 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 neighbour 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

$$\begin{aligned} V_1^{13} &= \{7, 13, 19\} \\ V_{1,1}^{13} &= \{1, 2, 8, 14, 20, 26, 25\} \\ V_{1,2}^{13} &= \{1, 2, 8, 14, 20, 26, 25, 3, 9, 15, 21, 27, 33, 32, 31\} \end{aligned} \tag{1.88}$$

These subsets correspond to the overlapping layers represented Figure 1.7.

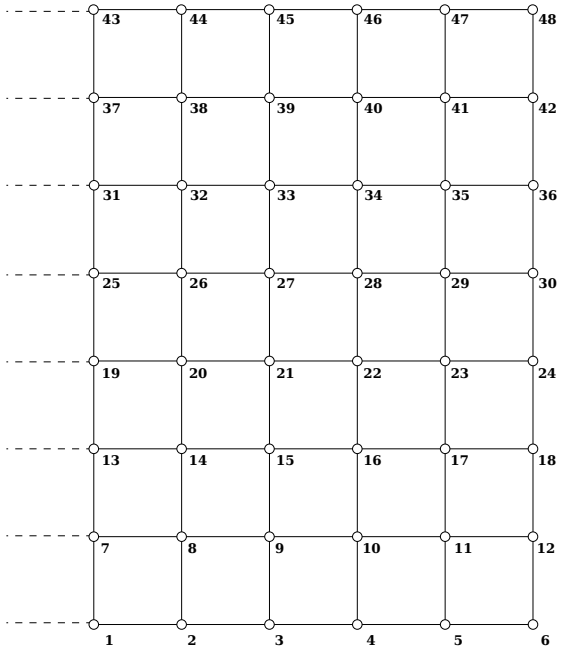

Figure 1.6 – Nodes numbering in subdomain $\Omega^{(s)}$.

With the previous computation of the augmentation matrix, the final algorithm is the one described theoretically in previous section Algorithm 6, but without any projection nor preconditioning, since the local problems are all well posed. The full orthogonalization process is kept as it is needed to avoid the loss of orthogonality produced when multiplying by the FETI operator.

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, since we need to add the computed $A^{(s)}$ matrix. The other big difference is in the computation by this new operator, this is done by changing the assembly of the residuals according to the definition in Equation 1.77.

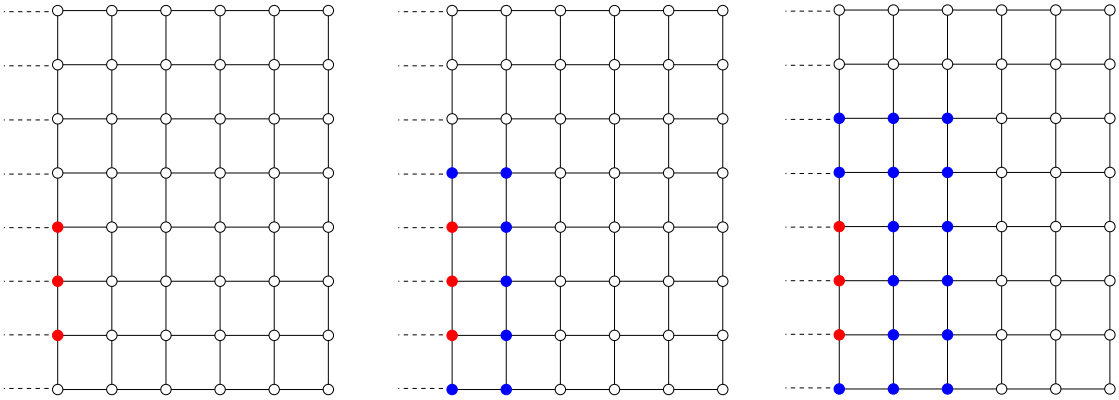

Figure 1.7 – Interface patch of size $p = 1$ (red) with one and two layers $d = 1, 2$ (blue).

1.3 New FETI as an 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 on 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 for the 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 basic scheme model of a rolling bearing, this kind of problems can be easily extended to other contact schemes were we want to compute the interactions between objects of different materials, that can be as different as steel and rubber. The heterogeneous problem with contact boundary conditions imposed with the penalty method is a type of problem that can make the discretization matrix of the model and consequently the FETI operator, very ill conditioned.

In this configuration a natural interface will be the one that divides the two materials, if this is the case, then the FETI-1LM method with the extended preconditioner will be appropriate to handle the heterogenities. However, this is not always the case when you do an automatic partitioning of the global mesh,

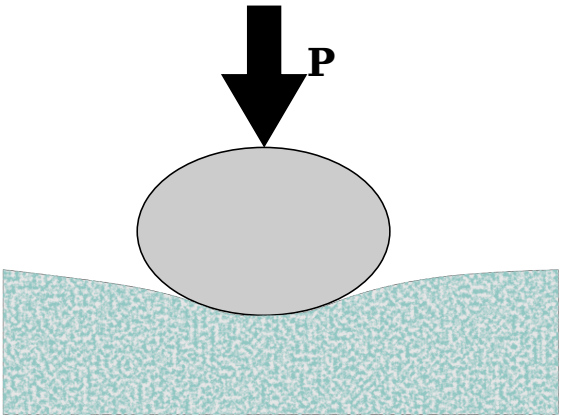

Figure 1.8 – A contact problem.

or if, for example, we use different discretizations that leads 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 surpass this 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 results, but we can try to get both of these good features if we choose properly which method to apply in every interface. If we manage to make the choice that better suits every local interface, then we will most likely have a new hybrid method with both good qualities.

This method we will call it Hybrid-FETI and it's formulation can be described mathematically as previous methods, but this time starting from a different basic configuration, as we need at least two different local interfaces. For simplicity reason the basic problem consist in three subdomains and two interfaces that are completely independent of one another and no crosspoints as shown in Figure 1.9. In any case the generalization is straightforward since the interfaces are always disconnected with FETI approaches.

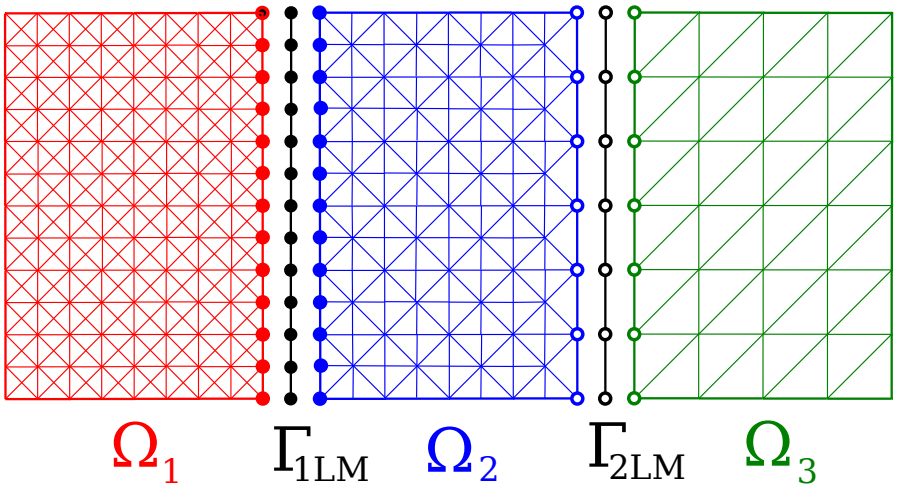

Figure 1.9 – Three subdomain divisions.

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

$$K^{(s)} = \begin{bmatrix} K_{ii}^{(s)} & K_{ib}^{(s)} \\ K_{bi}^{(s)} & K_{bb}^{(s)} \end{bmatrix}, \qquad f^{(s)} = \begin{bmatrix} f_i^{(s)} \\ f_b^{(s)} \end{bmatrix} \tag{1.89}$$

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

$$\begin{bmatrix} K_{ii}^{(1)} & 0 & 0 & K_{ib}^{(1)} \\ 0 & K_{ii}^{(2)} & 0 & K_{ib}^{(2)} \\ 0 & 0 & K_{ii}^{(3)} & K_{ib}^{(3)} \\ K_{bi}^{(1)} & K_{bi}^{(2)} & K_{bi}^{(3)} & K_{bb} \end{bmatrix} \begin{bmatrix} x_i^{(1)} \\ x_i^{(2)} \\ x_i^{(3)} \\ x_b \end{bmatrix} = \begin{bmatrix} f_i^{(1)} \\ f_i^{(2)} \\ f_i^{(3)} \\ f_b \end{bmatrix} \tag{1.90}$$

up to here we have the same as previous methods, but 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

$$\begin{aligned} K_{bi}^{(1)} &= \begin{bmatrix} K_{b_1 i}^{(1)} \\ 0 \end{bmatrix}, \qquad K_{bi}^{(2)} = \begin{bmatrix} K_{b_1 i}^{(2)} \\ K_{b_2 i}^{(2)} \end{bmatrix}, \qquad K_{bi}^{(3)} = \begin{bmatrix} 0 \\ K_{b_2 i}^{(3)} \end{bmatrix} \\ K_{ib}^{(1)} &= \begin{bmatrix} K_{ib_1}^{(1)} & 0 \end{bmatrix}, \quad K_{ib}^{(2)} = \begin{bmatrix} K_{ib_1}^{(2)} & K_{ib_2}^{(2)} \end{bmatrix}, \quad K_{ib}^{(3)} = \begin{bmatrix} 0 & K_{ib_2}^{(3)} \end{bmatrix} \end{aligned} \tag{1.91}$$

As for the interaction between the interfaces, we have

$$\begin{aligned} K_{bb} = K_{bb}^{(1)} + K_{bb}^{(2)} + K_{bb}^{(3)} &= \begin{bmatrix} K_{b_1 b_1}^{(1)} & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} K_{b_1 b_1}^{(2)} & 0 \\ 0 & K_{b_2 b_2}^{(2)} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & K_{b_2 b_2}^{(3)} \end{bmatrix} \\ f_b = f_b^{(1)} + f_b^{(2)} + f_b^{(3)} &= \begin{bmatrix} f_{b_1}^{(1)} \\ 0 \end{bmatrix} + \begin{bmatrix} f_{b_1}^{(2)} \\ f_{b_2}^{(2)} \end{bmatrix} + \begin{bmatrix} 0 \\ f_{b_2}^{(3)} \end{bmatrix} \end{aligned} \tag{1.92}$$

And the unknown displacements are

$$x_b = \begin{bmatrix} x_{b_1} \\ x_{b_2} \end{bmatrix} \tag{1.93}$$

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 to have

$$\begin{bmatrix} K_{ii}^{(1)} & K_{ib_1}^{(1)} \\ K_{b_1 i}^{(1)} & K_{b_1 b_1}^{(1)} \end{bmatrix} \begin{bmatrix} x_i^{(1)} \\ x_{b_1}^{(1)} \end{bmatrix} = \begin{bmatrix} f_i^{(1)} \\ f_{b_1}^{(1)} + \lambda_{b_1}^{(1)} \end{bmatrix} \tag{1.94}$$

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, an we add 2 extra at the interface in the "right" and the augmentation matrix, as in

the 2LM method, with this we have

$$\begin{bmatrix} K_{ii}^{(2)} & K_{ib_1}^{(2)} & K_{ib_2}^{(2)} \\ K_{b_1 i}^{(2)} & K_{b_1 b_1}^{(2)} & 0 \\ K_{b_2 i}^{(2)} & 0 & K_{b_2 b_2}^{(2)} + A_{b_2}^{(2)} \end{bmatrix} \begin{bmatrix} x_i^{(2)} \\ x_{b_1}^{(2)} \\ x_{b_2}^{(2)} \end{bmatrix} = \begin{bmatrix} f_i^{(2)} \\ f_{b_1}^{(2)} + \lambda_{b_1}^{(2)} \\ f_{b_2}^{(2)} + \lambda_{b_2}^{(2)} \end{bmatrix} \tag{1.95}$$

And for the third subdomain,

$$\begin{bmatrix} K_{ii}^{(3)} & K_{ib_2}^{(3)} \\ K_{b_2 i}^{(3)} & K_{b_2 b_2}^{(3)} + A_{b_2}^{(3)} \end{bmatrix} \begin{bmatrix} x_i^{(3)} \\ x_{b_2}^{(3)} \end{bmatrix} = \begin{bmatrix} f_i^{(3)} \\ f_{b_2}^{(3)} + \lambda_{b_2}^{(3)} \end{bmatrix} \tag{1.96}$$

Since 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

$$\lambda_{b1} = \lambda_{b1}^{(1)} = -\lambda_{b1}^{(2)} \tag{1.97}$$

Now, using the same condensation process as 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. Lets take the first subdomain, where we know from the first equation in 1.94 that

$$x_i^{(1)} = -K_{ii}^{(1)^{-1}} K_{ib_1}^{(1)} x_{b_1}^{(1)} + K_{ii}^{(1)^{-1}} f_i^{(1)} \tag{1.98}$$

introducing it in the second equation we have

$$\begin{aligned} \lambda_{b_1} &= K_{b_1 i}^{(1)} x_i^{(1)} + K_{b_1 b_1}^{(1)} x_{b_1}^{(1)} - b_{f_1}^{(1)} \\ &= -K_{b_1 i}^{(1)} K_{ii}^{(1)^{-1}} K_{ib_1}^{(1)} x_{b_1}^{(1)} + K_{b_1 b_1}^{(1)} x_{b_1}^{(1)} - f_{b_1}^{(1)} + K_{b_1 i}^{(1)} K_{ii}^{(1)^{-1}} f_i^{(1)} \\ &= S_{b_1 b_1}^{(1)} x_{b_1}^{(1)} - c_{b_1}^{(1)} \end{aligned} \tag{1.99}$$

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

$$x_{b_1}^{(1)} = F^{(1)} \lambda_{b_1} + F^{(1)} c_{b_1}^{(1)} \tag{1.100}$$

where

$$F^{(1)} := S^{(1)^{-1}}_{b_1 b_1} \tag{1.101}$$

is the inverse of the Schur complement.
For the second subdomain, we have from the first line in 1.95 that

$$x^{(2)}_i = -K^{(2)^{-1}}_{ii} K^{(2)}_{ib_1} x^{(2)}_{b_1} - K^{(2)^{-1}}_{ii} K^{(2)}_{ib_2} x^{(2)}_{b_2} + K^{(2)^{-1}}_{ii} f^{(2)}_i \tag{1.102}$$

replacing in the second and third equation, we have

$$\begin{aligned}
\begin{bmatrix} \lambda_{b_1} \\ \lambda^{(2)}_{b_2} \end{bmatrix} &= \begin{bmatrix} K^{(2)}_{b_1 i} x^{(2)}_i + K^{(2)}_{b_1 b_1} x^{(2)}_{b_1} - b^{(2)}_{b_1} \\ K^{(2)}_{b_2 i} x^{(2)}_i + (K^{(2)}_{b_2 b_2} + A^{(2)}_{b_2}) x^{(2)}_{b_2} - b^{(2)}_{b_2} \end{bmatrix} \\
&= \begin{bmatrix} K^{(2)}_{b_1 b_1} - K^{(2)}_{b_1 i} K^{(2)^{-1}}_{ii} K^{(2)}_{ib_1} & -K^{(2)}_{b_1 i} K^{(2)^{-1}}_{ii} K^{(2)}_{ib_2} \\ -K^{(2)}_{b_2 i} K^{(2)^{-1}}_{ii} K^{(2)}_{ib_1} & K^{(2)}_{b_2 b_2} - K^{(2)}_{b_2 i} K^{(2)^{-1}}_{ii} K^{(2)}_{ib_2} + A^{(2)}_{b_2} \end{bmatrix} \begin{bmatrix} x^{(2)}_{b_1} \\ x^{(2)}_{b_2} \end{bmatrix} + \begin{bmatrix} -f^{(2)}_{b_1} + K^{(2)}_{b_1 i} K^{(2)^{-1}}_{ii} f^{(2)}_i \\ -f^{(2)}_{b_2} + K^{(2)}_{b_1 i} K^{(2)^{-1}}_{ii} f^{(2)}_i \end{bmatrix} \\
&= \begin{bmatrix} S^{(2)}_{b_1 b_1} & S^{(2)}_{b_1 b_2} \\ S^{(2)}_{b_2 b_1} & S^{(2)}_{b_2 b_2} + A^{(2)}_{b_2} \end{bmatrix} \begin{bmatrix} x^{(2)}_{b_1} \\ x^{(2)}_{b_2} \end{bmatrix} - \begin{bmatrix} c^{(2)}_{b_1} \\ c^{(2)}_{b_2} \end{bmatrix}
\end{aligned} \tag{1.103}$$

where again the S terms denotes the corresponding Schur complements of the subdomain 2 in the interfaces 1LM and 2LM and $c^{(2)}_{b_1} := f^{(2)}_{b_1} - K^{(2)}_{b_1 i} K^{(2)^{-1}}_{ii} f^{(2)}_i$, $c^{(2)}_{b_2} = f^{(2)}_{b_2} - K^{(2)}_{b_1 i} K^{(2)^{-1}}_{ii} f^{(2)}_i$. So both the displacements of both local interfaces are

$$\begin{bmatrix} x^{(2)}_{b_1} \\ x^{(2)}_{b_2} \end{bmatrix} = F^{(2)} \begin{bmatrix} \lambda_{b_1} \\ \lambda^{(2)}_{b_2} \end{bmatrix} + F^{(2)} \begin{bmatrix} c^{(2)}_{b_1} \\ c^{(2)}_{b_2} \end{bmatrix} \tag{1.104}$$

here we have

$$F^{(2)} := \begin{bmatrix} S^{(2)}_{b_1 b_1} & S^{(2)}_{b_1 b_2} \\ S^{(2)}_{b_2 b_1} & S^{(2)}_{b_2 b_2} + A^{(2)}_{b_2} \end{bmatrix}^{-1} \tag{1.105}$$

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

$$F^{(2)} := \begin{bmatrix} F^{(2)}_{b_1 b_1} & F^{(2)}_{b_1 b_2} \\ F^{(2)}_{b_2 b_1} & F^{(2)}_{b_2 b_2} \end{bmatrix} \tag{1.106}$$

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

$$\begin{bmatrix} x_{b_1}^{(2)} \\ x_{b_2}^{(2)} \end{bmatrix} = \begin{bmatrix} F_{b_1b_1}^{(2)}\lambda_{b_1} + F_{b_1b_2}^{(2)}\lambda_{b_2}^{(2)} + F_{b_1b_1}^{(2)}c_{b_1}^{(2)} + F_{b_1b_2}^{(2)}c_{b_2}^{(2)} \\ F_{b_2b_1}^{(2)}\lambda_{b_1} + F_{b_2b_2}^{(2)}\lambda_{b_2}^{(2)} + F_{b_2b_1}^{(2)}c_{b_1}^{(2)} + F_{b_2b_2}^{(2)}c_{b_2}^{(2)} \end{bmatrix} \tag{1.107}$$

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

$$x_i^{(3)} = -K_{ii}^{(3)^{-1}} K_{ib_2}^{(3)} x_{b_2}^{(3)} + K_{ii}^{(3)^{-1}} f_i^{(3)} \tag{1.108}$$

and then replace it in the second equation of 1.96

$$\begin{aligned} \lambda_{b_2}^{(3)} &= K_{b_2i}^{(3)} x_i^{(3)} + (K_{b_2b_2}^{(3)} + A_{b_2}^{(3)}) x_{b_2}^{(3)} - f_{b_2}^{(3)} \\ &= -K_{b_2i}^{(3)} K_{ii}^{(3)^{-1}} K_{ib_2}^{(3)} x_{b_2}^{(3)} + (K_{b_2b_2}^{(3)} + A_{b_2}^{(3)}) x_{b_2}^{(3)} - f_{b_2}^{(3)} + K_{b_2i}^{(3)} K_{ii}^{(3)^{-1}} f_i^{(3)} \\ &= (S_{b_2b_2}^{(3)} + A_{b_2}^{(3)}) x_{b_2}^{(3)} - c_{b_2}^{(3)} \end{aligned} \tag{1.109}$$

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

$$x_{b_2}^{(3)} = F^{(3)} \lambda_{b_2}^{(3)} + F^{(3)} c_{b_2}^{(3)} \tag{1.110}$$

where

$$F^{(3)} := (S_{b_2b_2}^{(3)} + A_{b_2}^{(3)})^{-1} \tag{1.111}$$

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

$$x_{b_1}^{(1)} - x_{b_1}^{(2)} = 0 \tag{1.112}$$

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

in 1.69 we need to add another constraint that gives us a residual of the form

$$\begin{aligned} x_{b_2}^{(2)} - x_{b_2}^{(3)} &= 0 \\ \lambda_{b_2}^{(2)} + \lambda_{b_2}^{(3)} - A_{b_2}^{(2)} x_{b_2}^{(2)} - A_{b_2}^{(3)} x_{b_2}^{(3)} &= 0 \end{aligned} \tag{1.113}$$

this mixed form is recombined as in the 2LM method so together with Equation 1.112 we obtain the conditions needed to impose continuity across complete the interface

$$\begin{aligned} x_{b_1}^{(1)} - x_{b_1}^{(2)} &= 0 \\ \lambda_{b_2}^{(2)} + \lambda_{b_2}^{(3)} - (A_{b_2}^{(2)} + A_{b_2}^{(3)}) x_{b_2}^{(2)} &= 0 \\ \lambda_{b_2}^{(2)} + \lambda_{b_2}^{(3)} - (A_{b_2}^{(2)} + A_{b_2}^{(3)}) x_{b_2}^{(3)} &= 0 \end{aligned} \tag{1.114}$$

With this equations plus the relations between the lagrange multipliers and the different displacement, coming from the solution of local problems 1.100,1.107,1.110, we can do the condensation of previous problems on to the interface for the hybrid method

$$\begin{bmatrix} F^{(1)} + F_{b_1 b_1}^{(2)} & -F_{b_1 b_2}^{(2)} & 0 \\ \left(A_{b_2}^{(2)} + A_{b_2}^{(3)}\right) F_{b_2 b_1}^{(2)} & I - \left(A_{b_2}^{(2)} + A_{b_2}^{(3)}\right) F_{b_2 b_2}^{(2)} & I \\ 0 & I & I - \left(A_{b_2}^{(2)} + A_{b_2}^{(3)}\right) F^{(3)} \end{bmatrix} \begin{bmatrix} \lambda_{b_1} \\ \lambda_{b_2}^{(2)} \\ \lambda_{b_2}^{(3)} \end{bmatrix}$$
$$= \begin{bmatrix} -F^{(1)} c_{b_1}^{(1)} + F_{b_1 b_1}^{(2)} c_{b_1}^{(2)} + F_{b_1 b_2}^{(2)} c_{b_2}^{(2)} \\ \left(A_{b_2}^{(2)} + A_{b_2}^{(3)}\right)\left(F_{b_2 b_1}^{(2)} c_{b_1}^{(2)} + F_{b_2 b_2}^{(2)} c_{b_1}^{(2)}\right) \\ \left(A_{b_2}^{(2)} + A_{b_2}^{(3)}\right) F^{(3)} c_{b_2}^{(3)} \end{bmatrix} \tag{1.115}$$

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 neighbouring interface edges so that each one is treated depending on the chosen method.

We split the total interface nodes into this interface edges, that correspond to a collection of connecting interface nodes, denoted Γ^j as we did for the 2LM case.

We also add a unique marker to this edges in order to know a priori which one correspond 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. Crosspoints in particular are edges of one node, were we can arbitrarily marked them 1LM or 2LM, we will see the impact this choice in the numerical experiments.

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

$$B^{(s)}_{\Gamma^j} v^{(s)} = v^{(s)}_b|_{\Gamma^j} \tag{1.116}$$

again $v^{(s)} = \begin{bmatrix} v^{(s)}_i \\ v^{(s)}_b \end{bmatrix}$ with the notation for interior and interfaces nodes being i and b respectively.

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

$$\begin{aligned}
\left(K^{(s)} + \sum_{\Gamma^j_{2lm} \subseteq \Omega^{(s)}} B^{(s)^T}_{\Gamma^j_{2lm}} A^{(s)}_{\Gamma^j_{2lm}} B^{(s)}_{\Gamma^j_{2lm}} \right) x^{(s)} &= f^{(s)} - \sum_{\Gamma^l \subseteq \Omega^{(s)}} B^{(s)^T}_{\Gamma^l} \lambda^{(s)}_{\Gamma^l}, \qquad s = 1, \dots, N_s \\
B^{(s)}_{\Gamma^j} x^{(s)} - x_b|_{\Gamma^j} &= 0 \qquad \forall \Gamma^j, \quad \Omega^{(s)} \supseteq \Gamma^j \\
\sum_{\Omega^{(q)} \supseteq \Gamma^j_{2lm}} \left(\lambda^{(q)}_{\Gamma^j_{2lm}} - A^{(q)}_{\Gamma^j_{2lm}} x_b|_{\Gamma^j_{2lm}} \right) &= 0 \qquad \forall \Gamma^j_{2lm}
\end{aligned} \tag{1.117}$$

where $A^{(s)}_{\Gamma^j_{2lm}}$ is the augmentation matrix in subdomain $\Omega(s)$ for the interface Γ^j_{2lm}, the other terms are defined analogous as for the case in FETI-2LM. Lets also define the total augmentation matrix for a subdomain that contains an edge *"2lm"*, where if not the case, its value is zero

$$A^{(s)}_{2lm} := \sum_{\Gamma^j_{2lm} \subseteq \Omega^{(s)}} B^{(s)^T}_{\Gamma^j_{2lm}} A^{(s)}_{\Gamma^j_{2lm}} B^{(s)}_{\Gamma^j_{2lm}} \tag{1.118}$$

With this equation we can do the condensation and obtain the problem to solve in the interface. Since, in general, two methods are used, then equations in the interface will also need to be separated, finally we have the interface problems

$$\begin{aligned} &\sum_{\Omega^{(q)} \supseteq \Gamma^j_{2lm}} \left(\lambda^{(q)}_{\Gamma^j_{2lm}} - A^{(q)}_{\Gamma^j_{2lm}} B^{(s)}_{\Gamma^j_{2lm}} \left(K^{(s)} + A^{(s)}_{2lm} \right)^{-1} \sum_{\Gamma^l \subseteq \Omega^{(s)}} B^{(s)^T}_{\Gamma^l} \lambda^{(s)}_{\Gamma^l} \right) \\ &= - \sum_{\Omega^{(q)} \supseteq \Gamma^j_{2lm}} A^{(q)}_{\Gamma^j_{2lm}} B^{(s)}_{\Gamma^j_{2lm}} \left(K^{(s)} + A^{(s)}_{2lm} \right)^{-1} f^{(s)} \qquad \forall \Gamma^j_{2lm}, \forall \Omega^{(s)} \supseteq \Gamma^j_{2lm} \end{aligned} \tag{1.119}$$

$$\begin{aligned} &B^{(s)}_{\Gamma^j_{1lm}} \left(K^{(s)} + A^{(s)}_{2lm} \right)^{+} \left(\lambda_{\Gamma^j_{1lm}} + \sum_{\Gamma^l_{2lm} \subseteq \Omega^{(s)}} B^{(s)^T}_{\Gamma^l_{2lm}} \lambda^{(s)}_{\Gamma^l_{2lm}} \right) + B^{(s)}_{\Gamma^j_{1lm}} R^{(s)} \alpha^{(s)} \\ &+ B^{(q)}_{\Gamma^j_{1lm}} \left(K^{(q)} + A^{(q)}_{2lm} \right)^{+} \left(\lambda_{\Gamma^j_{1lm}} - \sum_{\Gamma^l_{2lm} \subseteq \Omega^{(q)}} B^{(q)^T}_{\Gamma^l_{2lm}} \lambda^{(q)}_{\Gamma^l_{2lm}} \right) - B^{(q)}_{\Gamma^j_{1lm}} R^{(q)} \alpha^{(q)} = \\ &- B^{(s)}_{\Gamma^j_{1lm}} \left(K^{(s)} + A^{(s)}_{2lm} \right)^{+} f^{(s)} + B^{(q)}_{\Gamma^j_{1lm}} \left(K^{(q)} + A^{(q)}_{2lm} \right)^{+} f^{(q)} \qquad \forall \Gamma^j_{1lm} = \Omega^{(s)} \cap \Omega^{(q)} \end{aligned} \tag{1.120}$$

where the terms $R^{(s)}$ (or q) is a kernel generator for the matrix $K^{(s)}$ with some coefficients $\alpha^{(s)}$ such that the admissibility condition 1.36 for this "floating domain" is fulfilled. We need then to use the same projection strategy explained in section 1.1.2, in order to treat this subdomains. In the case that the 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, so we use ORTHODIR 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 for the mesh size, we need to find a preconditioner as in FETI-1LM. Up to this day no optimal preconditioner as been found for the FETI-2LM, but this method has been tested and shown numerically to be scalable for the mesh size on his own [35]. No extra computation needs to be done for the interface edges where the Robin condition is imposed. On the other hand, for those where one lagrange multiplier is used, a preconditioner is mandatory in order 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 from a mechanical point of view, we know it consist on solving a local problem whose boundary condition is the jump in the displacements fields along this 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 this local contributions.

The local problem to solve, then comes from solving the following Dirichlet problem in $\Omega^{(s)}$

$$\begin{bmatrix} K^{(s)}_{ii} & K^{(s)}_{ib} \\ 0 & I \end{bmatrix} \begin{bmatrix} w^{(s)}_i \\ w^{(s)}_b \end{bmatrix} = \begin{bmatrix} 0 \\ \{w^{(s)}_b\} \end{bmatrix} \tag{1.121}$$

the subscript i is for interior nodes and b for the interface, so the jumps are imposed along the total local interfaces.

We can use this same 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.

We compute the multiplication of a vector w restricted to the interface edges Γ^j_{1lm} by our preconditioner as

$$D^{-1}_{\Gamma^j_{1lm}} w|_{\Gamma^j_{1lm}} = \left(B^{(s)}_{\Gamma^j_{1lm}} S^{(s)}_{bb} B^{(s)^T}_{\Gamma^j_{1lm}} + B^{(q)}_{\Gamma^j_{1lm}} S^{(q)}_{bb} B^{(q)^T}_{\Gamma^j_{1lm}} \right) w|_{\Gamma^j_{1lm}}, \quad \forall \Gamma^j_{1lm} : \Gamma^j_{1lm} = \partial\Omega^{(s)} \cap \partial\Omega^{(q)} \tag{1.122}$$

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

The multiplication by this operator are made 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 neighbour subdomain. We can write the previous preconditioner as a global matrix of the form

$$D^{-1} w = \left(\sum_{\Gamma^j_{1lm} \subseteq \Gamma} D^{-1}_{\Gamma^j_{1lm}} \right) w \tag{1.123}$$

with w any vector defined in the interface Γ.

One problem that we encounter in this preconditioner, happens when in some subdomain the local interface is composed of both 1LM and 2LM interface 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 were used, the upper one corresponding to the usual Dirichlet preconditioner, and the bottom one represent an option to build a new preconditioner.

The advantage of this boundary conditions is that now we obtain the exact local inverse of the Hybrid-FETI operator. The preconditioner is then changed to solve this problem in the subdomains with shared edges. It involves the computation of a different Schur complement, this time with the augmentation matrix that allows to mimic the behaviour of the local interface marked as 2LM and therefore of the whole subdomain.

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

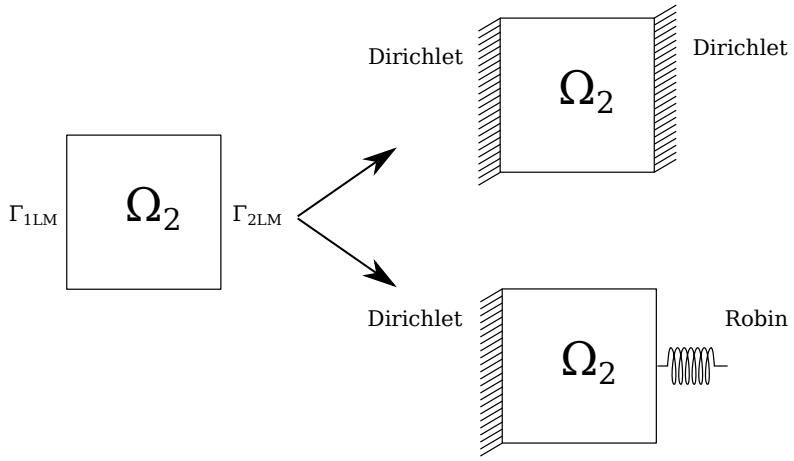

Figure 1.10 – Boundary conditions for both preconditioners.

subdomain with at least one edge marked as 1LM

$$\begin{bmatrix} K_{ii}^{(s)} & K_{ib_{1lm}}^{(s)} & K_{ib_{2lm}}^{(s)} \\ 0 & I & 0 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} w_i^{(s)} \\ w_{b_{1lm}}^{(s)} \\ w_{b_{2lm}}^{(s)} \end{bmatrix} = \begin{bmatrix} 0 \\ \{w_{b_{1lm}}^{(s)}\} \\ 0 \end{bmatrix} \tag{1.124}$$

where b_{1lm}, b_{2lm} are subscript for the union of 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

$$\begin{aligned} S_{bb}^{(s)} &= \begin{bmatrix} K_{b_{1lm}b_{1lm}}^{(s)} & 0 \\ 0 & K_{b_{2lm}b_{2lm}}^{(s)} + A_{b_{2lm}}^{(s)} \end{bmatrix} - \begin{bmatrix} K_{b_{1lm}i}^{(s)} & K_{b_{2lm}i}^{(s)} \end{bmatrix} K_{ii}^{(s)^{-1}} \begin{bmatrix} K_{ib_{1lm}}^{(s)} \\ K_{ib_{2lm}}^{(s)} \end{bmatrix} \\ &= \begin{bmatrix} \tilde{S}_{b_{1lm}b_{1lm}}^{(s)} & S_{b_{1lm}b_{2lm}}^{(s)} \\ S_{b_{2lm}b_{1lm}}^{(s)} & S_{b_{2lm}b_{2lm}}^{(s)} + A_{b_{2lm}}^{(s)} \end{bmatrix} \end{aligned} \tag{1.125}$$

The structure of this matrix is the same as usual and correspond to the local inverse of the condensed interface problem (as seen in 1.105).

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 previous, but remembering that the Schur complement now includes the augmentation matrices (whenever they exist)

$$R^{-1}_{\Gamma^j_{1lm}} w|_{\Gamma^j_{1lm}} = \left(B^{(s)}_{\Gamma^j_{1lm}} \tilde{S}^{(s)}_{bb} B^{(s)^T}_{\Gamma^j_{1lm}} + B^{(q)}_{\Gamma^j_{1lm}} \tilde{S}^{(q)}_{bb} B^{(q)^T}_{\Gamma^j_{1lm}} \right) w|_{\Gamma^j_{1lm}}, \quad \forall \Gamma^j_{1lm} : \Gamma^j_{1lm} = \partial\Omega^{(s)} \cap \partial\Omega^{(q)} \tag{1.126}$$

the definition of every term, other than the Schur complement, are 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

$$R^{-1} w = \left(\sum_{\Gamma^j_{1lm} \subseteq \Gamma} R^{-1}_{\Gamma^j_{1lm}} \right) w \tag{1.127}$$

1.3.4 Implementation

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

Depending on the method this computation of the residuals involves the resolution of the corresponding local Neumann and Robin problems in order to compute $x^{(i)}_b$. In general for our method there will be at least one subdomain where the local problem solved is the one that comes from imposing both conditions at the same time.

Lets recall how the gradients are computed for each method. To do so, lets take as 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 then

$$g = D^{-1}(F\lambda - d) = D^{-1}(x_b^{(2)} - x_b^{(1)}) \tag{1.128}$$

where $x_b^{(s)}$, $s = 1, 2$ are the displacement of the interface nodes of 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.

If we look at previous computation from an implementation point of view, we have that for each process that computes each subdomain, the gradient is obtained as

$$g = \begin{bmatrix} g_b^{(1)} \\ g_b^{(2)} \end{bmatrix} = D^{-1} \begin{bmatrix} x_b^{(2)} - x_b^{(1)} \\ x_b^{(1)} - x_b^{(2)} \end{bmatrix} \tag{1.129}$$

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 1.70, meaning that for each process we have

$$g = \begin{bmatrix} g_b^{(1)} \\ g_b^{(2)} \end{bmatrix} = \begin{bmatrix} \lambda_b^{(1)} + \lambda_b^{(2)} - (A_b^{(1)} + A_b^{(2)})x_b^{(2)} \\ \lambda_b^{(2)} + \lambda_b^{(1)} - (A_b^{(1)} + A_b^{(2)})x_b^{(1)} \end{bmatrix} \tag{1.130}$$

the values of $g_b^{(1)}$ and $g_b^{(2)}$ are different, since 2 independent lagrange multipliers are used for this method, but we can see that the computation structure is the same as the FETI-1LM method. The difference here lies in the exchanges needed to compute both gradients, in the case of the 2LM method more than one exchange is needed.

The computation of the gradients is the base to the computation by the F operator, needed in the use of the iterative method. For the case of an arbitrary number of subdomains, the previous computations is extended by defining a division of the total interface into interface edges as previously done. So, the multiplication of a vector v defined in the interface by the FETI operator is done by:

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

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

The first part is similar for the FETI-1LM, FETI-2LM and our method, only changing the existence of the augmentation term when some 2LM marked edge is present. Is in the assembly part of the gradient where we need to do the mix of both method.

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

$$g = \begin{bmatrix} g_{b_1}^{(1)} \\ g_{b_1}^{(2)} \\ g_{b_2}^{(2)} \\ g_{b_2}^{(3)} \end{bmatrix} = D^{-1} \begin{bmatrix} x_{b_1}^{(2)} - x_{b_1}^{(1)} \\ x_{b_1}^{(1)} - x_{b_1}^{(2)} \\ x_{b_2}^{(3)} - x_{b_2}^{(2)} \\ x_{b_2}^{(2)} - x_{b_2}^{(3)} \end{bmatrix} = \left(D_{\Gamma^{b_1}}^{-1} + D_{\Gamma^{b_2}}^{-1}\right) \begin{bmatrix} x_{b_1}^{(2)} - x_{b_1}^{(1)} \\ x_{b_1}^{(1)} - x_{b_1}^{(2)} \\ x_{b_2}^{(3)} - x_{b_2}^{(2)} \\ x_{b_2}^{(2)} - x_{b_2}^{(3)} \end{bmatrix} \tag{1.131}$$

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

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

$$g = \begin{bmatrix} g_{b_1}^{(1)} \\ g_{b_1}^{(2)} \\ g_{b_2}^{(2)} \\ g_{b_2}^{(3)} \end{bmatrix} = \begin{bmatrix} \lambda_{b_1}^{(1)} + \lambda_{b_1}^{(2)} - (A_{b_1}^{(1)} + A_{b_1}^{(2)}) x_{b_1}^{(2)} \\ \lambda_{b_1}^{(2)} + \lambda_{b_1}^{(1)} - (A_{b_1}^{(2)} + A_{b_1}^{(1)}) x_{b_1}^{(1)} \\ \lambda_{b_2}^{(2)} + \lambda_{b_2}^{(3)} - (A_{b_2}^{(2)} + A_{b_2}^{(3)}) x_{b_2}^{(2)} \\ \lambda_{b_2}^{(3)} + \lambda_{b_2}^{(2)} - (A_{b_2}^{(3)} + A_{b_2}^{(2)}) x_{b_2}^{(1)} \end{bmatrix} \tag{1.132}$$

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

assembling

$$g = \begin{bmatrix} g_{b_1}^{(1)} \\ g_{b_1}^{(2)} \\ g_{b_2}^{(2)} \\ g_{b_2}^{(3)} \end{bmatrix} = D^{-1}_{\Gamma^{b_1}_{1lm}} \begin{bmatrix} x_{b_1}^{(2)} - x_{b_1}^{(1)} \\ x_{b_1}^{(1)} - x_{b_1}^{(2)} \\ \lambda_{b_2}^{(2)} + \lambda_{b_2}^{(3)} - (A_{b_2}^{(1)} + A_{b_2}^{(2)}) x_{b_2}^{(2)} \\ \lambda_{b_2}^{(3)} + \lambda_{b_2}^{(2)} - (A_{b_2}^{(2)} + A_{b_2}^{(1)}) x_{b_2}^{(1)} \end{bmatrix} \tag{1.133}$$

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 residuals allows us to obtain in an analog way 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 due to the fact that it shares blocks of the also non symmetric 2LM operator, so we use the iterative ORTHODIR method with left preconditioner and full reconjugation, for the same reasons as we use it in the unsymmetric and 2LM methods.

The FETI-1LM method for unsymmetric matrices described in Algorithm 6 is the base of our implementation. Both of them share theoretical resemblance, changing the definition of our operator F and also modifying D to D_{1lm} because when using the preconditioner, 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 for the multiplication by our 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 next section we will test numerically our method to show the advantages of it in some kind of 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 null Dirichlet conditions will be imposed. Given $f \in L^2(\Omega)$, the problem is

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

$$\begin{cases} -\nu\Delta u &= f \quad in \quad \Omega \\ u &= 0 \quad on \quad \partial\Omega_D \\ \dfrac{\partial u}{\partial n} &= 0 \quad on \quad \partial\Omega \setminus \Omega_D \end{cases} \tag{1.134}$$

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

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*

$$\nu \int_\Omega \nabla u \cdot \nabla v dx = \int_\Omega f v, \quad \forall v \in V \tag{1.135}$$

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

$$Kx = f \tag{1.136}$$

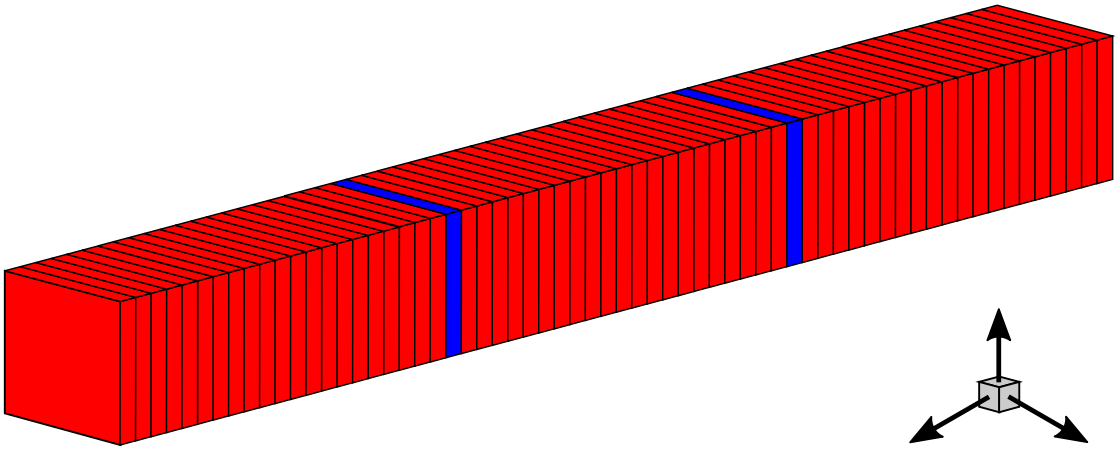

Figure 1.11 – Two material bar.

1.4.1 Two material bar

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

The problem will consist on 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

$$\begin{aligned} \nu_1 &= 10^5, \text{ "blue" blocks} \\ \nu_2 &= 10^0, \text{ otherwise} \end{aligned} \tag{1.137}$$

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

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

$$\frac{\|Kx^p - f\|_2}{\|f\|_2} < 10^{-4} \tag{1.138}$$

but we also use a second criterion, that is, the relative norm of the jump of the solution between subdomains [36].

$$\frac{\|x^p - x^{p-1}\|_2}{\|x^p\|_2} < 10^{-4} \tag{1.139}$$

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 due to the fact that the small number of interfaces where the preconditioner is different is not high 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 the most part 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

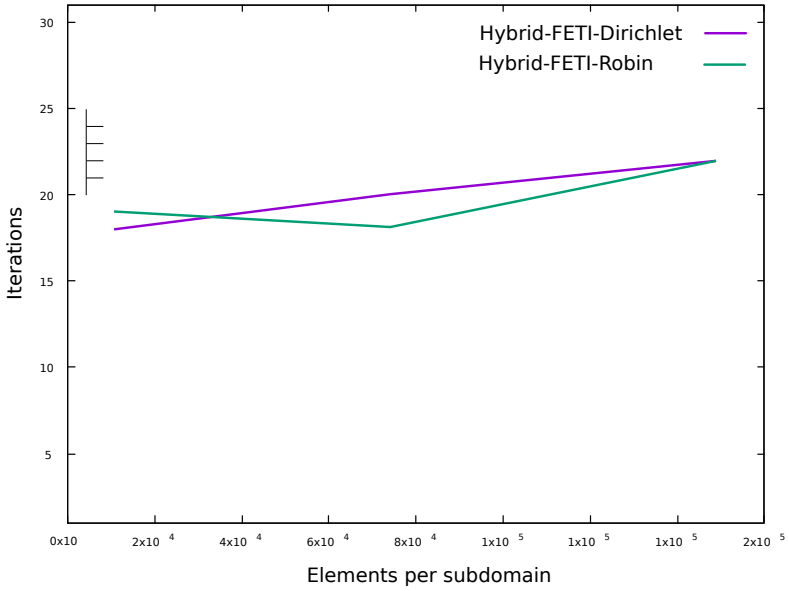

Figure 1.12 – Hybrid-FETI Iterations versus Elements number.

keep this property even in the harder problems.

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 are hold in the Hybrid case, in particular the property of having 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 due to the fact that this method is build from a combination of two numerically scalable methods.

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 marked as 2LM and the rest as 1LM. We will now test this choice against a different one, where two extra interfaces are marked as 2LM, as seen in Figure 1.13. The idea is to "cover" the problematic

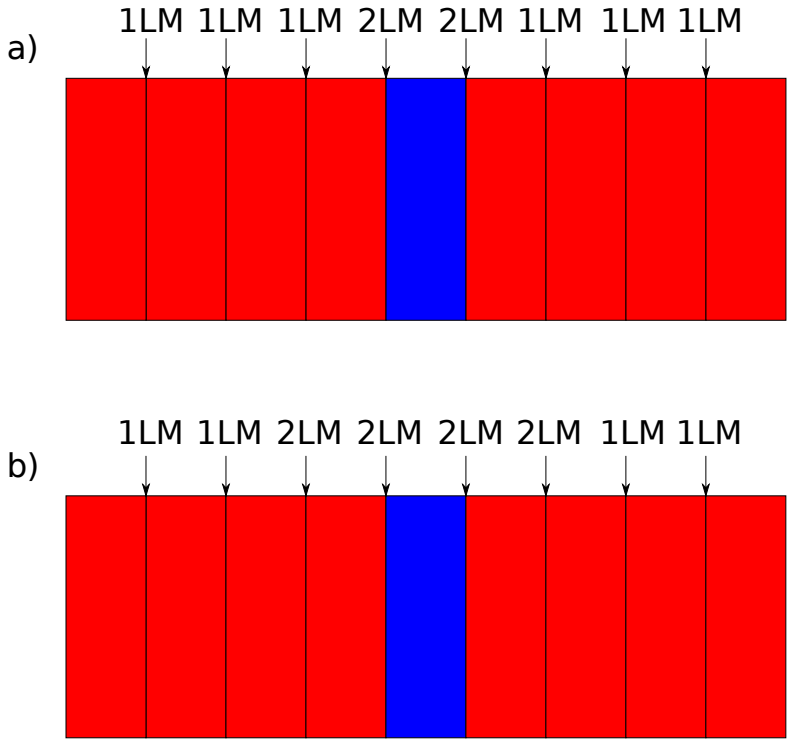

Figure 1.13 – a) Regular interface marking. b) Extra covering marking.

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

For the rest of this basic tests, no difference will be done between preconditioners as they are minimal.

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

In order to achieve a better performance, in the following for the Hybrid-FETI case we will mark as 2LM the interface that connects different materials and all the rest will be treated as 1LM.

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)

With the defaults parameters for the Hybrid method, now we can perform a comparative against the two base method. 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 against 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 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

$$\begin{aligned} \nu_1 &= 10^5, \text{ center block} \\ \nu_2 &= 10^0, \text{ otherwise} \end{aligned} \tag{1.140}$$

With this 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 non symmetric version of this method because the problem solved when working with contact models (one of our main objectives) is non symmetric, and in this case the performance is comparable to the one of Hybrid-FETI. As for the FETI-2LM, the very poor performance is explained by the type of divisions used, in this particular one, since the information is transmitted neighbour by neighbour, in order for it to cross the whole structure, we need at least 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 clearly outperforms that of FETI-2LM, meaning that we have ameliorated in terms of speed, 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 against both base ones. 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 know a priori, this condition is modeled as constraint of a minimization problem. For these reasons, these problems may lead to stiff non-linear systems of equations.

Several simplification 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 to not 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

$$\begin{aligned} -div(\sigma(\mathbf{u})) &= \mathbf{f}, \text{ in } \Omega \\ \mathbf{u} &= \mathbf{0} \text{ in } \partial\Omega_D \\ \sigma(\mathbf{u}) \cdot \mathbf{n} &= 0 \text{ in } \partial\Omega \setminus \partial\Omega_D \end{aligned} \tag{1.141}$$

where

$$\begin{aligned} \sigma_{ij}(\mathbf{u}) &= 2\mu\varepsilon_{ij}(\mathbf{u}) + \lambda\delta_{ij}div(\mathbf{u}) \\ \varepsilon_{ij}(\mathbf{u}) &= \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \\ \mu &= \frac{E}{2(1+\nu)} \\ \lambda &= \frac{E\nu}{(1+\nu)(1-2\nu)} \end{aligned} \tag{1.142}$$

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, variation of this values in a generated structure will create the heterogeneities along or across the interfaces.

This model correspond to the static linear elasticity, however 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.

Lets 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

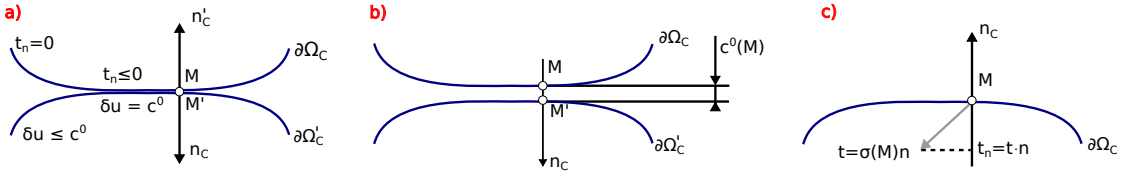

Figure 1.14 – a) Contact problem. b) Initial gap. c) Contact pressure.

contact, unknown a priori, n_C is the outward normal to $\partial\Omega_C$ at a point M , $\partial\Omega'_C$ also the potential area of contact with $\partial\Omega_C$ but for another linear elastic 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 denotes the displacement field (in one direction), the discontinuity of u in the normal direction at point $M \in \partial\Omega_C$ can be written as

$$\begin{aligned}\delta u(M) &= u(M)\cdot n_C + u(M')\cdot n'_C = u(M)\cdot n_C - u(M')\cdot n_C \\ &= (u(M) - u(M'))\cdot n_C\end{aligned} \tag{1.143}$$

we write the non-penetration condition in the next form

$$\delta u(M) - c(M)^0 \leq 0 \tag{1.144}$$

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 $Signorini - Fichera$ type of conditions, 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.

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

$$\begin{aligned} t_n &\leq 0 \\ \delta u(M) - c(M)^0 &\leq 0 \\ t_n \cdot (\delta u(M) - c(M)^0) &= 0 \end{aligned} \tag{1.145}$$

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

$$V = \{\mathbf{v} \in (H^1(\Omega))^2 : \mathbf{v}|_{\partial\Omega_D} = \mathbf{0}\} \tag{1.146}$$

and

$$U = \{\mathbf{v} \in V : \mathbf{v} \cdot n - c(M)^0 \leq 0 \text{ on } \partial\Omega_C\} \tag{1.147}$$

We define then the bilinear and linear forms

$$\begin{aligned} a(\mathbf{u}, \mathbf{v}) &:= 2 \int_\Omega \mu\varepsilon(\mathbf{u}) : \varepsilon(\mathbf{v}) dx + \int_\Omega \lambda(\nabla \cdot \mathbf{u})(\nabla \cdot \mathbf{v}) dx \\ b(\mathbf{v}) &:= \int_\Omega \langle \mathbf{f}, \mathbf{v} \rangle dx \end{aligned} \tag{1.148}$$

where

$$\langle \mathbf{f}, \mathbf{v} \rangle = \sum_{i=1}^{2} f_i v_i \tag{1.149}$$

and finally

$$\mathbf{F}(\mathbf{v}) = \frac{1}{2} a(\mathbf{v}, \mathbf{v}) - b(\mathbf{v}) \tag{1.150}$$

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

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

$$a(\mathbf{u}, \mathbf{v} - \mathbf{u}) \geq f(\mathbf{v} - \mathbf{u}), \quad \forall \mathbf{v} \in U \tag{1.151}$$

or

$$\mathbf{F}(\mathbf{u}) \leq \mathbf{F}(\mathbf{v}) \quad \forall \mathbf{v} \in U \tag{1.152}$$

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 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 form 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], [26], [23]. They are all based in 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, [24]. So from now on, we will only consider a FETI-like with the penalty method to solve the contact problem, since there is a better treatment of the contact constraint with no extra variables at the cost of ill-conditioned problems, that we will try to overcome.

For a contact problem, after the solver convergence we obtain a final active area of contact, with this we can "lock" the substructures in this final configuration, 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, that also treats with the large displacements non-linearities of 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, this 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 condi-

tion.

When iterating to solve the non linearities, at the beginning the penalty parameter is not necessarily very high and non-physical penetration may occur, but when approaching to the final computed loads we must increase the value of it, in order to nullify this penetration and have a correct imposition of the contact condition. Is this treatment, the one that produces the very ill-conditioning of the local stiffness matrices, when the contact area is active.

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 is provided with augmentation or regularization matrices, added to this local stiffness problems, thanks to the generalized Robin condition imposed in the interfaces. This matrices are usually taken as approximations of the Schur complement of the neighbour subdomain, meaning that even if the stiffness of one neighbour subdomain is orders of magnitude larger than the other (this happens when the penalization parameter is acting in contact) it will be taken into account by its neighbour.

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

$$\begin{bmatrix} I & I-(A_b^{(1)}+A_b^{(2)})(S_{bb}^{(2)}+A_b^{(2)})^{-1} \\ I-(A_b^{(2)}+A_b^{(1)})(S_{bb}^{(1)}+A_b^{(1)})^{-1} & I \end{bmatrix} \tag{1.153}$$

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 under the computed external loads produced by the interaction of both structures, including the penalization in the local stiffness matrices that share an interface with active contact. In this static problem we can compare the two standard FETI method against the new Hybrid solver.

In the first problem to solve, we have a block in top of a semi-circle, see Figure 1.15. The top block is the stiffer body and the bottom one is assumed to

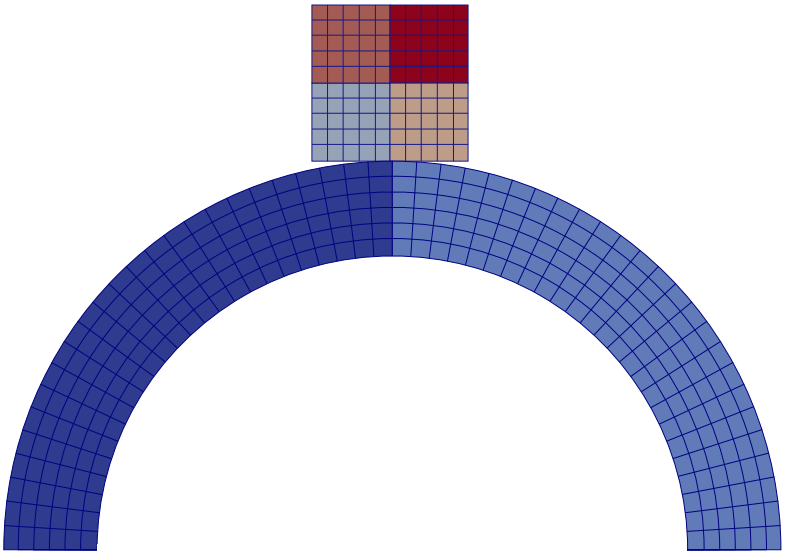

Figure 1.15 – First example, initial configuration and subdomain division.

have large displacements. The values for their respective Young's modulus and Poisson ratio are

$$\begin{aligned} E_{top} &= 10000, \quad \nu_{top} = 0.4 \\ E_{bottom} &= 6, \quad \nu_{bottom} = 0.3 \end{aligned} \tag{1.154}$$

The total 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 applied to the base of the bottom structure, and in order to give an initial stability, the two top corners of the block are also fixed.

Then, in Figure 1.16 we can see the solution, after applying the contact solver. This allows to compute the final active contact zone, that is 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 to recognize each subdomain as part of one or the other structure, which

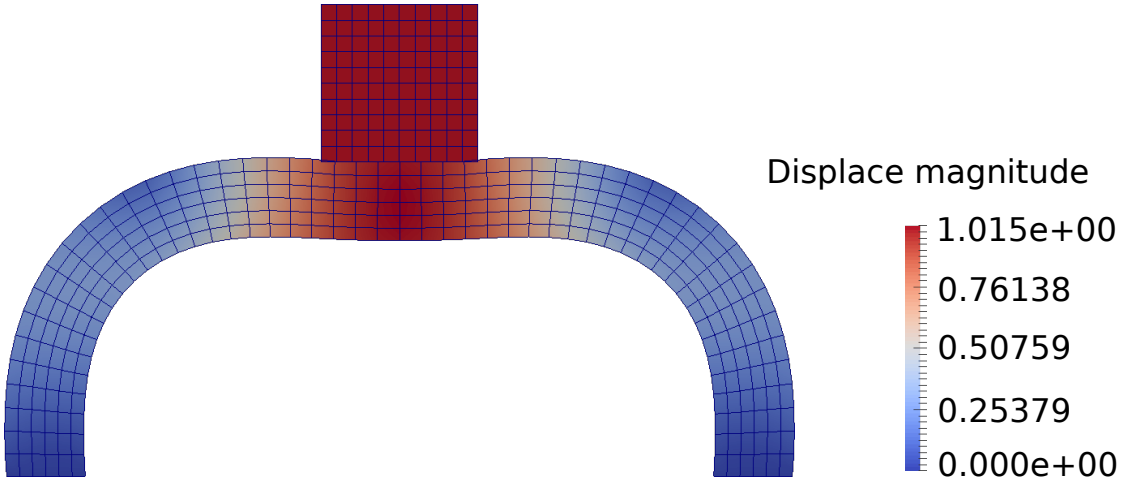

Figure 1.16 – First example, final configuration (solution).

at the same time automatizes the process of marking the different interfaces in the following way

$$\begin{cases} Mark\ \Gamma^{(ij)}\ as\ 1LM & if\ \Omega^{(i)}, \Omega^{(j)}\ are\ in\ the\ same\ body \\ Mark\ \Gamma^{(ij)}\ as\ 2LM & Otherwise \end{cases} \tag{1.155}$$

With this process, plus the Robin or Dirichlet preconditioner, diminishing the stopping criterions to errors less than 10^{-6} and a patch size of 3 and path depth of 5 for the augmentation matrices in the 2LM interfaces, we can launch our test 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 an 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 and the bottom and second 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 completely 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

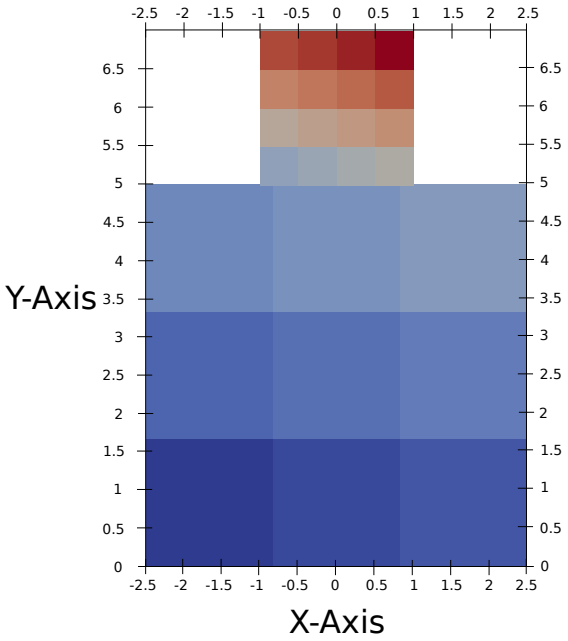

Figure 1.17 – Second example, initial configuration.

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 then 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 also we improved the convergence ratio in this case, 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 originally 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

In general the results given 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. This advantages are clearer in the context of solving the ill-conditioned problems arising from the simplest contact models. In this cases the advantages in terms of easier implementation and improved convergence shown in the numerical results gives us a very good start in the search of general better algorithms.

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

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 this challenging problems, the Simultaneous-FETI or S-FETI was developed, first in [17] for a two subdomain case, then generalized for arbitrary configurations in [39]. The main characteristic of the S-FETI method it's the generation from the preconditioning of more than one search direction for the Conjugate Gradient method, leading to a more efficient and in general more robust method.

With this new method, 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 using the sparse properties of the block of vectors from which we construct the search directions, in order 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 this reconstruction steps will add computations that from a local point of view will be more expensive, but since this cost is small

compared to the time spend in the communication of the different processes, the total time used can be as good as the original method. The simplest of the implementations of this idea will not be enough to achieve the good performance needed, but it will show the pertinence of the storage used and the precision achieved in relation to the method with no special storage. Later we will show some optimizations based on the parallel properties of the code that improve the computation time, making a more efficient algorithm.

This chapter start showing the basics of preconditioning in FETI, then we show how the ideas in the preconditioning steps lead to the formulation of the basic S-FETI method for two subdomains. We continue by presenting the formal general method, including its implementation and cost. Then we show in detail the way of sorting 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 to finally show some numerical results to compare the different algorithms.

2.1 Introduction and preliminarities

The development of the S-FETI method begins in [17] 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 will start by revisiting the Dirichlet preconditioner at its basic, for a two subdomain case. The notation in this case will be analogous to the previous chapter.

The objective of a preconditioner for the PCPG Algorithm 5 used in FETI, is to build in every iteration p a correction to the Lagrange multipliers given an interface compatibility error, i.e. a jump denoted as g. (In the case of the elasticity problem this is a jump in the displacement, but depending on the

problem this may change. Also we drop the notation as an iterative vector g^p for simplicity).

Let us consider the two subdomain case as in Figure 1.2 from section 1.1.1. Where the associated unknown are

$$x^{(s)} = \begin{bmatrix} x_i^{(s)} \\ x_b^{(s)} \end{bmatrix}, \qquad s = 1, 2 \tag{2.1}$$

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

$$g = B^{(1)} x_b^{(1)} + B^{(2)} x_b^{(2)} \tag{2.2}$$

with B the signed boolean operator that defined this continuity condition as in Equation 1.33. We can then build a continuous interface approximate solution by averaging the already computed $x^{(s)}$

$$\begin{aligned} \hat{x}_b^{(1)} &= \frac{1}{2}\left(x_b^{(1)} - B^{(1)^T} B^{(2)} x_b^{(2)} \right) \\ \hat{x}_b^{(2)} &= \frac{1}{2}\left(x_b^{(2)} - B^{(2)^T} B^{(1)} x_b^{(1)} \right) \end{aligned} \tag{2.3}$$

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

$$\begin{aligned} \delta x_b^{(1)} &= \hat{x}_b^{(1)} - x_b^{(1)} = -\frac{1}{2} B^{(1)^T} g \\ \delta x_b^{(2)} &= \hat{x}_b^{(2)} - x_b^{(2)} = -\frac{1}{2} B^{(2)^T} g \end{aligned} \tag{2.4}$$

If we modify the internal nodes to satisfy the internal equilibrium (the first equation in the local problem 1.29) then we have the correction

$$\delta x_i^{(s)} = -K_{ii}^{(s)^{-1}} K_{ib}^{(s)} \delta x_b^{(s)} \tag{2.5}$$

Multiplying by the local matrix, the interface response corrections on the inter-

face Γ are computed by

$$\delta f_b^{(s)} = S_{bb}^{(s)} \delta x_b^{(s)} \tag{2.6}$$

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

$$z = -\frac{1}{2}\left(B^{(1)} \delta f_b^{(1)} + B^{(2)} \delta f_b^{(2)}\right) \tag{2.7}$$

This correspond to a new search direction in the CG algorithm. If we look at this definition, using 2.4 and 2.6, we can also write it as

$$z = \left(\frac{1}{4}\sum_{s=1}^{2} B^{(s)} S_{bb}^{(s)} B^{(s)^T}\right) g \tag{2.8}$$

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 a totally symmetric splitting of a simple square, then the preconditioner is the exact inverse of the operator FETI, if not the case, we only have locally the exact inverse of this operator.

The difference between this Dirichlet preconditioner and his basic non-scaled form lies in what its defined as a *Consistent preconditioner*. In the next part, we will give the 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 general subdomain division. For an arbitrary partition into $N_s \geq 2$, we define

$$\Gamma^{(s)} = \partial\Omega^{(s)} \backslash \partial\Omega \tag{2.9}$$

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

$$\Gamma = \underset{s}{\cup}\Gamma^{(s)} \tag{2.10}$$

Next we consider the signed boolean matrix $B^{(s)}$ as the matrix 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 that represents the same global node on the neighbour subdomain.

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

$$\Gamma^j := \Gamma^{(sq)} = \partial\Omega^{(s)} \cap \partial\Omega^{(q)}, \qquad \forall s,q = 1,\dots,N_s \tag{2.11}$$

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

$$B^{(s)} = \begin{bmatrix} B^{(s)}_{\Gamma^1} \\ B^{(s)}_{\Gamma^2} \\ \vdots \\ B^{(s)}_{\Gamma^{n^{(s)}}} \end{bmatrix} \tag{2.12}$$

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

$$\mathbf{B} = \left[B^{(1)} \cdots B^{(N_s)}\right] \tag{2.13}$$

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

$$m_j = |neighbours| + 1 \tag{2.14}$$

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 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 2.12 and knowing that there are nodes shared by more than

two subdomains we can see that

$$B_{\Gamma^j}^{(s)^T} B_{\Gamma^j}^{(s)} = (m_j - 1)I \tag{2.15}$$

then for each subdomain we have that

$$\begin{aligned} B^{(s)^T} B^{(s)} &= \begin{bmatrix} B_{\Gamma^1}^{(s)^T} & \cdots & B_{\Gamma^{n^{(s)}}}^{(s)^T} \end{bmatrix} \begin{bmatrix} B_{\Gamma^1}^{(s)} \\ \cdots \\ B_{\Gamma^{n^{(s)}}}^{(s)} \end{bmatrix} \\ &= B_{\Gamma^1}^{(s)^T} B_{\Gamma^1}^{(s)} + \cdots + B_{\Gamma^{n^{(s)}}}^{(s)^T} B_{\Gamma^{n^{(s)}}}^{(s)} \\ &= (m_1 - 1)I + \cdots + (m_{n^{(s)}} - 1)I \\ &= diag(m_j - 1) \end{aligned} \tag{2.16}$$

Finally we can write

$$B^{(s)^T} B^{(s)} + I = diag(m_j) \tag{2.17}$$

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 analyse 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

$$g = \sum_{s=1}^{N_s} B^{(s)} x_b^{(s)} \tag{2.18}$$

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 $B^{(s)^T} z = B^{(s)^T} [\tilde{F}^{-1} g]$ in order 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

$$z = \tilde{F}^{-1} g = \sum_{s=1}^{N_s} B^{(s)} \left(S_{bb}^{(s)} \text{ or } K_{bb}^{(s)} \right) B^{(s)^T} g \tag{2.19}$$

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

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

$$\delta x_b^{(s)} = B^{(s)^T} g \tag{2.20}$$

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

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

$$\delta f_b^{(s)} = \left(S_{bb}^{(s)} \right) \delta x_b^{(s)} \tag{2.21}$$

The difference between the Dirichlet preconditioner based on this operator and the Lumped one lies in this step where we replace previous computation by

$$\delta f_b^{(s)} = \left(K_{bb}^{(s)} \right) \delta x_b^{(s)} \tag{2.22}$$

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

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

$$z = \sum_{s=1}^{N_s} B^{(s)} \delta f_b^{(s)} \tag{2.23}$$

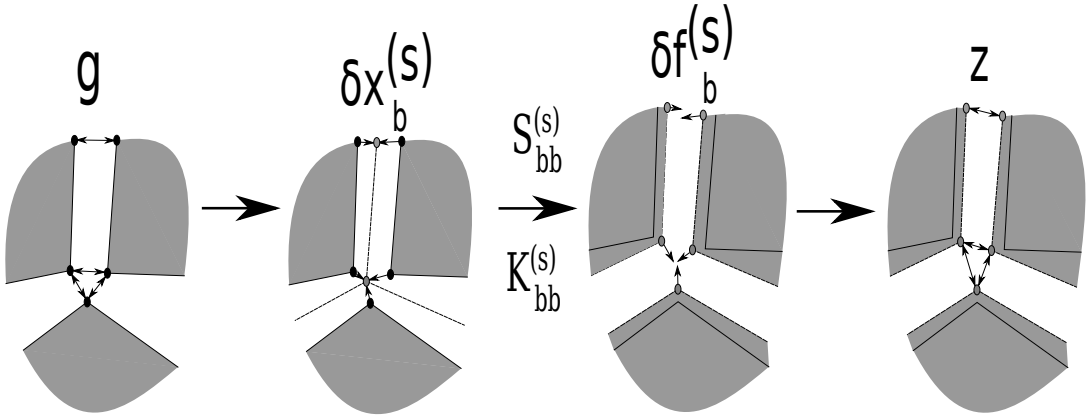

Figure 2.1 – Preconditioner construction

We can now define what is a *Consistent preconditioner* and it correspond to a preconditioner build following the previous three steps, but where the approximate solution corrections $\delta x_b^{(s)}$ fulfill the compatibility condition, meaning that are continuous across the interface and also the Lagrange multipliers corrections z are chosen based in 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 a common value (e.g. the average) to every node independently on the interface, will also be valid for the general case, i.e. considering crosspoint.

Graphically, the three steps construction can be seen in Figure 2.1

From Figure 2.1 (a) and (b) shows how the increments $\delta x_b^{(s)}$ are chosen so that the corrected interface approximate solution $\hat{x}_b^{(s)}$ satisfy the interface compatibility. Following Equation 2.21 we compute the nodal interface responses required to maintain the increment $\delta x_b^{(s)}$. This mapping $\delta f_b^{(s)}$ do not satisfy the interface equilibrium unless the corrected solutions $\hat{x}_b^{(s)}$ are the exact final solution. 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 be constructed in such a way that as the $\delta f_b^{(s)}$ approach equilibrium, $B^{(s)^T} z$ is exactly $\delta f_b^{(s)}$.

Lets consider the global interface response correction $\widehat{\delta f}_b$ as the vector defined in the interface, such that

$$\delta f_b^{(s)} = \widehat{\delta f}_b|_{\Gamma^{(s)}} \tag{2.24}$$

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

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

$$\sum_{s=1}^{N_s} B^{(s)} \hat{x}_b^{(s)} = \sum_{s=1}^{N_s} B^{(s)} \left(x_b^{(s)} + \delta x_b^{(s)} \right) = 0 \tag{2.25}$$

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

$$\widehat{\delta f}_b \in Im(\mathbf{B}) \tag{2.26}$$

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

$$B^{(s)^T} z = \delta f_b^{(s)} \tag{2.27}$$

The Equation 2.26 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

$$\forall i \in \Gamma, \qquad \sum_{s=1}^{N_s} \delta f_i^{(s)} = 0 \tag{2.28}$$

At a subdomain level the condition 2.26 can also be written

$$\delta f_b^{(s)} - B^{(s)^T} \sum_{\substack{r=1 \\ r \neq s}}^{N_s} B^{(r)} \delta f_b^{(r)} = 0 \tag{2.29}$$

With this definition, it can be expected that preconditioners that fulfills it are better than the ones who does not, this was shown in [61]. In this sense we will explain the modifications done to the basic Dirichlet preconditioner in order to make it a consistent one, leading to a new structure that will be generalized in order to have a preconditioner also suited for heterogeneous problems (across

the interface).

Remark:

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

$$\sum_{s=1}^{N_s} B^{(s)} \left(x_b^{(s)} + \delta x_b^{(s)} \right) = g + \sum_{s=1}^{N_s} B^{(s)} B^{(s)^T} g \neq 0 \tag{2.30}$$

which violates the compatibility condition for the solution correction.

Furthermore, if we take the construction

$$z = \sum_{s=1}^{N_s} B^{(s)} \delta f_b^{(s)} \tag{2.31}$$

and we assume that Equation 2.29 holds, then we have

$$\begin{aligned} B^{(s)^T} z &= B^{(s)^T} \sum_{r=1}^{N_s} B^{(r)} \delta f_b^{(r)} \\ &= B^{(s)^T} B^{(s)} \delta f_b^{(s)} + B^{(s)^T} \sum_{\substack{r=1 \\ r \neq s}}^{N_s} B^{(r)} \delta f_b^{(r)} \\ &= \left(B^{(s)^T} B^{(s)} + I \right) \delta f_b^{(s)} \end{aligned} \tag{2.32}$$

and using Equation 2.17

$$B^{(s)^T} z = diag(m_j) \delta f_b^{(s)} \neq \delta f_b^{(s)} \tag{2.33}$$

2 The natural self-equilibrium of the Lagrange multipliers can be expressed replacing $\delta f_b^{(s)}$ by $B^{(s)^T} z$ in Equation 2.29

$$B^{(s)^T} z - B^{(s)^T} \sum_{\substack{r=1 \\ r \neq s}}^{N_s} B^{(r)} B^{(r)^T} z = 0 \tag{2.34}$$

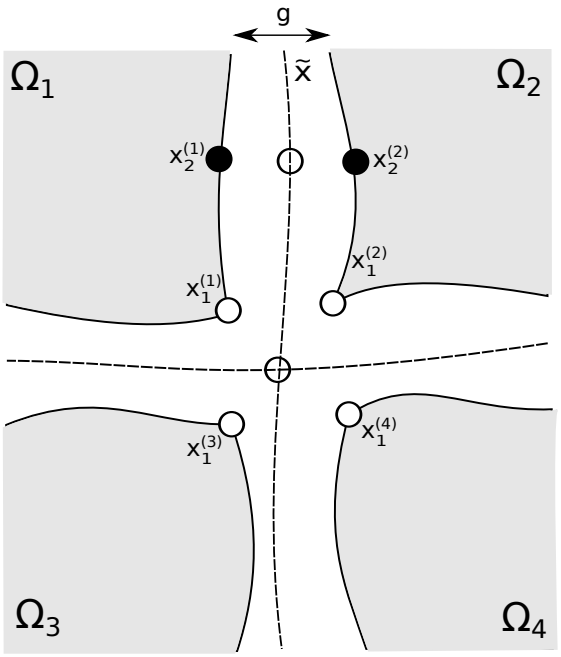

Figure 2.2 – 4 Subdomain problem

from where we have

$$B^{(s)^T} = B^{(s)^T} \sum_{\substack{r=1 \\ r \neq s}}^{N_s} B^{(r)} B^{(r)^T} \tag{2.35}$$

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 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 build 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

$$\begin{aligned} \hat{x}_1^{(1)} &= \hat{x}_1^{(2)} = \hat{x}_1^{(3)} = \hat{x}_1^{(4)} = \frac{x_1^{(1)} + x_1^{(2)} + x_1^{(3)} + x_1^{(4)}}{4} \\ \hat{x}_2^{(1)} &= \hat{x}_2^{(2)} = \frac{x_2^{(1)} + x_2^{(2)}}{2} \end{aligned} \tag{2.36}$$

which implies that the following corrections are consistent

$$\begin{aligned} \delta x_1^{(1)} &= \hat{x}_1^{(1)} - x_1^{(1)} \\ &= \frac{\left(x_1^{(2)} - x_1^{(1)}\right) + \left(x_1^{(3)} - x_1^{(1)}\right) + \left(x_1^{(4)} - x_1^{(1)}\right)}{4} \\ \delta x_1^{(2)} &= \frac{\left(x_1^{(1)} - x_1^{(2)}\right) + \left(x_1^{(3)} - x_1^{(2)}\right) + \left(x_1^{(4)} - x_1^{(2)}\right)}{4} \\ &\vdots \\ \delta x_2^{(1)} &= \frac{\left(x_2^{(2)} - x_2^{(1)}\right)}{2} \\ \delta x_2^{(2)} &= \frac{\left(x_2^{(1)} - x_2^{(2)}\right)}{2} \end{aligned} \tag{2.37}$$

If the interface nodal responses are computed as in Equation 2.21, 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

$$z_7 = \frac{-\delta f_2^{(1)} + \delta f_2^{(2)}}{2} \tag{2.38}$$

The minus sign in here comes from the fact that this vectors have opposite directions, because 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 same averaging to keep a symmetric preconditioner. Doing the analogous averaging for the crosspoint

of multiplicity 4, we have the corrections

$$\begin{aligned} z_1 &= \frac{-\delta f_1^{(1)} + \delta f_1^{(2)}}{4}, \qquad z_2 = \frac{-\delta f_1^{(2)} + \delta f_1^{(3)}}{4} \\ z_3 &= \frac{-\delta f_1^{(3)} + \delta f_1^{(4)}}{4}, \qquad z_4 = \frac{-\delta f_1^{(1)} + \delta f_1^{(4)}}{4} \\ z_5 &= \frac{-\delta f_1^{(1)} + \delta f_1^{(3)}}{4}, \qquad z_6 = \frac{-\delta f_1^{(2)} + \delta f_1^{(4)}}{4} \end{aligned} \tag{2.39}$$

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

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

$$\sum_{s=1}^{4} \delta f_1^{(s)} = 0 \text{ and } \sum_{s=1}^{2} \delta f_2^{(s)} = 0 \tag{2.40}$$

then

$$\begin{aligned} B^{(1)^T} z &= \begin{bmatrix} -z_1 - z_4 - z_5 \\ -z_7 \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{4}\left(\delta f_1^{(1)} - \delta f_1^{(2)}\right) + \frac{1}{4}\left(\delta f_1^{(1)} - \delta f_1^{(4)}\right) + \frac{1}{4}\left(\delta f_1^{(1)} - \delta f_1^{(3)}\right) \\ \frac{\delta f_1^{(1)} - \delta f_1^{(1)}}{2} \end{bmatrix} \\ &= \begin{bmatrix} \delta f_1^{(1)} \\ \delta f_2^{(1)} \end{bmatrix} \end{aligned} \tag{2.41}$$

And the same happens in analogous way with the other subdomains, so for the 4 subdomain case the averaging process shows the construction of a consistent preconditioner (Dirichlet or Lumped).

Lets 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

$$\hat{x}_{\Gamma^j}^{(s)} = \sum_{r:\Gamma^i \subseteq \{\Gamma \cap \Gamma^{(r)}\}} \frac{1}{m_i} x_{\Gamma^i}^{(r)} \tag{2.42}$$

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

$$\begin{aligned}\delta x_{\Gamma^j}^{(s)} &= \hat{x}_{\Gamma^j}^{(s)} - x_{\Gamma^j}^{(s)} \\ &= \sum_{\substack{r:\Gamma^i \subseteq \{\Gamma \cap \Gamma^{(r)}\} \\ r \neq s}} \frac{1}{m_i}\left(x_{\Gamma^i}^{(r)} - x_{\Gamma^i}^{(s)}\right)\end{aligned} \tag{2.43}$$

and extended to the total subdomain interface we have

$$\begin{aligned}\delta x_b^{(s)} &= -B^{(s)^T} diag\left(\frac{1}{m_j}\right) \sum_{r=1}^{N_s} B^{(r)} x_b^{(r)} \\ &= -B^{(s)^T} E^{(s)} g\end{aligned} \tag{2.44}$$

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 this corrections $\delta x_b^{(s)}$, the same as before we simply use Equation 2.21, as for the consistent Lagrange multiplier corrections, we generalize the averaging showed in 2.38 and 2.39 to obtain

$$z = \tilde{F}^{-1} g = diag\left(\frac{1}{m_i}\right) \sum_{r=1}^{N_s} B^{(r)} \delta f_b^{(r)} \tag{2.45}$$

Again, lets check the consistency of this correction. We assume the interface equilibrium condition 2.26 and using also 2.17 we have

$$\begin{aligned}B^{(s)^T} z &= diag\left(\frac{1}{m_i}\right) \sum_{r=1}^{N_s} B^{(s)^T} B^{(r)} \delta f_b^{(r)} \\ &= diag\left(\frac{1}{m_i}\right) B^{(s)^T} B^{(s)} \delta f_b^{(s)} + diag\left(\frac{1}{m_i}\right) \sum_{\substack{r=1 \\ r \neq s}}^{N_s} B^{(s)^T} B^{(r)} \delta f_b^{(r)} \\ &= diag\left(\frac{1}{m_i}\right) (diag\,(m_i) - I)\, \delta f_b^{(s)} + diag\left(\frac{1}{m_i}\right) \delta f_b^{(s)} \\ &= \delta f_b^{(s)}\end{aligned} \tag{2.46}$$

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

$$D^{-1} = \sum_s E^{(s)} B^{(s)} S_{bb}^{(s)} B^{(s)^T} E^{(s)} \tag{2.47}$$

$$L^{-1} = \sum_s E^{(s)} B^{(s)} K_{bb}^{(s)} B^{(s)^T} E^{(s)} \tag{2.48}$$

Consistent preconditioner for heterogeneous problems

The previous scaling works in homogeneous problems when the stiffness matrices of neighbouring 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 this 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

$$\begin{aligned} \hat{x}_1^{(1)} &= \hat{x}_1^{(2)} = \hat{x}_1^{(3)} = \hat{x}_1^{(4)} \\ &= \beta_1^{(1)} \hat{x}_1^{(1)} + \beta_1^{(2)} \hat{x}_1^{(2)} + \beta_1^{(3)} \hat{x}_1^{(3)} + \beta_1^{(4)} \hat{x}_1^{(4)} \\ \hat{x}_2^{(1)} &= \hat{x}_2^{(2)} \\ &= \beta_2^{(1)} \hat{x}_2^{(1)} + \beta_2^{(2)} \hat{x}_2^{(2)} \end{aligned} \tag{2.49}$$

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

$$\begin{aligned} \beta_1^{(1)} + \beta_1^{(2)} + \beta_1^{(3)} + \beta_1^{(4)} &= 1 \\ \beta_2^{(1)} + \beta_2^{(2)} &= 1 \end{aligned} \tag{2.50}$$

with this, the corrections vanish when the solution before weighting is already

compatible. In this case, the consistent corrections are

$$\begin{aligned}
\delta x_1^{(1)} &= \hat{x}_1^{(1)} - x_1^{(1)} \\
&= \beta_1^{(2)}\left(x_1^{(2)} - x_1^{(1)}\right) + \beta_1^{(3)}\left(x_1^{(3)} - x_1^{(1)}\right) + \beta_1^{(4)}\left(x_1^{(4)} - x_1^{(1)}\right) \\
&\vdots \\
\delta x_2^{(1)} &= \beta_2^{(2)}\left(x_2^{(2)} - x_2^{(1)}\right) \\
\delta x_2^{(2)} &= \beta_2^{(1)}\left(x_2^{(1)} - x_2^{(2)}\right)
\end{aligned} \tag{2.51}$$

the interface responses are again computed using Equation 2.21, so the corrections for the Lagrange multipliers are build using the same weighting

$$\begin{aligned}
z_1 &= -\beta_1^{(2)}\delta f_1^{(1)} + \beta_1^{(1)}\delta f_1^{(2)}, \qquad z_2 = -\beta_1^{(3)}\delta f_1^{(2)} + \beta_1^{(2)}\delta f_1^{(3)} \\
z_3 &= -\beta_1^{(4)}\delta f_1^{(3)} + \beta_1^{(3)}\delta f_1^{(4)}, \qquad z_4 = -\beta_1^{(4)}\delta f_1^{(1)} + \beta_1^{(1)}\delta f_1^{(4)} \\
z_5 &= -\beta_1^{(3)}\delta f_1^{(1)} + \beta_1^{(1)}\delta f_1^{(3)}, \qquad z_6 = -\beta_1^{(4)}\delta f_1^{(2)} + \beta_1^{(2)}\delta f_1^{(4)} \\
z_7 &= -\beta_2^{(2)}\delta f_2^{(1)} + \beta_1^{(2)}\delta f_2^{(2)}
\end{aligned} \tag{2.52}$$

Previous correction are in fact consistent, lets consider that the condition 2.26 is fulfilled, then for the subdomain 1 we have
if

$$\sum_{s=1}^{4} \delta f_1^{(s)} = 0 \text{ and } \sum_{s=1}^{2} \delta f_2^{(s)} = 0 \tag{2.53}$$

then

$$\begin{aligned}
B^{(1)^T} z &= \begin{bmatrix} -z_1 - z_4 - z_5 \\ -z_7 \end{bmatrix} \\
&= \begin{bmatrix} \left(\beta_1^{(2)} + \beta_1^{(4)} + \beta_1^{(3)}\right)\delta f_1^{(1)} - \beta_1^{(1)}\left(\delta f_1^{(2)} + \delta f_1^{(3)} + \delta f_1^{(4)}\right) \\ \beta_2^{(2)}\delta f_2^{(1)} - (1-\beta_2^{(2)})\delta f_2^{(2)} \end{bmatrix} \\
&= \begin{bmatrix} \delta f_1^{(1)} \\ \delta f_2^{(1)} \end{bmatrix}
\end{aligned} \tag{2.54}$$

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 as previous, but considering a more general scaling matrix $\beta^{(s)}$, defined as the diagonal matrix of weighting coefficients of the interface d.o.f. belonging to the neighbours of $\Omega^{(s)}$.

The choice of this coefficients can be done using a physical criteria. 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, so 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 connected to this d.o.f. in all neighbouring subdomains. For example, for the 4 subdomain case, we have

$$\begin{aligned}
\beta_1^{(1)} &= \frac{k_{11}^{(1)}}{k_{11}^{(1)} + k_{11}^{(2)} + k_{11}^{(3)} + k_{11}^{(4)}} \\
\beta_1^{(2)} &= \frac{k_{11}^{(2)}}{k_{11}^{(1)} + k_{11}^{(2)} + k_{11}^{(3)} + k_{11}^{(4)}} \\
&\vdots \\
\beta_2^{(1)} &= \frac{k_{22}^{(1)}}{k_{22}^{(1)} + k_{22}^{(2)}} \\
\beta_2^{(2)} &= \frac{k_{22}^{(2)}}{k_{22}^{(1)} + k_{22}^{(2)}}
\end{aligned} \tag{2.55}$$

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

$$\beta_{\Gamma^i}^{(s)} = diag\left(K_{\Gamma^i}^{(s)}\right)\left\{\sum_{r:\Gamma^j \subseteq \{\Gamma \cap \Gamma^{(r)}\}} diag\left(K_{\Gamma^j}^{(s)}\right)\right\}^{-1} \tag{2.56}$$

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

$$D^{-1} = \sum_s \beta^{(s)} B^{(s)} S_{bb}^{(s)} B^{(s)^T} \beta^{(s)} = \sum_s \tilde{B}^{(s)} S_{bb}^{(s)} \tilde{B}^{(s)^T} \tag{2.57}$$

$$L^{-1} = \sum_s \beta^{(s)} B^{(s)} K_{bb}^{(s)} B^{(s)^T} \beta^{(s)} = \sum_s \tilde{B}^{(s)} K_{bb}^{(s)} \tilde{B}^{(s)^T} \tag{2.58}$$

with

$$\tilde{B}^{(s)} := \beta^{(s)} B^{(s)} \tag{2.59}$$

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 neighbouring d.o.f.
- Construction of diagonal matrices $\beta^{(s)}$.
- At every multiplication by the preconditioner we add two extra by this local diagonal scaling, which are negligible in parallel computation.

2.1.3 Simultaneous FETI

As seen in previous subsection the Dirichlet (or Lumped) preconditioner 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. This 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 this two different corrected responses independently, as search directions for the Conjugate Gradient method, as shown in [17].

We start again by showing this in a two-subdomain division, so lets consider the p iteration of the CG method applied to the FETI-1LM method (no projection is consider in this simple case). Then, we first solve the local Dirichlet problem consisting in enforcing on the interface the compatibility gap g^p and we compute

the boundary reaction which write

$$\delta f^{(s),p} = S_{bb}^{(s)} \tilde{B}^{(s)^T} g^p, \qquad s = 1, 2 \tag{2.60}$$

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

$$\lambda^{p+1} = \lambda^p + \rho^{(1),p} \tilde{B}^{(1)} \delta f^{(1),p} + \rho^{(2),p} \tilde{B}^{(2)} \delta f^{(2),p} \tag{2.61}$$

Assuming now that, instead of using this boundary reactions directly, we use

$$\begin{aligned} \delta\lambda^{(1),p} &= \tilde{B}^{(1)} \delta f^{(1),p} + \sum_{s=1}^{2} \sum_{l=0}^{p-1} \alpha^{(s),l} \delta\lambda^{(s),l} \\ \delta\lambda^{(2),p} &= \tilde{B}^{(2)} \delta f^{(2),p} + \alpha^{(2),p} \delta f^{(1),p} + \sum_{s=1}^{2} \sum_{l=0}^{p-1} \gamma^{(s),l} \delta\lambda^{(s),l} \end{aligned} \tag{2.62}$$

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

$$\begin{aligned} &\delta\lambda^{(1),p^T} F \delta\lambda^{(2),p} = 0 \\ &\delta\lambda^{(s),p^T} F \delta\lambda^{(r),l} = 0, \qquad r, s = 1, 2 \quad l = 0, \ldots, p-1 \end{aligned} \tag{2.63}$$

Then, the new update is

$$\begin{aligned} \lambda^{p+1} &= \lambda^p + \rho^{(1),p} \delta\lambda^{(1),p} + \rho^{(2),p} \delta\lambda^{(2),p} \\ g^{p+1} &= g^p + \rho^{(1),p} F \delta\lambda^{(1),p} + \rho^{(2),p} F \delta\lambda^{(2),p} \end{aligned} \tag{2.64}$$

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

relations

$$\begin{aligned}(g^{p+1}\cdot\delta\lambda^{(1),p}) &= (g^{p}\cdot\delta\lambda^{(1),p}) + \rho^{(1),p}(F\delta\lambda^{(1),p}\cdot\delta\lambda^{(1),p}) + \rho^{(2),p}(F\delta\lambda^{(2),p}\cdot\delta\lambda^{(1),p})\\ 0 &= (g^{p}\cdot\delta\lambda^{(1),p}) + \rho^{(1),p}(F\delta\lambda^{(1),p}\cdot\delta\lambda^{(1),p})\end{aligned} \tag{2.65}$$

$$\begin{aligned}(g^{p+1}\cdot\delta\lambda^{(2),p}) &= (g^{p}\cdot\delta\lambda^{(2),p}) + \rho^{(1),p}(F\delta\lambda^{(1),p}\cdot\delta\lambda^{(2),p}) + \rho^{(2),p}(F\delta\lambda^{(2),p}\cdot\delta\lambda^{(2),p})\\ 0 &= (g^{p}\cdot\delta\lambda^{(2),p}) + \rho^{(2),p}(F\delta\lambda^{(2),p}\cdot\delta\lambda^{(2),p})\end{aligned} \tag{2.66}$$

which implies

$$\rho^{(s),p} = -\frac{(g^{p}\cdot\delta\lambda^{(s),p})}{(\delta\lambda^{(s),p}\cdot F\delta\lambda^{(s),p})}, \qquad s=1,2 \tag{2.67}$$

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 to the cost of a normal conjugate gradient direction. Actually, recalling that the directions come from a Dirichlet problem, we note that

$$\begin{aligned}F\tilde{B}^{(1)}\delta f^{(1),p} &= \left(\tilde{B}^{(1)}S_{bb}^{(1)^{-1}}\tilde{B}^{(1)^{T}} + \tilde{B}^{(2)}S_{bb}^{(2)^{-1}}\tilde{B}^{(2)^{T}}\right)\tilde{B}^{(1)}S_{bb}^{(1)}\tilde{B}^{(1)^{T}}g^{p}\\ &= g^{p} + \tilde{B}^{(2)}S_{bb}^{(2)^{-1}}\tilde{B}^{(2)^{T}}\tilde{B}^{(1)}\delta f^{(1),p}\end{aligned} \tag{2.68}$$

$$\begin{aligned}F\tilde{B}^{(2)}\delta f^{(2),p} &= \left(\tilde{B}^{(1)}S_{bb}^{(1)^{-1}}\tilde{B}^{(1)^{T}} + \tilde{B}^{(2)}S_{bb}^{(2)^{-1}}\tilde{B}^{(2)^{T}}\right)\tilde{B}^{(2)}S_{bb}^{(2)}\tilde{B}^{(2)^{T}}g^{p}\\ &= g^{p} + \tilde{B}^{(1)}S_{bb}^{(1)^{-1}}\tilde{B}^{(1)^{T}}\tilde{B}^{(2)}\delta f^{(2),p}\end{aligned} \tag{2.69}$$

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 don't expect to have this exact same property of cost equivalency between method, but we do expect a reduced number of multiplication by the FETI operator that will lead to a faster method in general.

We will now show the extension done in [39] 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 generates one for each subdomain, but in a straightforward way this number can be two times the total number of local interfaces in the problem.

In this section the notation and definitions comes 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, used by the Conjugate Gradient algorithm.

The consistent Dirichlet operator, defined in 2.57, is first applied to the global residual vector g

$$w = D^{-1}g = \left(\sum_s D^{(s)^{-1}} \right) g \tag{2.70}$$

where

$$D^{(s)^{-1}} = \tilde{B}^{(s)} S_{bb}^{(s)} \tilde{B}^{(s)^T} \tag{2.71}$$

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 minimisation process in CG by spanning a search space, not from the addition of local contributions, but from each of this terms separately, letting the process to choose the best combination. Although more costly, this optimal combination minimizes the residual in the space generated by

$$Z = \left[D^{(1)^{-1}} g, D^{(2)^{-1}} g, \ldots, D^{(N_s)^{-1}} g \right] \tag{2.72}$$

where N_s is the number of subdomains.

Each one of this columns is then projected and orthogonalized to give the block W of N_s columns, where each column correspond to a new search direction. The update of the solution λ of the CG algorithm at the p iteration is done simultaneously (analogous to Equation 2.64) for all this 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

$$\lambda_{p+1} = \lambda_p + W_p \rho_p \tag{2.73}$$

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

$$\rho_p = (W_p^T F W_p)^+ Z_p^T g_p \tag{2.74}$$

Since the classical search direction is such that

$$w_p = W_p(1,\ldots,1)^T \in range(W_p) \tag{2.75}$$

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

We can see the need of 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 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

$$N^T (W_p^T F W_p) N = L L^T \tag{2.76}$$

where N is a permutation matrix and

$$L = \begin{bmatrix} \tilde{L} & 0 \\ \times & 0 \end{bmatrix} \tag{2.77}$$

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

$$\begin{aligned} W_p &\leftarrow W_p N \begin{bmatrix} \tilde{L}^{-T} \\ 0 \end{bmatrix} \\ FW_p &\leftarrow FW_p N \begin{bmatrix} \tilde{L}^{-T} \\ 0 \end{bmatrix} \end{aligned} \tag{2.78}$$

where we are suppressing the redundant directions. The computation of the optimization parameters ρ_p is now

$$\rho_p = W_p^T g_p \tag{2.79}$$

In Algorithm 8 we can find the description for this method. The definition of the projection and the matrices to compute λ_0 are defined as in previous FETI methods (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 in order 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*

$$\lambda_p \in \bigoplus_{i=0}^{p-1} span\left\{W_i\right\} \tag{2.80}$$

where $\oplus$ *is the direct sum and* W_i *is defined by Algorithm 8*

Proof. The proof is done in [39] 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 won't allow to find a bound (heuristic) for the number of iterations, but in any case we will expect a good 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

$$S_{bb}^{(s)} v^{(s)} - \mu B^{(s)^T} D^{-1} B^{(s)} v^{(s)} = 0 \qquad s = 1, \ldots, N_s \tag{2.81}$$

This problem allows to isolate 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 in account. It is then similar to the deflated space where the Geneo iterations take place, and thus convergence is expected to be very fast.

Extension to local interface division

In the regular first version of the S-FETI method, the preconditioner is decomposed in the local contributions

$$D^{(s)^{-1}} = B^{(s)} S_{bb}^{(s)} B^{(s)^T} \tag{2.82}$$

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, but different interactions can occur for the same subdomain, depending on the characteristics of each of his neighbours. If all of the neighbours 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 neighbour, to take into account this differences, increasing the number of search directions built and improving the approximation of the local behaviour, hence ameliorating the convergence of the method.

The formal construction of this search directions comes from the definition of the preconditioner, and the definition of the interface edge given in previous

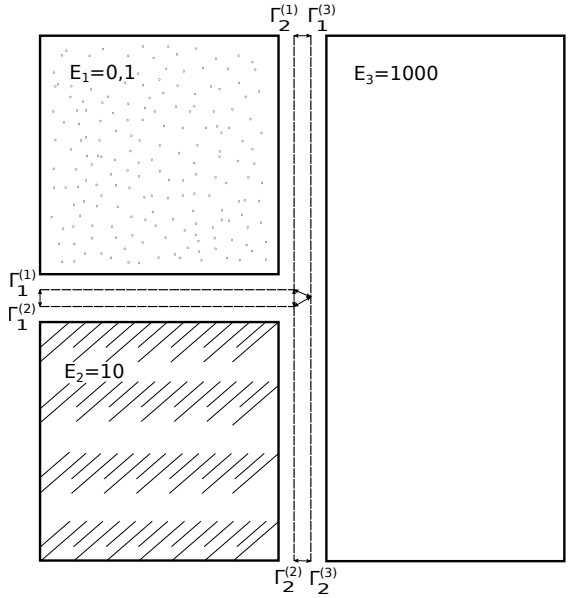

Figure 2.3 – Local interfaces with completely different subdomains (Young's modulus $0,1 \leq E \leq 1000$)

sections. Lets consider the Dirichlet preconditioner with any consistent scaling

$$D^{-1} = \sum_{s=1}^{N_s} \tilde{B}^{(s)} S_{bb}^{(s)} \tilde{B}^{(s)^T} \tag{2.83}$$

then in the S-FETI method we consider the local contributions separately

$$D^{(s)^{-1}} = \tilde{B}^{(s)} S_{bb}^{(s)} \tilde{B}^{(s)^T} \tag{2.84}$$

from which we created the N_s different search directions. This operator is applied in the CG method to the computed gradient g, defined in the total 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

$$\delta f_{bb}^{(s)} = S_{bb}^{(s)} \tilde{B}^{(s)^T} g \tag{2.85}$$

We know that this forces are defined in the local interface $\Gamma^{(s)}$, but since their interactions may change from one neighbour to another, we create new vectors

that reflect this interactions separately by considering the restrictions to the interface edges $\Gamma^{(sq)} = \partial\Omega^{(s)} \cup \partial\Omega^{(q)}$

$$\delta f^{(s)}_{\Gamma^{(sq)}} = \delta f^{(s)}_{bb}|_{\Gamma^{(sq)}} \qquad s = 1,\dots,N_s, \quad q = 1,\dots,n^{(s)} \tag{2.86}$$

where $n^{(s)}$ is the number of neighbour subdomains in $\Omega^{(s)}$. We extend this vector by zero to match the size of the interface $\Gamma^{(s)}$

$$\tilde{f}^{(s)}_{\Gamma^{(sq)}} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \delta f^{(s)}_{\Gamma^{(sq)}} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{2.87}$$

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

$$D^{(s)^{-1}}_{\Gamma^{(sq)}} g = \tilde{B}^{(s)} \tilde{f}^{(s)}_{\Gamma^{(sq)}} \qquad s = 1,\dots,N_s, \quad q = 1,\dots,n^{(s)} \tag{2.88}$$

so finally the search space will now be generated by the block

$$Z = \left[\dots, D^{(s)^{-1}}_{\Gamma^{(sq)}} g, \dots\right] \qquad s = 1,\dots,N_s, \quad q = 1,\dots,n^{(s)} \tag{2.89}$$

in this case we won't be having one column from each subdomain, but instead there will be one for each local edge (or neighbour). This will allow to span an even bigger search space at the cost of computing all the extra directions.

Graphically we can see the difference between this block and the regular one,

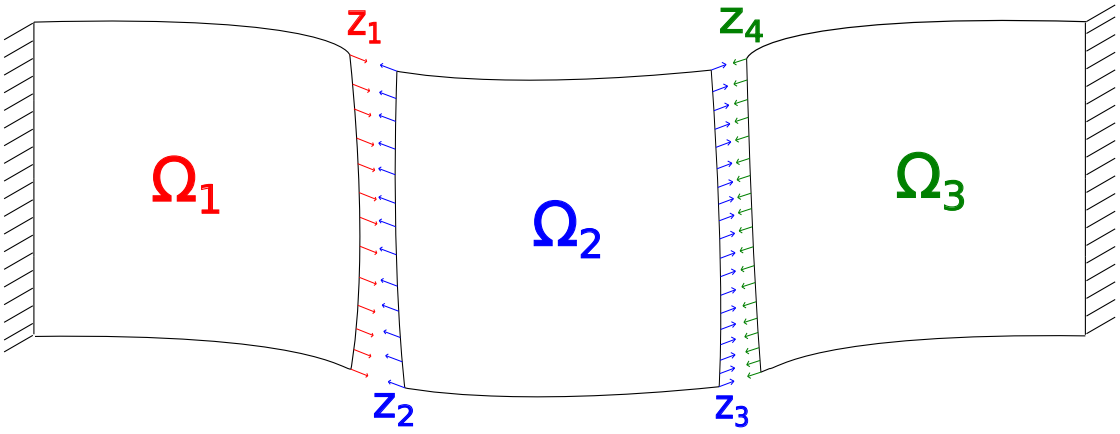

Figure 2.4 – Three subdomain subdivision and computed corrections

by looking at Figure 2.4 an noting that the blocks in both cases will be

$$\begin{aligned} \text{Subdomain division} \Rightarrow Z &= \begin{bmatrix} z_1 & z_2 & 0 \\ 0 & z_3 & z_4 \end{bmatrix} \\ \text{Interface division} \Rightarrow Z &= \begin{bmatrix} z_1 & z_2 & 0 & 0 \\ 0 & 0 & z_3 & z_4 \end{bmatrix} \end{aligned} \tag{2.90}$$

We note here that when the problem is divided in much more subdomains, the block Z will show an sparse pattern, since in every column, only the values associated to $\Gamma^{(s)}$ (or in particular $\Gamma^{(sq)}$) are non zero. This fact will be a 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 directions. Then we use the same rank revealing strategy 2.78 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 9.

Both algorithm for S-FETI, shows 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.

Algorithm 9 S-FETI algorithm interface

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

2.1.5 Cost and implementation of S-FETI

One of the characteristic 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 in each iteration does require a much larger computational effort, several strategies in the implementation of the code leads to efficient local and global optimizations, in order to reduce this cost differences. We start by summarizing the main changes in the cost of S-FETI vs FETI-1LM.

1. The number of exchanges phases is the same. The communications between neighbours 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, where N_T is either the number of subdomains (in the basic S-FETI) or $N_T = \mathcal{O}(N_s)$ (actually is two times the number of local interfaces) when using the local interface subdivision. Also the full reorthogonalization process is now done in a block way, so we change the vector operations by matrix operations, so again, same number of exchanges and computation, but with more data involved.

   The advantages from keeping the number of interchanges, but increasing the size of the data are more clear 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

   $$T(n) = \alpha + n\beta \tag{2.91}$$

   where

   $$\begin{aligned} &n : number\ of\ data\ items. \\ &\alpha : startup\ time. \\ &\beta : transmission\ time\ per\ data\ item. \end{aligned} \tag{2.92}$$

The start-up 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 greater than β, where β is on the order of the cost of an instruction.

All this considerations, gives us some general lower bounds in time for the communications, the most important one in the start-up time, which depends on the number of processes (subdomains), and is independent of the type of communication, either send-receive or reduce operations. This is

$$T(n) \geq \log_2(p)\alpha, \qquad p : processes \tag{2.93}$$

For this reason, avoiding communication is a main issue in parallel programming and 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 in every iteration, we now store 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 mayor 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, this computations can be performed efficiently, see Table 2.1.

As already said in the first point, communication cost is mainly driven by the start-up time, and 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 start-up 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. 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. Lets note that the matrix Z_{p+1} is a sparse matrix, this is due to the fact that each column associated to the preconditioner coming from the subdomain $\Omega^{(s)}$ is non zero only in the local interface $\Gamma^{(s)}$ (or $\Gamma^{(sq)}$) while in contrast W_{p+1} it is not, because of the projection and orthogonalization. Moreover, we can observe that

$$\begin{aligned} FW_{j+1} &= FPZ_{j+1} + \sum_{i=0}^{j} FW_i\Phi_{ij} \\ &= \left(FZ_{j+1} - FAG[G^TAG]^{-1}G^TZ_{j+1}\right) + \sum_{i=0}^{j} FW_i\Phi_{ij} \end{aligned} \tag{2.94}$$

where we remember that A is a symmetric matrix, that can be the taken as the preconditioner, some scaling matrix or the identity matrix in the simplest case.

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

$$A = \sum_{s=1}^{N_s} \tilde{B}^{(s)} diag\left(K_{bb}^{(s)}\right) \tilde{B}^{(s)^T} \tag{2.95}$$

This is the so called $SumperLumped$ 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 ratio of the FETI method is improved, see [36].

Previous equation shows that in every iteration we only need to solve the localized problems FZ_{p+1} using the sparsity of Z_{p+1} and the computation of the projection is done by computing the also sparse matrix (FAG) during the initialization once and for all.

More details of this computations will be given in the details of the implementation to come next.

Remark: With all this differences, the final extra cost per iteration remains very small 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 costs needs to be compared with the final number of iterations given by 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 in order to improve the efficiency of S-FETI, we can now give some details of the practical implementation

1. Lets return to the last point in the cost of the S-FETI method, the simulta-

neous 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 only in the subdomain itself and its neighbour $\Omega^{(q)}$. On the other hand, given one subdomain $\Omega^{(s)}$, the workload associated with the computation of FZ_p is $\mathcal{N}$ Neumann solves, where

$$\begin{aligned} \text{Subdomain division} &\Rightarrow \mathcal{N} = neighbours + 1 \\ \text{Interface division} &\Rightarrow \mathcal{N} = 2 \times neighbours \end{aligned} \tag{2.96}$$

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 subdomain 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 number 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. This leveles can be seen as a pyramid, where the top of the pyramid contains the memory that is the fastest, and also the smallest, and 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 they can take the greatest advantage of this hierarchy is that their memory interaction can be organized in a fashion such that it takes advantage of the various

Number of cores	Substitutions	Time (s)
1	1	0.7
12	12	1

Table 2.1 – Time for forward-backward substitution on multi-core processor.

stages of the pyramid, this leads to have an order of magnitude more computation than memory interaction. In Level-2 BLAS we have $O(n^2)$ data moves and $O(n^2)$ operations, whereas in BLAS3 we have the same number of data moves $O(n^2)$ but we improve the operations to $O(n^3)$.

The pyramid concept does not completely map to modern multi-core optimization, however 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.

As an example given in [39] 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 this 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 important issue is the parallel implementation of the projector P. Lets recall its definition

$$P = I - AG(G^T AG)^{-1} G^T \tag{2.97}$$

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 consider

$$P_A Z_p = Z_p - (AG)\alpha_p \text{ where } \alpha_p \text{ solves } (G^T AG)\alpha_p = G^T Z_p \tag{2.98}$$

This will guarantee that $G^T P_I Z_p = 0$. We recall the definition of G

$$G = \left[B^{(1)} t^{(1)} R^{(1)} \dots B^{(N_s)} t^{(N_s)} R^{(N_s)}\right] \tag{2.99}$$

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 to 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 neighbours.

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 for his own columns (the ones associated to $\Gamma^{(sq)}$, $q = 1, \dots, n^{(s)}$) the following system

$$(G^T AG)\alpha_p^{(sq)} = G^T Z_p^{(sq)} \tag{2.100}$$

We know that α_p is a dense matrix with the same number of rows that the $rank(G^T)$ and with N_T columns (the same as Z_p), but in any case, once it is computed we only require a low-rank correction of Z_p in each subdomain, because only a few columns of G are nonzero in $\Omega^{(s)}$, with this we compute the total correction

$$P_A Z_p = Z_p - AG\alpha_p \tag{2.101}$$

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*, this 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 neighbours) modes that describes Z or G, so for each interface we save one local mode and one from its neighbour. This double storage allows to perform computations of, for example $G^T Z$, in each subdomain by only performing dot products between the coarse modes saved in each interface of each subdomain. This way of storage is generalized to allow local computations between this 4 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 or in the basic case they are considered as an single mode (when doing the local subdivision it is $2 \times neighours$), and then also storing 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 describes FZ or FAG stored in one of the interfaces of certain subdomains. For each local interface we save the modes corresponding to the local modes after the forward-backward substitution plus the modes of the neighbour after the substitution.

Again, this implementation in form of coarse modes, allows the computations of matrix products such as $G^T(FAG)$, only by performing dot products between this modes. This implementation also explains how the single multi-rhs substitution is done, 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 improve the performance of this type of computations (See previous point).

3. The previous process it is also important in the calculations to obtain FW_p. Lets recall that the orthogonalization process that leads to FW_p is applied to the block FPZ_p, so we need first to compute this block. To do so, we use

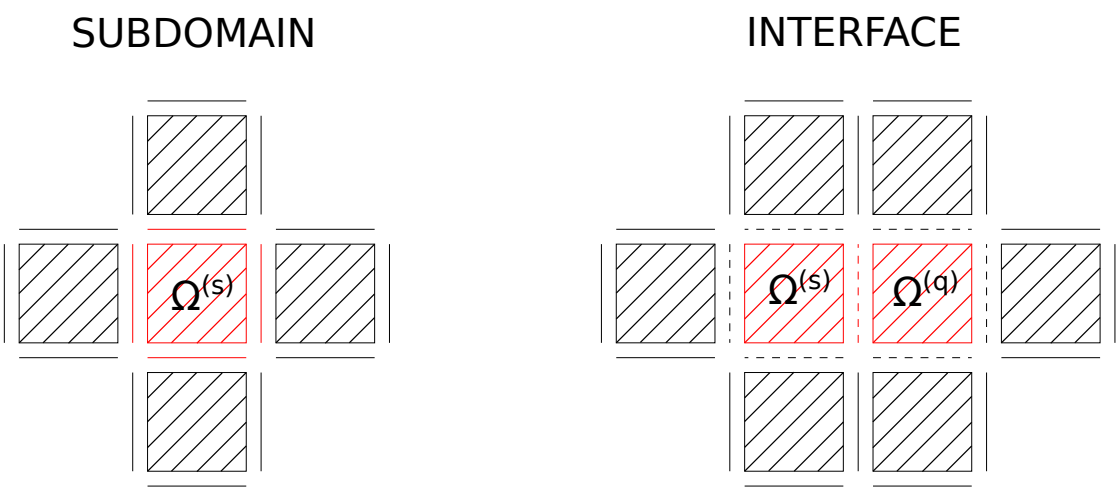

Figure 2.5 –
Subdomain point of view (Left): In red the coarse modes describing Z or G owned by subdomain $\Omega^{(s)}$, in black the interfaces where F times the red modes is also non null.
Interface point of view (Right): In doted lines the coarse modes describing FZ or FAG owned by local interface $\Gamma^{(sq)}$ between subdomains $\Omega^{(s)}$ and $\Omega^{(q)}$, in black the modes where, due to $\Gamma^{(sq)}$, $Z^T(FZ)$ is non null.

the matrix α_p previously computed in the previous point, the we need to do

$$FPZ_p = FZ_p - (FAG)\alpha_p \tag{2.102}$$

Again what we are doing here is a low-rank correction, but this time of FZ_p. The difference is that,to do this corrections, we now use the sparse pattern of (FAG) (each column is nonzero in $\Omega^{(s)}$ and its neighbours) 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 classical 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 this many communications we can compute $(PZ_p)^T(FPZ_p)$ by using the local contribution of each subdomain obtained with the BLAS3 and then doing 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 this same BLAS3 strategy in order to have a number of MPI reductions equal to the ones used in classical FETI, but changing from one vector to a matrix and also changing the size of the reductions from 1 to at most $N_T \times N_T$.

The evaluation of all this 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 is proved to be a great improvement in comparison with his basic version, all based in the idea 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 name 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 dependency that may occur in the directions built so 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 his impact in 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, that is computed in every iteration as

$$W_p = PZ_p + \sum_{i=0}^{p-1} W_i \Phi_{ip} \tag{2.103}$$

This block is orthogonalized with respect to previous blocks W_i already stored. Also the columns of each of the stored block, need to form an F-orthonormal family, which is the norm used in CG algorithm, i.e. for every block they need to fulfill

$$W_p^T F W_p = I \tag{2.104}$$

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's clear that this matrix is also symmetric, but it is not so evident that 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 certain point of the main iteration. This dependence comes mainly from two issues, the first is the convergence in 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 neighbour subdomain, and the characteristics of each of this local interfaces varies 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 characteristic have an impact in 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 is made of 1 node (e.g. in a corner) the convergence is achieved in 2 iterations, so the columns created from this interface (in the subdivided version of S-FETI) will be no longer useful and the linear dependence appears.

For the problem of representation in limited precision arithmetic, depending on the configuration, but also closely related to the phenomenon of achieved convergence, we can have directions whose differences are so small that it's difficult to distinguish it from numeric noise, meaning that in practice we have the same direction, so the use of them 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 full 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 block with the exact 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 [39] a Cholesky factorization is proposed, but we can also add a second way that consist in apply 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

$$W_p^T F W_p = L_p L_p^T \tag{2.105}$$

This decomposition only works if the previous matrix is of full ranking, since this is not our case, we need to use the alternative Cholesky decomposition with complete pivoting (symmetric pivoting) described in [44]. The algorithm for this factorization is proved to work as a rank revealing procedure. This is achieved by computing an square permutation matrix N, in this case the decomposition can be written as

$$N^T W_p^T F W_p N = L_p L_p^T \tag{2.106}$$

and if we look in detail the matrix L_p we know that it can be described as

$$L_p = \begin{bmatrix} \tilde{L}_p & 0 \\ \times & 0 \end{bmatrix} \tag{2.107}$$

where

$$\tilde{L}_p \in \mathbb{R}^{r \times r} \tag{2.108}$$

this smaller matrix is lower triangular with positive elements in the diagonal and it has a full rank $r \leq N_T$, which is at the same time the rank of the $W_p^T F W_p$ matrix.

The existence of the permutation matrix (proved in [46], Thm. 10.9) allows to sort the directions that are useful from the rest. This is simply done 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

$$W_p \leftarrow (W_p N) \begin{bmatrix} \tilde{L}_p^{-1} & 0 \end{bmatrix}^T \tag{2.109}$$

this block of sorted directions fulfills the 2.104 condition, in fact

$$\begin{aligned} \begin{bmatrix} \tilde{L}_p^{-1} & 0 \end{bmatrix} (W_p N)^T F (W_p N) \begin{bmatrix} \tilde{L}_p^{-1} & 0 \end{bmatrix}^T &= \begin{bmatrix} \tilde{L}_p^{-1} & 0 \end{bmatrix} L_p L_p^T \begin{bmatrix} \tilde{L}_p^{-1} & 0 \end{bmatrix}^T \\ &= \begin{bmatrix} \tilde{L}_p^{-1} \tilde{L}_p & 0 \end{bmatrix} \begin{bmatrix} \tilde{L}_p^{-1} \tilde{L}_p & 0 \end{bmatrix}^T \\ &= \begin{bmatrix} I \\ 0 \end{bmatrix} \end{aligned} \tag{2.110}$$

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, but 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 easy task and is usually chosen based in numerical experiences, but as a basic consideration, we'll try to be closer to the

"losing" directions approach because, even if we can have a slower convergence, eventually we will recover this directions in later iterations. On the other hand, working within the range of the natural noise of the method, can only introduce sources of error harder to control, and may even be ending in blowups.

Usually a regular double precision implementation should be enough, but introducing new sources of numerical instabilities needs to be done carefully, so later in numerical tests we'll 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 search directions block

The second way to treat the sorting of independent search directions can be done using a diagonalization process. At first we now that this procedure will lead to the decomposition

$$W_p^T F W_p = E_p D_p E_p^{-1} \tag{2.111}$$

Where E_p is the matrix of the eigenvectors in the columns, and D_p is a diagonal matrix with the eigenvalues in 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 2.104 we can use the previous decomposition to build it. First, we know that

$$W_p^T F W_p = E_p D_p E_p^T \tag{2.112}$$

this implies that

$$E_p^T W_p^T F W_p E_p = D_p \tag{2.113}$$

and since the matrix D_p is diagonal, 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

$$\begin{aligned} E_p^T W_p^T F W_p E_p &= D_p^{1/2} D_p^{1/2} \\ (W_p E_p D_p^{-1/2})^T F (W_p E_p D_p^{-1/2}) &= I \end{aligned} \tag{2.114}$$

And so our F-orthogonalized search directions can be constructed by doing

$$W_p \leftarrow W_p E_p D_p^{-1/2} \tag{2.115}$$

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 eigenvectors are greater or equal than zero.

The common 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 biggest of this eigenvalues and their associated eigenvectors, so again, the construction is the same as previous, but we will only consider the r biggest eigenvalues to build then the block

$$W_p \leftarrow W_p \tilde{E}_p \tilde{D}_p^{-1/2} \tag{2.116}$$

where

$$rank(W_p^T F W_p) = rank(\tilde{E}_p^T) = rank(\tilde{D}_p) = r \tag{2.117}$$

meaning that 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 clearer way, because 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 so we can eliminate the smallest eigenvalues and vectors below the threshold ε. In this case the dependence between this parameter and the final result is less important 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.

The impact in 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	Memory (Gb)	Directions stored
125	7	484	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 add into this new decomposition.

2.3 Memory usage in S-FETI

In previous sections we stipulate that one of the difference between the S-FETI method and the classical FETI lies in the total memory use of both of them. For the classical FETI, 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 is in fact one main 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 linearity of the directions, we expect a smaller number that correspond 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 small cube divided into $5 \times 5 \times 5$ smaller cubes, we see in 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 for the total memory needed this 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 by the exchange of information between processes, so if we add some, not too expensive, extra computation 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 this sparse matrices we also compute and store some smaller coefficient matrices that will allow this reconstruction. The size of this new matrices will depend on the number of used search directions at each iteration, and will be independent of the size of the mesh, so 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 in a more important way on the problem configurations, that determines the maximum total search directions in every iteration, rather than the size of the discretization. Compared with the total unknowns of the problem, this number (N_T) is usually a small one, because the number of neighbours of each subdomain is also limited, so the problems that we can solve can increase in its total size and have a smaller impact in the memory used by it, allowing us to use the S-FETI method in bigger problems.

The fact that we no longer store the complete search directions will also have an impact in the precision achieved with this implementation, but we hope to keep the good 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

Use the definition of the projector and full reorthogonalization process, to try to find a recurrence for the coefficients that define each search direction. Lets

consider the sparse matrices Z_p from 2.89 and lets recall the definition of the projector

$$P = I - AG(G^T AG)G^T \tag{2.118}$$

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 2.101. Starting from Z_p we have

$$Z_1 \Longrightarrow PZ_1 = Z_1 + (AG)D_1 \tag{2.119}$$

its clear that

$$G^T PZ_1 = 0 \tag{2.120}$$

and that the computation of D_1 is done by solving

$$(G^T AG)D_1 = -G^T Z_1 \tag{2.121}$$

meaning that the value of D_1 is

$$D_1 = -(G^T AG)^{-1} G^T Z_1 \tag{2.122}$$

Since no previous directions were computed, we have our first search direction block, and the projection coefficients that define it

$$\begin{aligned} W_1 &:= Z_1 \\ \Delta_1 &:= D_1 \\ PW_1 &:= PZ_1 = Z_1 + (AG)\Delta_1 \end{aligned} \tag{2.123}$$

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

$$Z_2 \Longrightarrow PZ_2 = Z_2 + (AG)D_2 \tag{2.124}$$

where

$$G^T PZ_2 = 0 \tag{2.125}$$

and D_2 is computed by solving

$$(G^T AG)D_2 = -G^T Z_2 \tag{2.126}$$

so again, its value is

$$D_2 = -(G^T AG)^{-1} G^T Z_2 \tag{2.127}$$

Now, we need to consider the full reorthogonalization against the directions built in the previous step. Hence we have

$$\begin{aligned} W_2 &= Z_2 + W_1 \Phi_{12} \\ \Longrightarrow PW_2 &= PZ_2 + PW_1 \Phi_{12} \end{aligned} \tag{2.128}$$

and to compute Φ_{12} we use the orthogonality properties of the new directions and the previous ones in the CG algorithm

$$(FPW_1)^T PW_2 = 0 \tag{2.129}$$

replacing PW_2 we have

$$(FPW_1)^T (PW_1)\Phi_{12} = -(FPW_1)^T PZ_2 \tag{2.130}$$

this problem is solved in order to compute Φ_{12} whose value is

$$\Phi_{12} = -\left[(PW_1)^T (FPW_1)\right]^{-1} (FPW_1)^T PZ_2 \tag{2.131}$$

then, in 2.128 we use the projection definition again, to obtain

$$\begin{aligned} PW_2 &= Z_2 + (AG)D_2 + (Z_1 + (AG)D_1)\Phi_{12} \\ &= Z_2 + Z_1 \Phi_{12} + (AG)(D_2 + D_1 \Phi_{12}) \end{aligned} \tag{2.132}$$

so

$$PW_2 = Z_2 + Z_1 B_{12} + (AG)\Delta_2 \tag{2.133}$$

where

$$B_{12} := \Phi_{12} \text{ and } \Delta_2 := D_2 + D_1\Phi_{12} \tag{2.134}$$

but also, since

$$W_2 = Z_2 + Z_1 B_{12} \tag{2.135}$$

the we have that

$$PW_2 = W_2 + (AG)\Delta_2 \tag{2.136}$$

from this equation, plus from 2.128 and 2.133 we extend the recurrence to a general case where we have the next equivalent equations

$$PW_p = W_p + (AG)\Delta_p \tag{2.137}$$

$$PW_p = PZ_p + \sum_{j=1}^{p-1} PW_j\Phi_{jp} \tag{2.138}$$

$$PW_p = Z_p + \sum_{j=1}^{p-1} Z_j B_{jp} + (AG)\Delta_p \tag{2.139}$$

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. This matrices along with the sparse ones Z_p define the construction of the search directions.

Remark: From previous construction we can extend the sparse multiplication against the operator F, namely from Equation 2.133 we have

$$FPW_2 = FZ_2 + FZ_1 B_{12} + (FAG)\Delta_2 \tag{2.140}$$

and then in the general form, using Equation 2.139 we have

$$FPW_p = FZ_p + \sum_{j=1}^{p-1} FZ_j B_{jp} + (FAG)\Delta_p \tag{2.141}$$

Formal construction

At the p iteration, we apply to Z_p the low-rank correction definition of P

$$PZ_p = Z_p + (AG)D_p \tag{2.142}$$

with

$$D_p = -(G^T AG)^{-1} G^T Z_p \tag{2.143}$$

also from the definition of P, we apply it this time to the reorthogonalized block W_p

$$PW_p = W_p + (AG)\Delta_p \tag{2.144}$$

later we will compute Δ_p, not from the classical definition, but recursively.

On the other hand and to avoid notation problems, lets consider some iteration j. We replace in the previous equation 2.144 the computation of W_j (W_p in there) by the Gram-Schmidt orthogonalization process applied to Z_j

$$PW_j = Z_j + \sum_{i=1}^{j-1} Z_i B_{ij} + (AG)\Delta_j \tag{2.145}$$

with B_{ij} to be computed later, recursively. If we replace this in 2.138 that actually correspond to the formal construction of the orthogonalized and projected search directions, then we have

$$\begin{aligned} PW_p &= PZ_p + \sum_{j=1}^{p-1} \left(Z_j + \sum_{i=1}^{j-1} Z_i B_{ij} + (AG)\Delta_j \right) \Phi_{jp} \\ &= Z_p + (AG)D_p + \sum_{j=1}^{p-1} Z_j \Phi_{jp} + \sum_{j=1}^{p-1} \sum_{i=1}^{j-1} Z_i B_{ij} \Phi_{jp} + \sum_{j=1}^{p-1} (AG)\Delta_j \Phi_{jp} \\ &= \left\{ Z_p + \sum_{j=1}^{p-1} Z_j \Phi_{jp} + \sum_{j=1}^{p-1} \sum_{i=1}^{j-1} Z_i B_{ij} \Phi_{jp} \right\} + (AG) \left\{ \sum_{j=1}^{p-1} \Delta_j \Phi_{jp} + D_p \right\} \end{aligned} \tag{2.146}$$

from this equation and 2.144 it's clear that we can define the computation of W_p

and Δ_p as

$$W_p = Z_p + \sum_{j=1}^{p-1} Z_j \Phi_{jp} + \sum_{j=1}^{p-1} \sum_{i=1}^{j-1} Z_i B_{ij} \Phi_{jp} \tag{2.147}$$

$$\Delta_p = \sum_{j=1}^{p-1} \Delta_j \Phi_{jp} + D_p \tag{2.148}$$

Developing the computation of W_p

$$\begin{aligned} W_p &= Z_p + \sum_{j=1}^{p-1} Z_j \Phi_{jp} + \sum_{j=1}^{p-1} \sum_{i=1}^{j-1} Z_i B_{ij} \Phi_{jp} \\ &= Z_p + \sum_{j=1}^{p-1} Z_j \Phi_{jp} + \sum_{i=1}^{p-2} Z_i \left(\sum_{j=i+1}^{p-1} B_{ij} \Phi_{jp} \right) \\ &= Z_p + \sum_{i=1}^{p-1} Z_i \Phi_{ip} + \sum_{i=1}^{p-2} Z_i \left(\sum_{j=i+1}^{p-1} B_{ij} \Phi_{jp} \right) \end{aligned} \tag{2.149}$$

and thus our orthogonalized search direction is

$$W_p = Z_p + \sum_{i=1}^{p-1} Z_i B_{ip} \tag{2.150}$$

with

$$B_{ip} := \Phi_{ip} + \sum_{j=i+1}^{p-1} B_{ij} \Phi_{jp} \tag{2.151}$$

Finally, using the computation of W_p from 2.150 and Δ_p defined recursively in 2.148, the projected search directions at the p iteration can now be computed as

$$PW_p = W_p + (AG)\Delta_p \tag{2.152}$$

So we can now 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}, $i = 1,\dots,j$, $j = 1,\dots p$ that are stored instead of the full larger matrices W_p or FW_p.

At each iteration with start with the Z_p matrix that is built from previous gradient and with it we can easily compute the coefficients D_p that do not need to be stored (we can, for example use the same memory space that uses Δ_p and later overwrite it).

To define the computation of the coefficients in Φ_{ip} we need to add some extra calculus, that cannot be based in the storage of any mesh dependant full vector. To achieve this, lets consider at each iteration the generalization of 2.129, thats based in the general orthogonal properties of the CG algorithm

$$\begin{aligned} (FPW_j)^T PW_p &= 0 \qquad \text{for } j < p \\ (FPW_j)^T (PZ_p + \sum_{j=1}^{p-1} PW_j \Phi_{jp}) &= 0 \end{aligned} \tag{2.153}$$

which implies

$$(FPW_j)^T (PW_j)\Phi_{jp} = -(FPW_j)^T PZ_p \qquad \text{for } j < p \tag{2.154}$$

The left side matrix correspond 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 to it, giving

$$L_j L_j^T \Phi_{jp} = -\left(FPW_j\right)^T PZ_p \tag{2.155}$$

The L_j matrix need to be stored at every iteration, but then again this are small triangular matrices of size $rank(Z_j^T)$.

The right hand side, is the one needing a special treatment as 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, as 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 2.145 and the fact that F is a linear operator,

we have

$$FPW_j = FZ_j + \sum_{i=1}^{j-1} FZ_i B_{ij} + (FAG)\Delta_j \tag{2.156}$$

From this, as expected, we also need to store the sparse term FZ_j.
Finally the right term in 2.155 is now computed

$$\begin{aligned}\left(FPW_j\right)^T (PZ_p) &= \left[FZ_j + \sum_{i=1}^{j-1} FZ_i B_{ij} + (FAG)\Delta_j\right]^T \left[Z_p + (AG)D_p\right] \\ &= Z_j^T (FZ_p) + \sum_{i=1}^{j-1} B_{ij}^T Z_i^T (FZ_p) + \Delta_j^T (FAG)^T Z_p + \\ &\quad + Z_j^T (FAG)D_p + \sum_{i=1}^{j-1} B_{ij}^T Z_i^T (FAG)D_p + \Delta_j^T (AG)^T (FAG)D_p\end{aligned} \tag{2.157}$$

We have to compute the six previous terms at every iteration for $j = 1, p-1$. With this we can know obtain from 2.155 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.

$$\begin{aligned}(AG)^T(FAG) &: \text{ Computed once for all and sparse stored} \\ Z_j^T FZ_p &: \text{ Computed at every iteration for } j = 1, p-1 \text{ and sparse stored} \\ Z_p^T(FAG) &: \text{ Computed at every iteration and sparse stored}\end{aligned} \tag{2.158}$$

$$\begin{aligned}D_p &: \text{ Computed at every iteration and overwritten by } \Delta_p \\ \Delta_p &: \text{ Small matrix stored every iteration} \\ (AG)^T(FAG)\Delta_p &: \text{ Small matrix stored every iteration} \\ \Phi_{jp} &: \text{ Small matrices stored every iteration, for } j = 1, p-1 \\ B_{jp} &: \text{ Small matrices stored every iteration, for } j = 1, p-1\end{aligned} \tag{2.159}$$

With all this considerations, we can describe the S-FETI method with this new search directions storage in Algorithm 10

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 and reorthogonalized blocks

  $FW = Q_p + \sum_{j=0}^{p-1} Q_j B_{jp} + (FAG)\Delta_p$

  $N_p L_p L_p^T N_p^T = W^T (FW)$

  $\rho = -(\tilde{L}_p \tilde{L}_p^T)^{-1} (WN_p)^T g_p$ ▷ $\tilde{L}_p \in \mathbb{R}^{r \times r}$ is full ranked

  $\lambda_{p+1} = \lambda_p + (WN_p)\rho$

  $g_{p+1} = g_p + (FWN_p)\rho$

end loop

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 previous section, in order to save 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 this new computations needs to be implemented in the most efficient way possible. The algorithm shown in 10 describes a simple presentation of the construction of the search directions, that can be implemented in a straitforward way, but here we will give some considerations that can give an important time boost to the implementation of this method.

Before explaining this, we will explain the sparsity of the new extra terms 2.158 that we need to compute and store, lets recall them

$$(G)^T(FG),\ Z_j^T F Z_p,\ Z_p^T(FG) \tag{2.160}$$

with $j = 1,\ldots,p-1$ and for simplicity we consider $A = I$. If we look at this three terms we note that each one shares very similar structures, this comes from the definitions of Z and G

$$\begin{aligned} G &= \left[B^{(1)}t^{(1)}R^{(1)},\ldots,B^{(N_s)}t^{(N_s)}R^{(N_s)}\right] \\ Z &= \left[B^{(1)}\Delta f^{(1)},\ldots,B^{(N_s)}\Delta f^{(N_s)}\right] \end{aligned} \tag{2.161}$$

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 to a subdomain s can be consider to have the exact same structure.

With this definitions we see that G and Z have the same sparse structure given by the definition of boolean matrices $B^{(s)}$, to see it more clearly, lets take as example the Figure 2.6, where the domain is divided in 6 subdomains in a

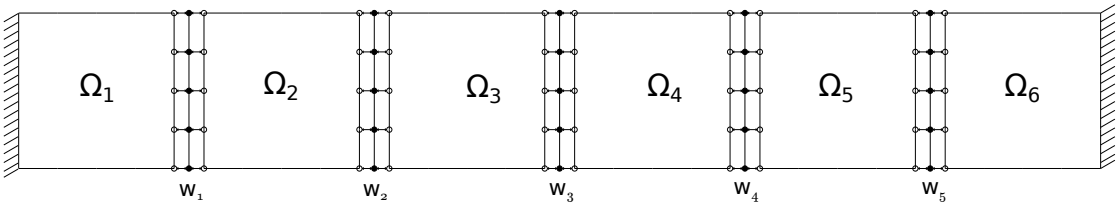

Figure 2.6 – One-way split of six subdomains

one-way split. Lets denote any vector defined in the interface of subdomain s as $v_b^{(s)}$, and divide a vector w along the total interface into local interface blocks

$$w = \begin{bmatrix} w^1 \\ w^2 \\ w^3 \\ w^4 \\ w^5 \end{bmatrix} \tag{2.162}$$

we see that the size and structure of w is 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

$$\begin{aligned} B^{(1)}v_b^{(1)} &= \begin{bmatrix} \times \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad B^{(2)}v_b^{(2)} = \begin{bmatrix} \times \\ \times \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad B^{(3)}v_b^{(3)} = \begin{bmatrix} 0 \\ \times \\ \times \\ 0 \\ 0 \end{bmatrix}, \\ B^{(4)}v_b^{(4)} &= \begin{bmatrix} 0 \\ 0 \\ \times \\ \times \\ 0 \end{bmatrix}, \quad B^{(5)}v_b^{(5)} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \times \\ \times \end{bmatrix}, \quad B^{(6)}v_b^{(6)} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \times \end{bmatrix}, \end{aligned} \tag{2.163}$$

Here we see that in each column only a small part of the interface is non zero, and it correspond to the d.o.f. in the column s shared by the subdomain s ($\Omega^{(s)} \cap \Gamma \neq 0$).

The next structure to consider is the one of the terms FG and FZ, that are the multiplications of previous blocks by the FETI operator. From the definition of F we can see the structure of each previous blocks multiplied by F depends on the multiplication of the boolean matrices

$$B^{(s)^T} B^{(q)},\ \forall s,q = 1,\ldots,N_s \tag{2.164}$$

When subdomains s and q are neighbours, this multiplication is non zero, so when multiplying Z or G by F only local interfaces corresponding to the neighbours are non zero. In our example the structures are

$$\begin{aligned} FB^{(1)}v_b^{(1)} &= \begin{bmatrix}\times\\ \times\\ 0\\ 0\\ 0\end{bmatrix},\ FB^{(2)}v_b^{(2)} = \begin{bmatrix}\times\\ \times\\ \times\\ 0\\ 0\end{bmatrix},\ FB^{(3)}v_b^{(3)} = \begin{bmatrix}\times\\ \times\\ \times\\ \times\\ 0\end{bmatrix}, \\ FB^{(4)}v_b^{(4)} &= \begin{bmatrix}0\\ \times\\ \times\\ \times\\ \times\end{bmatrix},\ FB^{(5)}v_b^{(5)} = \begin{bmatrix}0\\ 0\\ \times\\ \times\\ \times\end{bmatrix},\ FB^{(6)}v_b^{(6)} = \begin{bmatrix}0\\ 0\\ 0\\ \times\\ \times\end{bmatrix}, \end{aligned} \tag{2.165}$$

With previous blocks, the final structure for the terms $(G)^T(FG)$, $Z_j^T FZ_p$, $Z_p^T(FG)$ is given, in this case, by

$$\begin{bmatrix}\times & 0 & 0 & 0 & 0\\ \times & \times & 0 & 0 & 0\\ 0 & \times & \times & 0 & 0\\ 0 & 0 & \times & \times & 0\\ 0 & 0 & 0 & \times & \times\\ 0 & 0 & 0 & 0 & \times\end{bmatrix}\begin{bmatrix}\times & \times & \times & 0 & 0 & 0\\ \times & \times & \times & \times & 0 & 0\\ 0 & \times & \times & \times & \times & 0\\ 0 & 0 & \times & \times & \times & \times\\ 0 & 0 & 0 & \times & \times & \times\end{bmatrix} = \begin{bmatrix}\times & \times & \times & 0 & 0 & 0\\ \times & \times & \times & \times & 0 & 0\\ \times & \times & \times & \times & \times & 0\\ 0 & \times & \times & \times & \times & \times\\ 0 & 0 & \times & \times & \times & \times\\ 0 & 0 & 0 & \times & \times & \times\end{bmatrix} \tag{2.166}$$

here we see that the terms that are non zero, correspond in each subdomain to the subdomain itself plus its neighbours plus the neighbours of neighbours. This

is a general fact that comes from the definitions of Z, G and F

$$(G^T F G)_{ik} = v_b^{(i)^T} B^{(i)^T} \left(\sum_s B^{(s)} S_{bb}^{(s)^{-1}} B^{(s)^T} \right) B^{(k)} v_b^{(k)} \tag{2.167}$$

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 neighbours.

Even though the last term is less sparse that both terms Z and FZ (G and FG) its total size is much smaller than previous blocks since they are square matrices of size N_T .

Optimization of the code

With this structures in mind we can now give the details of several possible optimization to any basic implementation of Algorithm 10. Lets remember that as part of the new orthogonalization process we have to compute the six terms in 2.157, so the improvements come, in one hand, from the fact that some matrix vector computations can be done as block, using the advantages of BLAS3, and on the other hand some of the coefficient matrices computations can be done in parallel.

This 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, lets see how the computation of each term $Z_j^T (F Z_p)$ is done locally. Lets consider a column associated to the subdomain s of Z_p, defined in the complete interface, but taking only local values

$$z_p^{(s)} = B^{(s)} \tilde{f}_{\Gamma^{(sq)}}^{(s)} \tag{2.168}$$

where $\tilde{f}^{(s)}_{\Gamma^{(sq)}}$ is defined in 2.87. Then, each value of $Z_j^T(FZ_p)$ is computed as

$$\begin{aligned}(Z_j^T F Z_p)_{ik} &= z_j^{(i)^T} F z_p^{(k)} \\ &= z_j^{(i)^T} \left(\sum_s F^{(s)} \right) z_p^{(k)} \\ &= \sum_s z_j^{(i)^T} F^{(s)} z_p^{(k)}\end{aligned} \tag{2.169}$$

where, as usual, $F^{(s)} = B^{(s)} S_{bb}^{(s)^{-1}} B^{(s)^T}$ and as seen previously, the values that are non zero are the ones where i-s and s-k are neighbours. This implies that in each subdomain s, we need to do an exchange with every neighbour 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 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 $k \in neighbour(s)$.

Finally each subdomain s will compute the local dot products

$$z_j^{(i)} F^{(s)} z_p^{(k)}, \qquad \forall i,k \in neighbour(s) \cup \{s\} \tag{2.170}$$

then an assembly into $(Z_j^T F Z_p)_{ik}$ plus a global reduction are done to compute the total matrix $Z_j^T(FZ_p)$. This are small sparse matrices stored in every subdomain.

This computation, is done for each $j = 1, \ldots, p-1$ at the p iteration, 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.

In order to improve the time of this calculations what we can do is to take advantage of block implementation, as already done in this chapter. To do so, in every subdomain $\Omega^{(s)}$ we do a pseudoallocation that will concatenate the newly computed and exchanged local vectors $z_p^{(i)}$ into a block of vectors

$$\left[z_1^{(i)}, \ldots, z_{p-1}^{(i)} \right], \qquad \forall i \in neighbour(s) \cup \{s\} \tag{2.171}$$

with this we can compute the terms in Equation 2.170 as a block

$$\left[z_1^{(i)},\ldots,z_{p-1}^{(i)}\right]^T F^{(s)} z_p^{(k)}, \qquad \forall i,k \in neighbour(s)\cup\{s\} \tag{2.172}$$

we can use the optimized libraries BLAS3 to obtain a better performance than usual dot products.
Since every Z_j shares the same structure, a multiple assemblage 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 now done (slightly bigger in size). We then reduce the number of global exchanges from $p-1$ to 1 in each iteration, improving them, and thus the time cost.

- If we look at the following term in the loop to compute Φ_{jp}

$$\sum_{i=1}^{j-1} B_{ij}^T T_i D_p \tag{2.173}$$

we can see that the matrix D_p is independent of the addition, so it can be treated as

$$\left[\sum_{i=1}^{j-1} B_{ij}^T T_i\right] D_p \tag{2.174}$$

so 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

$$V_p \leftarrow \left[\sum_{i=1}^{p-1} B_{ip}^T T_i\right] \tag{2.175}$$

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, including the matrices coming from the Cholesky factorization. Hence, a prori, each process compute the same terms.

  This involves the computation of several small matrix products, that increase in numbers at each iteration. To avoid this 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 cannot forget that a global exchange is added after, so that every process can have access to the new computed Φ_{jp} matrices. The impact of this optimization will be tested in the numerical examples.

2.4 Numerical results

In this section we want to focus in the S-FETI method, by testing the different implementations and changes done to the method that are 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*

$$\begin{cases} -\nu\Delta u &= f \quad in \quad \Omega \\ u &= 0 \quad on \quad \partial\Omega_D \\ \frac{\partial u}{\partial n} &= 0 \quad on \quad \partial\Omega \setminus \Omega_D \end{cases} \tag{2.176}$$

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*

$$\nu \int_\Omega \nabla u \cdot \nabla v dx = \int_\Omega f v, \quad \forall v \in V \tag{2.177}$$

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

$$Kx = f \tag{2.178}$$

2.4.1 FETI and S-FETI

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 [39], but adding some small differences that we will consider to be our basic starting S-FETI algorithm when doing the comparative performance of the different implementations.

In terms of 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 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 ([58],[57]) for the sparse local solutions is used in this type of implementation, but in our case we will change this to the use of a different solver, the one called DISSECTION Solver.

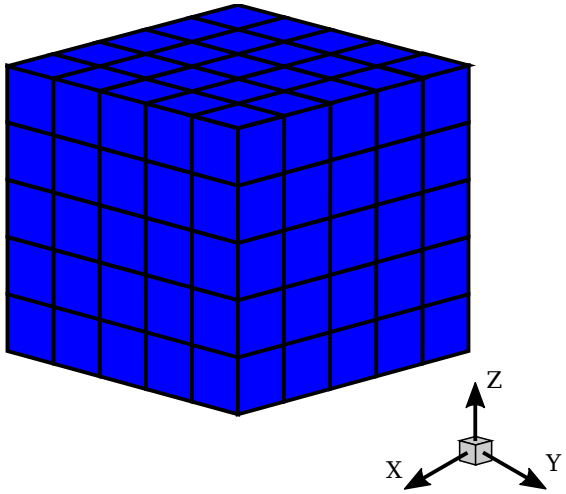

Figure 2.7 – Cube with 125 subdomains.

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].

In this first case, for the 3D Poisson problem, we model the cube $\Omega = [0,1]^3$, with null 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 as a stopping criterion the same as previous chapter, that is the global relative error and the relative solution jump norm across the interfaces. In some cases precision measures will be performed, if that is not the case the criteria will be to stop when both errors are less than 10^{-6}.

We consider the "*Super lumped*" scaling for the projection, i.e $P = P_A = I - AG(G^TAG)^{-1}G^T$ with A as in Equation 2.95, 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 iterations is greatly reduced, but at the cost of adding also the extra search directions. This change tries to reduces the impact of communications between processes which is usually the bottleneck in parallel algorithms.

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

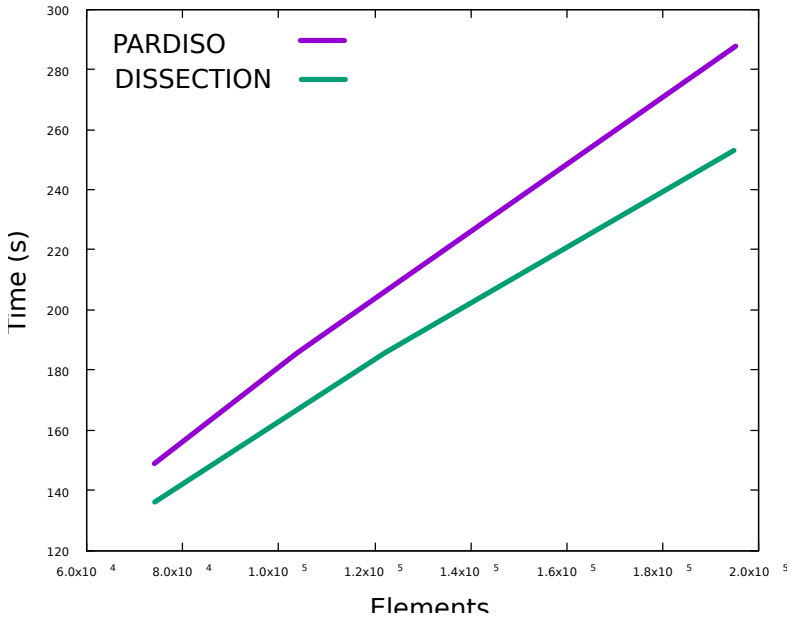

Figure 2.8 – Time of Pardiso vs Dissection for different element number in each subdomain.

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, as 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 test, 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.

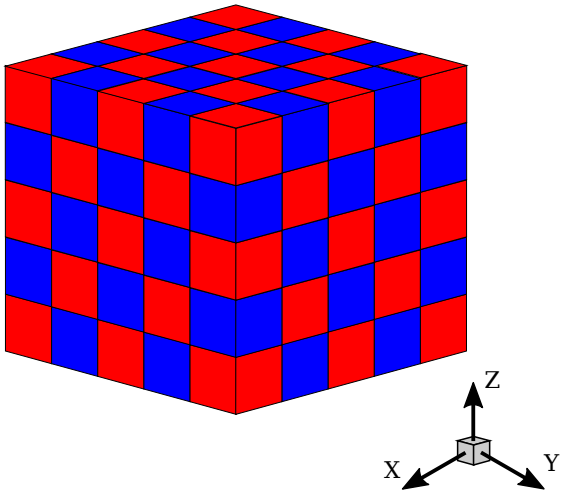

Figure 2.9 – Checkerboard cube with 125 subdomains.

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. This speedup however is at an extra memory cost. From now on, both local interface subdivision and Dissection will be the default for the following S-FETI test.

2.4.2 Decomposition of W^tFW

Following the propositions of section 2.2 in this part we will compare the different strategies to sort the search directions built in S-FETI. We can name both Cholesky and Eigenvalues Decompositions of the matrix W^TFW.

For this problem, we use the same cube as before, with 125 smaller cubes, but we make an 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 zero value parameters. Is this process which introduces new sources of numerical instability, so as a first approach, although very time costly, the use of quadruple precision in the decomposition process may improve the quality of the same and possibly reduce the numerical noise. We test for different values of ε and we check the number of directions kept as

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

well as the total iterations.

In Table 2.4 we see that as the parameter ε is closer to zero, clearly the number of directions saved augments, and 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 show in the minimal error column 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, but since the behaviour of both implementations is the same and the cost of the quadruple computations is much more elevated, 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, meaning values that may be within the limits of just noise, so a value of, for example $\varepsilon = 10^{-4}$ is 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 used to augment the precision 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 implementation will be considered.

One of the advantages searched in changing the decomposition is to make the method less sensitive to parameter variations, in this case, the value of $\varepsilon > 0$ 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 recognizes 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, making this decomposition somehow more robust, as we avoid the noise at the cost of eliminating some other maybe useful directions. In any case this "lost" directions 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 limits the stored directions to the ones associated to the relative eigenvalues that are bigger than this new tolerance ε. This way of sorting directions, is a much more stable algorithm for sorting the directions than the previous Cholesky decomposition.

In the Table 2.5 we show the number of directions kept with each zero value until max precision for each method is achieved.

Here we can see that the bigger values of ε reduces the number of directions kept, but the convergence is still assured, as the important information is kept in the directions associated to the bigger eigenvalues. The eigenvalue decomposition in this case is less sensitive to variations in the zero parameters and in all the cases shows a reduction in the number of directions kept and total iteration number, meaning that at a small cost (in parallel computation it is negligible) we can have a method with less memory charge, less total iterations and improved precision.

We also insist in 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

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.

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, in terms of 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 the previous examples.

We also use 125 subdomains with local interface subdivisions, in order to have at least the same amount of search directions as 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 precision 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 straitghtforward implementation when operating with the sparse blocks and the second one with optimizations in memory allocation to use the BLAS3 routines and reduce the bottleneck of doing several matrix-vector operations instead of just one matrix-matrix product. We

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

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 in terms of memory and time are shown in Table 2.6, where we measure the total time used and also the maximum between the memory used by each process. We can see an speed up of roughly two times for both examples with 100 and 150 thousand elements approximately in each subdomain. We did not added the expected results in terms of number of iterations and precision, where there is no difference between both implementations. Since no precision is lost, plus the gain shown in the memory column and also in the time obtained, makes the second implementation to be the one used from here on.

The objective of the sparse storage implementation is to keep a memory usage controlled in order to be able to use this method for the biggest applications. In this part we will compare the memory usage of the SPARSE-OPT implementation versus the regular FULL one.

In the Figure 2.10 we can see the maximum memory usage between process of both methods for the same problem, but with two different number of total elements in each one. Since we are interested in memory performance we will force to save all the search directions, with no regarding whatsoever of 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

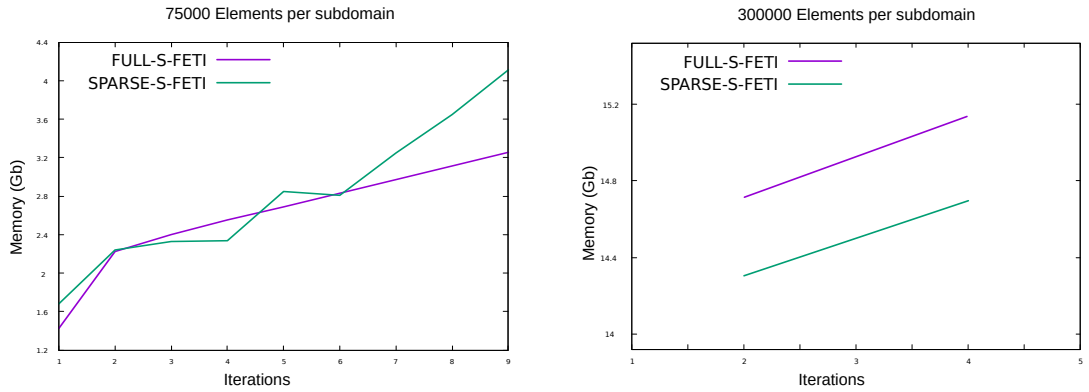

Figure 2.10 – Max local memory usage in cube problem for 75 (left) and 300 (right) thousand elements per subdomain.

of iterations (starting from 2) and the memory usage, or in this case the number of search directions stored, since every vector uses 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, which are, as already explained dense but much smaller than a regular full block of search directions and is also independent of the size of the mesh (the size of the full 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 neighbour connectivity of each subdomain.

This results shows that the difference between the SPARSE storage and the FULL implementation is more clear as we augment the number of elements in each subdomain. In the first case, the FULL case 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 and regular method to see the cost of all the extra computations.

The problem to solve is the same checkerboard configuration as previous with two materials, $\nu_1 = 10^0$ and $\nu_2 = 10^4$. The results are shown in Table 2.7

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

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 goes down whenever the problem gets bigger. 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 difference between the two existent preconditioners for S-FETI, the Dirichlet and Lumped. We test for memory, number of iterations and total time for the computations.

Added to this, we want to compare the sparse and full storage implementations with this preconditioners, as well as the two decomposition for sorting the search direction. The idea is to have global view that can gives some notion 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 really 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 ([47]). This type of 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 see the difference in term of number of local solves and total search directions, we can "eliminate" this corners and test the method in a more realistic way, with non of the search directions created by this corner interfaces. In our particular case, we lower the number of total search direction per iteration from 2072 to 600.

One if the crucial difference that we want to test is the one 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 Equation 2.57, Equation 2.58, and knowing that a multiplication by the Schur complement implies the resolution of a local problem, we can name two of the big differences between them, this is, the computational and memory cost, where for both characteristics the Dirichlet is much more expensive, however we expect a reduced number of iteration for this preconditioner, since it consider the variation of the complete subdomain and not only of the nodes in the interface.

The test case will be the exact 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, as already said, we will measure the time and memory performance of them. Each result will be a different combination of four variables

In the Table 2.8 we show the results obtained, we note that due to the characteristic of the problem, meaning a number of subdomain and size of the mesh not high enough, we obtain results that does not go with the lines of what to expect, as we see a reduction in time for the SPARSE-OPT, 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 this results closer to a real implementation with automatic subdomain partitioning. In terms of memory usage, there is no big difference between

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.

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.

methods, since the size of the problem is not big enough to make them clearer, as was the case presented before in this section. In any case we still recall that 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 basic S-FETI. We added a new way of sorting search directions, the Eigenvalue decomposition can show to be a more robust option that reduces the number of directions stored in each iteration. In terms of memory usage we developed a new sparse storage that allows to use this method in bigger applications without losing significant performance. The use of the Dissection solver with his kernel detection algorithm are a good alternative for the largely used PARDISO solver. The Lumped preconditioner for this method, even if the results shown here are not definitive to say in which cases should be used, is still a good alternative to reduce memory and computation efforts, 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 for this method.

Chapter 3

Block strategies as a preconditioner

As a continuity on the development of the domain decomposition methods, in particular the FETI method, always searching for a fast and accurate algorithm that can be used in as much problems as possible, we will try now a different approach to the improvements already done to the S-FETI, using some of the ideas exposed in the previous chapter.

The main advantage shown by the S-FETI method is the construction of an enlarged search space in the Conjugate Gradient algorithm where the capture of the largest eigenvalues of the FETI operator, even for some hard problems, can be achieved in very few iterations. The price to paid 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 mimic 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 this idea that mixes it with the S-FETI method to end up with numerical results of

both new methods.

3.1 Introduction and preliminaries

The FETI method is based in finding the solution of an interface problem using the Preconditioned Conjugate Gradient Method. This iterative method (CG) has been largely studied over the years, leading to different results in its research from which several generalizations were made to it.

Within this generalizations we can name the Flexible Conjugate Gradient, the Multipreconditioned CG and the Method of Conjugate Directions. Although the last one was developed before the CG method it can still be considered as one of the generalization of it.

We will now show in detail, two of this methods and their relation of some of them with previous FETI methods. This relations will allow to understand the roots of the new FETI methods proposed in this chapter and justify its convergence.

3.1.1 Method of Conjugate Directions

We consider the solution of the general system

$$F\lambda = d \tag{3.1}$$

with F symmetric positive definite.

This method, shown in [45] can be considered as a more general method than the CG method.

In this method the search directions

$$w_1, w_2, \dots \tag{3.2}$$

are selected to be mutually conjugate, that is

$$(w_i, Fw_j) = 0, \qquad \forall i \neq j \tag{3.3}$$

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 resume of the algorithm is done in 11. We can see that the CD-method

Algorithm 11 Method of Conjugate Directions

1: **Initialization**
2: Choose an approximation vector λ_0
3: $g_0 = F\lambda_0 - d$
4: Choose any non-zero vector w_0
5: **loop** $p = 0, 1, 2, ...$ 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: $\quad$ Choose a direction w_{p+1} such that $(w_{p+1}, Fw_j) = 0$, $\forall j = 0, \ldots, p$
10: **end loop**

is not precise, in the sense that no formula is given for choosing the search directions. Depending on how to build this directions, different methods can be obtained (including the CG method). Later we will 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*

$$(w_i, Fw_j) = 0, \qquad i \neq j \tag{3.4}$$

$$(w_i, g_p) = 0, \qquad i = 0, 1, \ldots p-1 \tag{3.5}$$

$$(w_p, g_0) = (w_p, g_1) = \cdots = (w_p, g_p) \tag{3.6}$$

The scalar ρ_p can be given by the formula

$$\rho_p = -\frac{(w_p, g_0)}{(w_p, Fw_p)} \tag{3.7}$$

Proof. From the last line in 11 we have directly 3.4. Next, using the update of the gradient, we have that

$$(w_i, g_{p+1}) = (w_i, g_p) + \rho_p (w_i, F w_p) \tag{3.8}$$

if $i = p$ then, by the definition of ρ_p we know that

$$(w_i, g_{p+1}) = 0 \tag{3.9}$$

Moreover, from Equation 3.4 we have

$$(w_i, g_{p+1}) = (w_i, g_p) \tag{3.10}$$

Equations 3.5 and 3.6 follow from the last two relations. The last equation 3.7 follows from 3.6 and 3.4. □

As for the convergence of the method we have the following theorem

Theorem 3.2. *The CD-method is an 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

$$y_0 = \lambda - \lambda_0 \in span\{w_0, \ldots, w_{p-1}\} \tag{3.11}$$

Clearly $m \leq n$ since the vectors are $w_0, w_1, \ldots$ are linearly independent. On the other hand, we may choose scalar $\alpha_0, \ldots, \alpha_{p-1}$ such that

$$y_0 = \alpha_0 w_0 + \cdots + \alpha_{p-1} w_{p-1} \tag{3.12}$$

Hence,

$$\lambda = \lambda_0 + \alpha_0 w_0 + \cdots + \alpha_{p-1} w_{p-1} \tag{3.13}$$

Moreover

$$g_0 = F\lambda_0 - d = F(\lambda_0 - \lambda) = -\alpha_0 F w_0 - \cdots - \alpha_{p-1} F w_{p-1} \tag{3.14}$$

Computing the inner product

$$\begin{aligned}(w_i, g_0) &= -\alpha_0(w_i, Fw_0) - \cdots - \alpha_{p-1}(w_i, Fw_{p-1}) \\ &= -\alpha_i(w_i, Fw_i)\end{aligned} \tag{3.15}$$

an using 3.7 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*

$$f(v) = \|\lambda - v\|_F = (\lambda - v, F(\lambda - v)) = (v, Fv) - 2(v, d) + (\lambda, d) \tag{3.16}$$

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

$$\lambda_0 + span\{w_0, \ldots, w_{p-1}\} \tag{3.17}$$

and this space contains the vectors, $\lambda_0, \ldots, \lambda_p$

Proof. If w is a search direction, then we have

$$f(v + \alpha w) = f(v) - 2\alpha(w, g) + \alpha^2(v, Fv) \tag{3.18}$$

where

$$g = d - Fv = F(\lambda - v) \tag{3.19}$$

Considered as a function of α the function $f(v + \alpha w)$ has a minimum value at $\alpha = \rho$, where

$$\rho = \frac{(w, g)}{(w, Fw)} \tag{3.20}$$

Comparing this, with the definition of ρ_p in the algorithm, we have the minimization result.

For the last part, lets consider the point $v \in \lambda + span\{w_O, \ldots, w_{p-1}\}$, then we have

$$v = \lambda_0 + \alpha_0 w_0 + \cdots + \alpha_{p-1} w_{p-1} \tag{3.21}$$

applying f

$$f(v) = f(\lambda_0) - \sum_{i=0}^{p-1} [2\alpha_i(w_i, g_0) - \alpha_i^2(w_i, Fw_i)] \tag{3.22}$$

At its minimum we have

$$\alpha_i = \frac{(w_i, g_0)}{(w_i, Fw_i)} \tag{3.23}$$

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 name the General CG or Flexible CG (as denoted in [55]). We consider again 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 accelerate the convergence by using suitable search directions, usually this is achieve by finding the solution of a problem of the type

$$Bx = y \tag{3.24}$$

so that the conditioning of the matrix $B^{-1}F$ is smaller than the conditioning of F.

In PCG the preconditioner B^{-1} is symmetric positive-definite and fixed, in order to keep the convergence properties of CG. In the Flexible-CG we relax this conditions and change the preconditioning step to a general mapping

$$g \to \mathcal{B}[g] \tag{3.25}$$

this mapping also tries to approximate the inverse of F in order to improve the converge. This more general mapping does not need to have and explicit form as it can be a recursive non-linear mapping. We will simply 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 12. This

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

algorithm explicits the construction of the F-conjugate search directions by introducing a preconditioning procedure and applying the Modified Gram-Schmidt procedure in order 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 $\mathcal{B}$, were conditions on coercivity and boundness are assumed [7]. In our case, and in order 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 [39], this is because the fact that several directions are created in the preconditioning step, thanks to the additive nature of the Dirichlet preconditioner in FETI. 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 possibles). In the FETI framework there are no practical problems in the use of this extra computations, since we are forced to do them in order to keep a good convergence (See [66]).

Following this idea, and since we already do a full reorthogonalization, we can consider the general method of Conjugate Directions which give us 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 in consider a sort of mixing of this methods, where we consider some special preconditioner that changes from iteration to iteration, but it is a much better preconditioner than the usual Dirichlet in the FETI method, since it is built straightforward from the one in S-FETI where clearly a better 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

$$w_p = \tilde{F}^{-1} g_p \tag{3.26}$$

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 in a direct way, 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 of the minimization process of the CG method applied in the S-FETI method, where we construct a bigger search space, based in the combination of local terms. Lets 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, for 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

$$w_p = W_p\rho_p = -W_p(W_p^T g_p) \tag{3.27}$$

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 problem (if 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

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** $\mathcal{B}[g] \leftarrow w$

the same as in chapter 2 so $D^{(s)^{-1}}_{\Gamma^{(sq)}}$ is the local preconditioner partitioned into neighbour 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 certain 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 does not come from a direct multiplication for 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 to define $\mathcal{B}[\cdot]$ 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, by only storing one direction per iteration and at the same time diminishing 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

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**
        $\gamma_i = -\dfrac{(w_i, Fw_{p+1})}{(w_i, Fw_i)}$
        $w_{p+1} = w_{p+1} + \gamma_i w_i$
        $Fw_{p+1} = Fw_{p+1} + \gamma_i F w_i$
    end for
end loop

capture the largest eigenvalues, and we expect this will be enough to compensate for the extra cost on the mapping of the preconditioner.

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 in the preconditioning step in FETI versus the preconditioning procedure defined in Algorithm 14. We note that this is also because the procedure $\mathcal{B}[\cdot]$ will also give the vector Fw_p at the end of the procedure since its value correspond to $FW\rho$, hence no extra multiplication by F are needed after this part.

Based on the analysis of the cost in S-FETI we can say that the most expensive part here will also be the computation of the block FW. This includes a projection and several forward-backward computations, but we can use the same strategies to minimizes the computations as done in S-FETI, meaning that for each iteration

we only add to a process the workload of $\mathcal{N}$ Neumann solves, with $\mathcal{N}$ being either $neighbours + 1$ or $2 \times neighbours$ 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 the section 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 is at the same time have a few extra cost compared to FETI) plus an iteration of FETI without the Neumann solution, but since we can not estimate the number of iteration needed for convergence (hopefully much less than FETI), we will analyse in the numerical examples the real performance of this method and 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, where 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. This number will be a limited number of vectors between 1 and the total number of linear independent search directions in the iteration N_S.

In order to achieve this, we will proceed in a similar way 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, since 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

$$w_p = W_p \rho = \left[\begin{array}{c|c|c} & & \\ W_p^1 & \dots & W_p^{N_S} \\ & & \end{array}\right] \begin{bmatrix} \rho_1 \\ \vdots \\ \rho_{N_S} \end{bmatrix} = \sum_i \rho_i W_p^i \tag{3.28}$$

with ρ the vector of minimization parameters $\rho = -W_p g_p$. This time we split this sum and we will recover a new block of search directions, smaller than the previous one, and to try and not to lose information we add an extra direction either the vector $W_p \rho$, this implies the construction of yet another block $\tilde{W}_p$ defined by

$$\tilde{W}_p = \left[\begin{array}{c|c|c|c} & & & \\ W_p \rho & \rho_1 W_p^1 & \dots & \rho_M W_p^M \\ & & & \end{array}\right] \tag{3.29}$$

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.

As previous method, the definition of the FETI projector P is as usual. In Algorithm 17 we described the final algorithm, build 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 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 on 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, since we do the same construction to obtain the block of search directions. The only difference is the size of the stored vector 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.

3.3 Numerical results

Since the methods shown in this chapter can be considered as variations of the S-FETI algorithm, the characteristics of them in terms of problems that can or cannot solve are very similar to the original S-FETI. Is for this reason that a simple test case can and will be considered next, in order to make this first comparative between this methods.

The Poisson problem, or any other simple case, solved in 3D will allow us to do this first comparative of both method 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*

$$\nu \int_\Omega \nabla u \cdot \nabla v dx = \int_\Omega f v, \quad \forall v \in V \tag{3.30}$$

where

$$V := \left\{ v \in H^1(\Omega) : v|_{\partial\Omega_D} = 0 \right\} \tag{3.31}$$

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 criterion is 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 use as the FETI-1LM original, so no test related to memory will be performed.

Our first and main approach will be to asset the number of iterations needed to converge for 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.

In terms of iteration numbers we can see a reduction in the total number of them, compared to FETI-1LM, but no versus 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 previous method, so in this case, we change the reduction of the search directions from a single vector to a small number of vectors. Since this vectors are computed in descending order of importance, we want to keep a minimal number of search directions without loss of convergence.

We continue to use the same test case, that is, the Poisson problem in the cube with 125 subdomains and fixed 75 thousand elements per subdomain. This

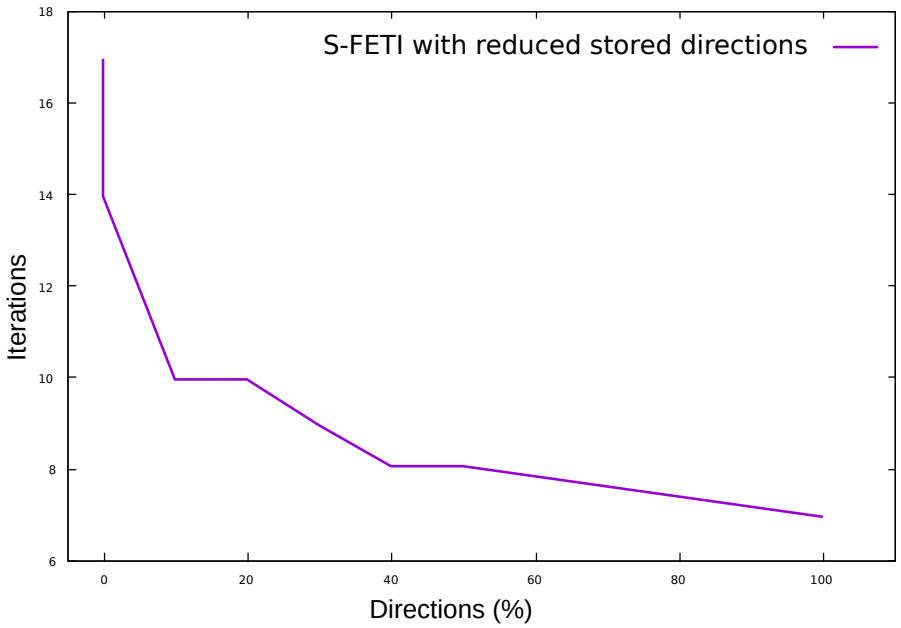

Figure 3.1 – Iterations versus percentage of computed directions. Example 1

is complex enough, mainly due to the number if directions produced in each iterations, so we can expose the differences between this method and the original S-FETI and FETI.

In the definition of this method, we have to choose a fixed number of search directions to keep at every iterations. 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 in the method, so we need to do variation of it to try in find an "optimal" value if possible.

In Figure 3.1 we show the behaviour of the method in terms of total iterations versus the percentage of directions stored. Lets recall that the total number of search directions generated, without consider linear independence is 2072, from which we do a first reduction in number, to later keep a part of this reduced number.

In this image we see the behaviour of the method, with an initial fast 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 for the

method. 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 shows that in the actual conditions of the first variation, even if we obtain and improvement in terms of iterations to convergence, precision and memory, there is no gain total time due to the too much expensive cost of the implementations 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 find important to state this results, to avoid the same work in them and at the same time, it can still be source of new development for the S-FETI and other FETI like domain decomposition methods.

Chapter 4

FETI-2LM with enlarged search space

In this final chapter we will now show the extension of the main idea in the S-FETI method, that is, enriching the search space of the FETI algorithm, but this time to one of the other FETI method shown in this thesis, in particular the FETI-2LM method.

The FETI-1LM and FETI-2LM methods share similarities in the iterative algorithm used, both of them are Krylov based, the Conjugate Gradient and the ORTHODIR method. This fact makes the idea of enriching the search space in FETI-2LM, a priori, a good one, but big part of the convergence properties of S-FETI comes from splitting the preconditioner to isolate the modes that make the convergence slower, meaning that the fast convergence can be expected, also because the relation of this method with the FETI-Geneo method which has a proven fast convergence speed.

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 speed up as in S-FETI, but we will have in any case an large improvement in the convergence, due to the constructions of directions using local properties that tend to get lost in the FETI-2LM method, but also we will gain from the size of the new enriched space compared to the original one. Finally in terms of time spend, we will be applying the same implementation strategies as in S-FETI, hence a time speedup is most assured.

We start by recalling the FETI-2LM method, to then pass directly to the

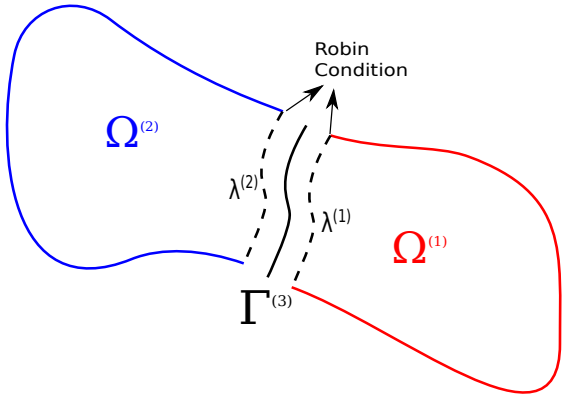

Figure 4.1 – Two subdomains with Robin Interface Condition

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, but in any case we want to recapitulate some of the more important ones, in order to keep clarity in 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 [35], 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, as the general formulation comes from using the two-subdomain case in each pair of neighbour subdomains.

Consider the linear problem, arising from a Finite Element discretization of a PDE

$$Kx = f \tag{4.1}$$

We divide the global problem in two subdomains $\Omega^{(1)}$, $\Omega^{(2)}$ and their interface $\Gamma^{(3)}$. Then the global problem has the structure

$$\begin{bmatrix} K_{ii}^{(1)} & 0 & K_{ib}^{(1)} \\ 0 & K_{ii}^{(2)} & K_{ib}^{(2)} \\ K_{bi}^{(1)} & K_{bi}^{(2)} & K_{bb}^{(1)} + K_{bb}^{(2)} \end{bmatrix} \begin{bmatrix} x_i^{(1)} \\ x_i^{(2)} \\ x_b \end{bmatrix} = \begin{bmatrix} f_i^{(1)} \\ f_i^{(2)} \\ f_b^{(1)} + f_b^{(2)} \end{bmatrix} \tag{4.2}$$

and the subdomain stiffness matrices and right hand sides are:

$$K^{(1)} = \begin{bmatrix} K_{ii}^{(1)} & K_{ib}^{(1)} \\ K_{bi}^{(1)} & K_{bb}^{(1)} \end{bmatrix}, f^{(1)} = \begin{bmatrix} f_i^{(1)} \\ f_b^{(1)} \end{bmatrix} \qquad K^{(2)} = \begin{bmatrix} K_{ii}^{(2)} & K_{ib}^{(2)} \\ K_{bi}^{(2)} & K_{bb}^{(2)} \end{bmatrix}, f^{(2)} = \begin{bmatrix} f_i^{(2)} \\ f_b^{(2)} \end{bmatrix} \tag{4.3}$$

The FETI-2LM method comes from the imposition of independent generalized Robin condition on the interface $\Gamma^{(3)}$.

The discrete local problem are then

$$\begin{bmatrix} K_{ii}^{(s)} & K_{ib}^{(s)} \\ K_{bi}^{(s)} & K_{bb}^{(s)} + A_{bb}^{(s)} \end{bmatrix} \begin{bmatrix} x_i^{(s)} \\ x_b^{(s)} \end{bmatrix} = \begin{bmatrix} f_i^{(s)} \\ f_b^{(s)} + \lambda_b^{(s)} \end{bmatrix}, \qquad s = 1, 2 \tag{4.4}$$

The augmentation matrix $A_{bb}^{(s)}$ will be considered as an sparse approximation of the neighbour Schur complement, as seen in subsection 1.2.3.

In order to be the restrictions of the global problem, each local solution must satisfy two conditions, first the continuity condition:

$$x_b^{(1)} - x_b^{(2)} = 0 \tag{4.5}$$

The second condition is the interface equilibrium, which is the last line in the global block formulation

$$K_{bi}^{(1)} x_i^{(1)} + K_{bi}^{(2)} x_i^{(2)} + (K_{bb}^{(1)} + K_{bb}^{(2)}) x_b = f_b^{(1)} + f_b^{(2)} \tag{4.6}$$

if we use the continuity condition, we have

$$K_{bi}^{(1)} x_i^{(1)} + K_{bi}^{(2)} x_i^{(2)} + K_{bb}^{(1)} x_b^{(1)} + K_{bb}^{(2)} x_b^{(2)} = f_b^{(1)} + f_b^{(2)} \tag{4.7}$$

Now using the last line of the local problems, we can write the equilibrium condition as

$$A_{bb}^{(1)} x_b^{(1)} + A_{bb}^{(2)} x_b^{(2)} = \lambda_b^{(1)} + \lambda_b^{(2)} \tag{4.8}$$

we combine both continuity and equilibrium conditions to obtain the interface mixed equations

$$\begin{aligned} A_{bb}^{(1)} x_b^{(2)} + A_{bb}^{(2)} x_b^{(2)} &= \lambda_b^{(1)} + \lambda_b^{(2)} \\ A_{bb}^{(1)} x_b^{(1)} + A_{bb}^{(2)} x_b^{(1)} &= \lambda_b^{(1)} + \lambda_b^{(2)} \end{aligned} \tag{4.9}$$

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 $\lambda_b^{(s)}$

$$(K_{bb}^{(s)} - K_{bi}^{(s)} K_{ii}^{(s)^{-1}} K_{ib}^{(s)} + A_{bb}^{(s)}) x_b^{(s)} = \lambda_b^{(s)} + f_b^{(s)} - K_{bi}^{(s)} K_{ii}^{(s)^{-1}} f_i^{(s)} \tag{4.10}$$

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 $x_b^{(s)}$ by their values as functions of $\lambda_b^{(s)}$ coming from previous explicit relation into the mixed equations 4.9, we can write the condensed interface problem of the form $F\lambda = d$ associated to the FETI-2LM method

$$\begin{bmatrix} I & I - (A_{bb}^{(1)} + A_{bb}^{(2)})(S_{bb}^{(2)} + A_{bb}^{(2)})^{-1} \\ I - (A_{bb}^{(1)} + A_{bb}^{(2)})(S_{bb}^{(1)} + A_{bb}^{(1)})^{-1} & I \end{bmatrix} \begin{bmatrix} \lambda_b^{(1)} \\ \lambda_b^{(2)} \end{bmatrix} = \begin{bmatrix} (A_{bb}^{(1)} + A_{bb}^{(2)})(S_{bb}^{(2)} + A_{bb}^{(2)})^{-1} c_b^{(2)} \\ (A_{bb}^{(1)} + A_{bb}^{(2)})(S_{bb}^{(1)} + A_{bb}^{(1)})^{-1} c_b^{(1)} \end{bmatrix} \tag{4.11}$$

The definition and implementation of this method for general problems, comes from the computation of this operator between every two neighbour 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 operators is dense so its assembling is computationally inefficient. On the contrary, in order to multiply some vector by this operator (denoted F as usual FETI methods)

we only need local data and exchanges between neighbours, hence an iterative method such as ORTHODIR is more suited.

We present in 18 the ORTHODIR algorithm 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 using the decomposition of the gradient to build the block of search directions used to update the solution in every iteration.

4.2 The Block-2LM Algorithm

Following the work done 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, with the difference that the gradient this time will no longer be preconditioned,

but it can also be considered as a sum of local gradients that when separated will improve the mimic of the local subdomains behaviour. In any case, since in FETI-2LM we have two independent lagrange multipliers instead of one in each the interface, we can create as much directions as the S-FETI method in both of its version (subdomain or interface divisions).

Following the notation of chapter 2, the gradient $g \in \mathbb{R}^n$ in the FETI-2LM method is

$$g = \begin{bmatrix} g^{(1)} \\ \vdots \\ g^{(N_s)} \end{bmatrix} \tag{4.12}$$

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 $\Omega^{(s)}$, this local gradients are independent from each other since there are two lagrange multipliers on the interface. This subdivision is enough to form a new block of directions, all by considering the fact that g can be written as

$$g = \sum_{s=1}^{N_s} \bar{g}^{(s)} \tag{4.13}$$

where $\bar{g}^{(s)} \in \mathbb{R}^n$ is the extension by zero of each local vector, i.e.

$$\bar{g}^{(s)} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ g^{(s)} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{4.14}$$

with this division, our block of search directions will be defined by

$$Z = \left[\bar{g}^{(1)}, \bar{g}^{(2)} \ldots, \bar{g}^{(N_s)}\right] \tag{4.15}$$

Which is similar to the subdomain division in S-FETI.

In an analogous way, to form the interface subdivision, we can consider at the local level that $g^{(s)}$ can also be decomposed in smaller vectors, one for each of the subdomain neighbours

$$g^{(s)} = \begin{bmatrix} g_{loc}^{(s),1} \\ \vdots \\ g_{loc}^{(s),n^{(s)}} \end{bmatrix} \tag{4.16}$$

with $n^{(s)}$ the number of neighbours of subdomain $\Omega^{(s)}$. Hence, the general gradient can also be written as

$$g = \sum_{s=1}^{N_s} \sum_{i=1}^{n^{(s)}} g^{(s),i} \tag{4.17}$$

where $g^{(s),i} \in \mathbb{R}^n$ is again an extension by zero, defined as

$$g^{(s),i} := \begin{bmatrix} 0 \\ \vdots \\ 0 \\ g_{loc}^{(s),i} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad s = 1,\ldots,N_s, \quad i = 1,\ldots,n^{(s)} \tag{4.18}$$

From the equation 4.17, we can use the additive form of the gradient in order to define the new search space for the Block-2LM method. This space will be spanned at each iteration by the column vectors in

$$Z = \left[g^{(1),1},\ldots,g^{(1),n^{(1)}}, \quad \ldots, \quad g^{(N_s),1},\ldots,g^{(N_s),n^{(N_s)}} \right] \tag{4.19}$$

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 $F^T F$-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

$$Span\{g^{(1)} + \cdots + g^{(m)}\} \subseteq Span\{g^{(1)}, \ldots, g^{(m)}\} \tag{4.20}$$

for any vectors $g^{(i)} \in \mathbb{R}^m$, $m \in \mathbb{N}$, $i = 1, \ldots, m$.

Since the norm used in this method changes, from F to $F^T F$-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

$$W_p = Z_p + \sum_{i=0}^{p} W_i \Phi_i \tag{4.21}$$

The same linear dependency issue occurs for this matrix as in the S-FETI 2.2. Lets recall that this is due to the phenomenon of work within the computer precision or because we achieve convergence in some of the interfaces before the other. This dependency between columns in Z leads 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 directions to be used (The Eigenvalues decomposition 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 are similar as the one from FETI and S-FETI, we can name the augmentation on the size of the information shared between neighbours, but with no incremental number of exchanges, and we also have the pseudo-inversion of the matrix $(FW)^T(FW)$, both of this changes are fairly cheap ones.

The most expensive part is again the multiplication by the FETI operator, this can be done with a simple straightforward implementation that solves a local

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

problem for each column in Z, this can only serve for academical purposes, to know the precision and convergence behaviour of it, but if we want a practical method we need again to exploit the sparse structure of the search directions matrix Z.

We know from the definition that the columns $g^{(s),i}$ of Z are non null only on the interface of subdomain $\Omega^{(s)}$ and the multiplication by the FETI operator only requires, at the subdomain level, the solution of a number of local problems equal to the number of neighbours, which can be done in a block implementation. With this consideration we can have the speed up to make this method useful in real problems, since the time spend during the resolutions we hope to recover it by reducing the iterations needed for convergence.

To understand the implementation of this method, we will again use the so called *Coarse Modes*, that correspond to the non-null vectors in each column of Z or FZ. This modes are stored locally in each subdomain, in this case we have one mode for each neighbour, this correspond to each of the vectors $g_{loc}^{(s),i}$ defined in Equation 4.16, but at the same time we shared them with every neighbour, 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, where in dotted lines are the ones own by a subdomain, also we see the subdomains involved in the computing of FZ, and in the right, the dotted modes are the ones owned by the red interface, at the same time this modes are non null modes of FZ in the red interface, this comes from the definition of the operator.

To better understand this, lets look at the definition of the operator F for a single interface

$$F\begin{bmatrix}\lambda_1^{(s)}\\ \lambda_1^{(q)}\end{bmatrix} = \begin{bmatrix}\lambda_1^{(s)} + \lambda_1^{(q)} - (A^{(s)} + A^{(q)})(S^{(q)} + A^{(q)})^{-1}\lambda_1^{(q)}\\ \lambda_1^{(s)} + \lambda_1^{(q)} - (A^{(s)} + A^{(q)})(S^{(s)} + A^{(s)})^{-1}\lambda_1^{(s)}\end{bmatrix} \tag{4.22}$$

this implies that for the case of the Block-2LM, at the interface level, we have to

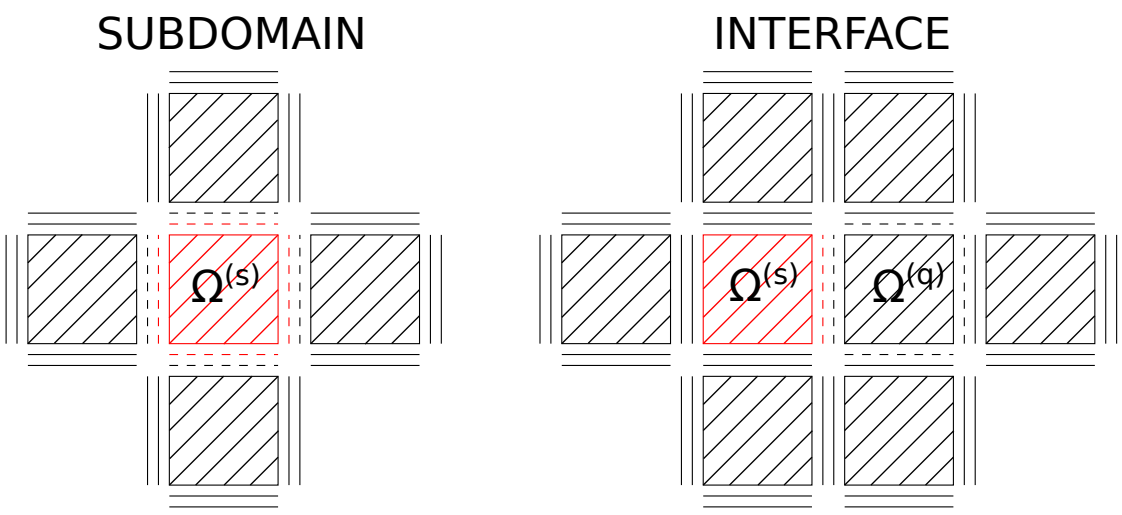

Figure 4.2 –
Left: In red the coarse modes describing Z owned by subdomain $\Omega^{(s)}$. In dotted lines, the modes shared between $\Omega^{(s)}$ and the subdomains involved in multiplying by the FETI-2LM operator.
Right: In dotted lines the coarse modes describing FZ owned by local interface of $\Omega^{(s)}$ and conversely they are the non null modes in the interface in red.

compute

$$\begin{aligned} F\begin{bmatrix} \lambda_1^{(s)} \\ 0 \end{bmatrix} &= \begin{bmatrix} \lambda_1^{(s)} \\ \lambda_1^{(s)} - (A^{(s)} + A^{(q)})(S^{(s)} + A^{(s)})^{-1}\lambda_1^{(s)} \end{bmatrix} \\ F\begin{bmatrix} 0 \\ \lambda_1^{(q)} \end{bmatrix} &= \begin{bmatrix} \lambda_1^{(q)} - (A^{(s)} + A^{(q)})(S^{(q)} + A^{(q)})^{-1}\lambda_1^{(q)} \\ \lambda_1^{(q)} \end{bmatrix} \end{aligned} \tag{4.23}$$

We note that each subdomain solves one local problem for each local interface (number of neighbours). All the local solutions plus the multiplication by local augmentation matrix are then send together to each of the neighbours, meaning we send the vectors

$$\begin{aligned} (S^{(s)} + A^{(s)})^{-1}\lambda_1^{(s)} \\ A^{(s)}(S^{(s)} + A^{(s)})^{-1}\lambda_1^{(s)} \end{aligned} \tag{4.24}$$

this for every local computation performed, allowing to multiply this by the local augmentation matrix associated to the interface and in general compute

and store a local contribution to FZ. With this, each local interface stores the contributions to FZ done by the neighbour subdomain.

Finally, in terms of memory use, the same issue described in chapter 2 is expected with this implementation, but we can again use the same strategy to store the sparse directions and reconstruct them later, although we will leave that as a future work.

Remark: We want to point out some of the differences between this method and S-FETI, mainly the fact that no projection is needed in this case, so no extra cost is added in this part and also the fact that the connectivity is more limited than in the S-FETI method, thanks to the FETI-2LM operator, hence we need less modes stored in each interface.

4.4 Numerical results

In this section, we will be performing another basic comparative testing, that will include, mainly the FETI-2LM and Block-2LM methods. Classical comparison between iteration number, as well as a full fortran-mpi implementation to measure for time performance will be analyzed.

To perform this benchmark, we do no need for a special complex case, as one of the main properties of both method 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 neighbour subdomains, meaning that an 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

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

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 criterion, 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 the tables 4.1 and 4.2 we can see a general comparative of both methods. We have set the values for the parameter $patch\_size = 2$ and $patch\_depth = 3$, see Algorithm 7, also the number of elements per subdomain is change, 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.

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 spend 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.

This effects is less present in the S-FETI methods, because the computation in this part can be reduced by 2 since the values of the Lagrange multiplier are shared by neighbour subdomains, but in this case 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 produces the notorious slowdown.

We consider this implementation as a first step before doing the sparse storage implementation, 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 neighbour 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

$$Z = \left[\ldots, g^{(s),1}, \ldots, g^{(s),n^{(s)}}, \ldots\right], \qquad s = 1, \ldots, N_s \tag{4.25}$$

and when doubling the direction we will have

$$Z = \left[\ldots, g^{(s),1}, g^{vec\{(s),1\}}, \ldots, g^{(s),n^{(s)}}, g^{vec\{(s),n^{(s)}\}}, \ldots\right], \qquad s = 1, \ldots, N_s \tag{4.26}$$

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 neighbour subdomain of $\Omega^{(s)}$ through his local interface i.

4.5 Conclusion

A new Block method as been developed for the FETI-2LM following the ideas used in S-FETI. Although the properties of FETI-2LM, such as the lack of a 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 fact of enlarging the search space, still

predicts a better convergence in terms of iterations for this method.

The numerical results validate the reduction of 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 spend in the full reconjugation as a factor controlled. Making this factor the major slowdown for the Block-2LM method.

Alternatives to the reconjugation must be found, mainly the sparse storage proposed in chapter 2, but we leave that as a future work.

Conclusion and perspectives

Throughout the work done in the four chapters exposed in this thesis, we have detailed several improvement to existing classic FETI method found in the literature that are currently used in real life applications that comes from the industry, namely structure, electromagnetism models among others.

The first part consisted in the development of the Hybrid-FETI method, born from the mixing of the FETI-1LM and FETI-2LM methods. This methods consist in the imposition of either Neumann or Robin conditions, as done in FETI-1LM and FETI-2LM respectively, in the local interfaces as 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 a bad conditions beyond the heterogeneities. This is the case of contact problems, where the contact area presents an ill-conditioning which the FETI-1LM even with the consistent version of the Dirichlet method does not assure the convergence. This problems are usually solved using the FETI-2LM due to the formulation with two independent lagrange multipliers provided by this method, that handles the issue of the contact area. Since the use of the FETI-2LM is only required for the contact area, we can improve the global convergence by treating the non contact interfaces with the regular and faster FETI method. This global liberty of choice for the interfaces is our 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 this 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 this subdomains the augmentation matrix present in the local problem is added to the Schur complement usually computed in the Dirichlet preconditioner.

Although we see and 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 actually applied or in general a new preconditioner for the global operator of the Hybrid-FETI.

Looking at numerical results, the best one 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. This 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 his first version presented by [17] to his more precise general formulation, shown in [39]. The development of the consistent Dirichlet preconditioner is also exposed, as this is a key 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, that are present in almost all problems. A good recognition of them can have a large impact in the final results. The Eigenvalue decomposition is proposed as an alternative to the basic Cholesky factorization that uses the basic S-FETI, the idea of this variation is to keep the good behaviour but using less directions in each iteration.

Confirmed by the numerical results we can see that the same convergence

can be achieved, but using less directions in each iteration. This reduction comes from the fact that the Eigenvalue decomposition can order 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 lost of information at each iteration.

One of the goals of the Eigenvalue decomposition is try 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 solve 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 neighbours. Using this information we can reconstruct the directions from a series of smaller coefficient matrices and at the same time we can reduce the multiplication of the sparse blocks by the FETI operator to faster matrix-matrix computations.

In the numerical results we show that the impact in total time done 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 be also improved. We leave that as future work.

The third chapter is dedicated to test different ideas regarding variations of the S-FETI, we can name the two main ones, which consist on one side of using the block of search directions build by S-FETI as a preconditioner for the original FETI method, and the second idea is to add to the search direction created in the first method a small part of the directions associated to the biggest eigenvalues when doing the decomposition of W^tFW.

Both of the method show an improvement in term 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 an future development of methods derived from S-FETI. The second one, needs further tests to asset his value.

Finally, the Block-2LM method was developed as an extension of FETI-2LM, using the same ideas that lead to the S-FETI method. The construction of a bigger search space containing the original Krylov space assures convergence. 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 important as the one seen in S-FETI and also with no reduction in the total time spend, however this method is considered as a first step, since an sparse storage version is also needed in this case, knowing that the reduction achieved is not small enough and that we will want to solve some big cases presented in problems arising from electromagnetism, as the ones seen in [8],[64].

Bibliography

[1] 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.

[2] J. Alberty et al. "Matlab Implementation of the Finite Element Method in Elasticity". In: *Computing* 69.3 (2002), pp. 239–263.

[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 et al. "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 et al. "Collective communication: theory, practice, and experience". In: *Concurrency and Computation: Practice and Experience* 19.13 (2007), pp. 1749–1783.

[13] Ernie Chan et al. "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] Rixen D. "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.

[18] 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.

[19] 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.

[20] Jack Dongarra et al. "A Proposal for a Set of Level 3 Basic Linear Algebra Subprograms". In: *SIGNUM Newsl.* 22.3 (July 1987), pp. 2–14.

[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 et al. "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 et al. "Scalable FETI Algorithms for Frictionless Contact Problems". In: *Domain Decomposition Methods in Science and Engineering XVII*. Ed. by Ulrich Langer et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 263–270.

[24] Z. Dostál et al. "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.

[25] Iain S. Duff. "Parallel implementation of multifrontal schemes". In: *Parallel Computing* 3.3 (1986), pp. 193–204.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] Charbel Farhat et al. "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.

[33] Charbel Farhat et al. "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.

[34] Charbel Farhat et al. "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.

[35] Charbel Farhat et al. "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.

[36] C. Farhat et al. *Implicit Parallel Processing in Structural Mechanics*. Computational mechanics advances. North-Holland, 1994.

[37] 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.

[38] 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.

[39] Pierre Gosselet et al. "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.

[40] William Gropp et al. "A high-performance, portable implementation of the MPI message passing interface standard". In: *Parallel Computing* 22.6 (1996), pp. 789–828.

[41] Tongxiang Gu et al. "Multiple search direction conjugate gradient method I: methods and their propositions". In: *International Journal of Computer Mathematics* 81.9 (2004), pp. 1133–1143.

[42] Tongxiang Gu et al. "Multiple search direction conjugate gradient method II: theory and numerical experiments". In: *International Journal of Computer Mathematics* 81.10 (2004), pp. 1289–1307.

[43] A. Gupta and V. Kumar. "Optimally scalable parallel sparse Cholesky factorization". In: Society for Industrial and Applied Mathematics, Philadelphia, PA (United States), Dec. 1995.

[44] 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 et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 137–146.

[45] 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.

[46] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Second. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2002.

[47] 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).

[48] N. Kikuchi and J. Oden. *Contact Problems in Elasticity*. Society for Industrial and Applied Mathematics, 1988.

[49] Armel de La Bourdonnaye et al. "A Non-Overlapping Domain Decomposition Method for the Exterior Helmholtz Problem". In: *Contemporary Mathematics* 218 (1998), pp. 42–66.

[50] F. Magoules, F.X. Roux, and G. Houzeaux. *Parallel Scientific Computing*. ISTE. Wiley, 2015.

[51] 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.

[52] Jan Mandel. "Balancing domain decomposition". In: *Communications in Numerical Methods in Engineering* 9.3 (1993), pp. 233–241. ISSN: 1099-0887.

[53] Jan Mandel and Radek Tezaur. "Convergence of a substructuring method-with Lagrange multipliers". In: *Numerische Mathematik* 73.4 (1996), pp. 473–487.

[54] 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.

[55] Yvan Notay. "Flexible Conjugate Gradients". In: *SIAM Journal on Scientific Computing* 22.4 (2000), pp. 1444–1460.

[56] 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.

[57] 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.

[58] Cosmin G. Petra et al. "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.

[59] 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.

[60] A. Quarteroni and A. Valli. *Domain Decomposition Methods for Partial Differential Equations*. Numerical Mathematics and Scie. Clarendon Press, 1999.

[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 et al. "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 et al. 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 et al. "29. Optimization of Interface Operator Based on Algebraic Approach". In: *Domain Decomposition Methods in Science and Engineering* (2002), p. 297.

[68] F.-X. Roux et al. "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.

Index

Hybridization of FETI Methods

Abstract

In this work new domain decomposition methods and new implementations for existing methods are developed. A new method based on previous domain decomposition methods is formulated. The classic FETI [30] plus FETI-2LM [35] methods are used to build the new Hybrid-FETI. 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. By doing this we search to have a faster and more robust code that can work with configurations that the base methods will not handle it optimally by himself. The performance is tested on a contact problem.
The following part involves the development of a new implementation for the S-FETI method [39], the idea is to reduce the memory usage of this method, to make it able to work in larger problem. Different variation for this method are also proposed, all searching the reduction of directions stored each iteration of the iterative method. 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