Internship report
(Delft University of Technology, the Netherlands)

Study of FETI methods and implementation of FETI in Oofelie code

Master first year

Summer 2007

—o0o—

Maxime LEFRANCOIS

ENS de Cachan

RESUME

Ce stage concerne dans un premier temps l'étude de différentes méthodes de décomposition de domaines du type FETI (en français : Elements Finis Etirés et Interconnectés). Dans un premier temps les différentes algorithmes et les points clé qui mènent à la méthode FETI et à la méthode Duale-Primale FETI (FETI-DP) sont expliqués :
La formulation basique de l'équilibre d'un domaine une fois décomposé : formulation des problèmes locaux et méthodes pour exprimer les conditions d'interface entre ces sous-domaines. Je présente succintement une notation simple qui peut permettre de mieux interpréter des notations d'assemblage de matrices bloc, possédant quelques règles propres cohérentes. Ensuite quelques définitions topologiques concernant la description des interfaces (Sommets, Arêtes et Faces) et des Moyennes de Déplacement (translation) et de Moment au premier ordre (rotation) des dits éléments topologiques.
Ensuite une présentation très succinte des différences entre solveurs directs et solveurs itératifs, et les implications sur une eventuelle parallèlisation.
Ensuite vient la présentation des méthodes FETI à proprement parler :

La méthode FETI originale utilise une description totalement duale de l'assemblage, avec l'utilisation systématique et redondante de multiplicateurs de Lagrange pour l'expression de l'equilibre des interfaces. Une itération de la méthode FETI implique une résolution de problèmes locaux de type Neumann et un preconditionneur de type Dirichlet. Je présente donc dans un premier temps comment le problème se pose et se résoud : comment on se débarasse des équations d'équilibre local par recherche d'une solution partiulière. Ensuite je présente certaines considérations mécaniques à propos du préconditionneur dit de Dirichlet, et ses variantes Lumped et Super-Lumped. Finalement la définition d'un préconditionneur *mechanicallyscalable* est donnée, ce qui ouvre la voie à différentes variantes assurant une meilleure convergence, en particulier lorsqu'il existe des hétérogeneités dans les coefficients des matériaux de part et d'autre d'une interface.

Les différentes méthodes FETI-2 et FETI-DP sont ensuite présentées, comme améliorant la convergence en particulier pour les problèmes de plaques (hors 2D) et de coques. La méthode FETI-2 utilise une seconde projection de la solution sur un espace tel qu'on s'assure qu'il y ait coincidance des Sommets, alors que La méthode FETI-DP utilise simplement une description en partie primale des degrés de liberté, en s'assurant ainsi que la valeur de certains degrés de liberté est la même pour chaque sous domaine concerné. Ces deux méthodes s'assurent par ailleurs qu'aucune des matrices de rigidité n'est alors singulière et il n'existe alors pas de modes rigides utilisable pour la formulation du problème grossier. C'est alors le fait que le problème soit couplé pour les variables primales qui définit ce problème grossier. Dans le cas du seul déplacements des sommets, on comprend que ce problème perd un peu de son sens au niveau mécanique. Des parades existent donc en particulier pour les assemblages qui présentent des hétérogénéités de coefficients qui consistent à utiliser comme variables primales les moyennes en translation ou en rotation d'éléments topologiques. Je présente donc à ce moment la méthode qui utilise un changement de variables.

La seconde partie de ce stage concernait l'implémentation de la méthode FETI dans le code élément fini Oofelie. Oofelie est un annagrame qui veut dire en français code Elément Finis Orienté Objet et Mené par Execution Intéractive. Je présente donc quelques-une des spécificités de ce code, en particulier les différents niveaux de programmation qui existent, les classes principales nommées PhySet, l'architecture de la base de données ce qui est une des raison principales pourquoi on dit que Oofelie est un code Multi-Physique, et dans quel ordre on définit un modèle d'étude.

Finalement je présente la partie programmation à proprement parler :

un premier programme qui applique la méthode FETI sur un domaine paramétré écrit à la main ; quelques approches des niveaux inférieurs à l'interpréteur de commande pour la définition d'une nouvelle fonction d'extraction de matrice et aussi pour faire en sorte que la matrice des modes rigides calculée puisse être utilisée dans l'interpréteur. Ensuite une fois que j'étais suffisement à l'aise avec la programmation dans l'interpréteur et la modification du projet C++ pour -après compilation- obtenir des fonctionnalités supplémentaires, et une fois que Mr Paquay a fini de créer une base de classe nommée *DomainPartitioning* qui utilise le partitionneur de domaine Metis, Une implentation plus générale de l'algorithme FETI est possible.

ACKNOWLEDGEMENT

I thank Mr. Pierre Gosselet for his help to get this internship. Thanks for the different bibliography that were a startpoint for the understanding of FETI-DP.

I thank Ms. Corinne DuBurck for having received us in the department of Precision and Microsystems Engineering of the 3mE faculty (Mechanical, Maritim and Material engineering) of TU Delft, for having helped us in many administrative demarches. Thanks for being such a nice person and for having been here when we needed something. Thanks for having helped us and for having made sure that everything was good for us

I thank a lot Ms. Veronique Rochus for our enlightening dicussions about Oofelie, and of course for the very pleasant time I spent in her office.

I also thank the Open-Engineering community members, especialy Mr. Stephan Paquay for the work he has done with Metis and on the *DomainPartitioning* class, which was of capital importance in this topic. And finaly Mr. Pascal DeVincenzo for his great help on the Oofelie forum, to having helped me to quickly understand so some very dark bugs in some part of my program. Thanks to them two for the different tips about the Oofelie philosophy.

Finally, a lot of thanks to Mr. Daniel Rixen for having received us in received me in the department and for having spent time with us, although he was very busy. Thank you for the lunches and for having been here and helped me to understand more about this topic.

Figure 1: Schwarz's first model for domain decomposition method (1870)

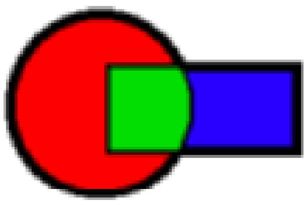


TABLE OF CONTENTS

INTRODUCTION

About FETI

The Finite Element Tearing and Interconnecting method (FETI) was proposed in its original form by Farhat and Roux in reference [12]. It led to a whole family of methods. They all consist of decomposing the domain into non-overlapping subdomains, and using Krylov-type solvers. In all these methods, all or almost all corresponding degrees of freedom on subdomain interface coincidence are enforced by the use of Lagrange multipliers, and eliminating all degrees of freedom, leaving a dual system for the Lagrange multipliers. This dual system is then solved by preconditioned conjugate gradients with a diagonal (or block diagonal) preconditititioner. Evaluation of the dual operator involves the solution of independent Neumann problems in all subdomains, and of a small system of equation for the remaining components. This small system acts like a coarse problem that corresponds to the nullspace of the local stiffness operator and/or to the choosen remaining degrees of freedom for which interface compatibility have been enforced in a different ways. This coarse problem facilitates global exchange of information between the subdomains and therefore causes the condition number to be bounded independently from the number of subdomains.

About Oofelie

OOFELIE is a finite elements code originally developed by a group of researchers at the Laboratoire LTAS - Dynamique des structures and at the INTEC, under the initiative of Igor Klapka and Alberto Cardona. OOFELIE's birth date was at the beginning of the 90's. Today, OOFELIE is developped by Open Engineering and the members of the OOFELIE community. Oofelie stands for Object Oriented Finite Element Led by Interactive Executor and is a toolbox adapted to the solution of multi-fields and multi-methods problems. For the moment it regroups about 1500 files and 650 classes. It is a three-level code : a ground level containing basic classes written in C++, a second level containing what is called I_classes that allows the top level to herit some of the ground functions. Once the C++ project is compiled, the top level is a commands interpretor and acts exactly like any Matlab-like program that uses black-box functions. We should therefore assimilate Oofelie to an opened box Matlab-like code.

The main objectives

When I arrived at Delft, Pr.Rixen told me that he had thought about two different topics of internship. The first was mainly on studying theory about domain decomposition methods and to work with Matlab, whereas the second was really to implement as much algorithms as possible in Oofelie code. Each of them were designed for eight months training periods, as this is often the lenght of internships for Master students in the Netherlands. The few following reasons made me choose a mix a little bit of these two subjects : on the one hand, as I had an overview of some domain decomposition methods at the first semester, I really wanted to learn more about theory, especially on differences that exist between FETI-family methods. But on the second hand, I really wanted to discover Oofelie instead of carrying on working with black boxes in Matlab. Of course I knew I could never finish both jobs in only three months so I chose two main objectives : read and learn as much as possible about FETI, FETI-2, and FETI-DP algorithms, and learn how Oofelie works before trying to implement at least the orginal FETI method in the interpretor.

Part I
THEORETICAL DESCRIPTION OF FETI METHODS

11 Linear elasticity problems and finite elements

11.1 Weak formulation and discret problem associated

We consider an elastic body which occupies a domain $\Omega \in I\!R^3$ and denote his boundary by $\partial\Omega$. We assume that one part $\partial\Omega_D$ of the boundary is clamped, i.e. has homogeneous Dirichlet boundary conditions, and that the rest $\partial\Omega_N := \partial\Omega\ \partial\Omega_D$ is subject to a surface force $\mathbf{g}$, i.e., a natural or Neumann boundary condition. We can also introduce a body force $\mathbf{f}$, e.g., gravity. With $\mathbf{H}^1(\Omega) := \left(H^1(\Omega)\right)^3$, the appropriate space for a variational formulation is the Sobolev space

$$\mathbf{H}_0^1(\Omega, \partial\Omega_D) := \left\{\mathbf{v} \in \mathbf{H}^1(\Omega) : \mathbf{v} = \mathbf{0}\, on\, \partial\Omega_D\right\}$$

The linear elasticity problem which then consists in finding the displacement $\mathbf{u} \in \mathbf{H}_0^1(\Omega, \partial\Omega_D)$ of the elastic structure Ω such that :

$$\int_\Omega G(\mathbf{x})\,\epsilon(\mathbf{u}) : \epsilon(\mathbf{v})\, d\mathbf{x} + \int_\Omega G(\mathbf{x})\,\beta(\mathbf{x})\,\nabla(\mathbf{u})\,\nabla(\mathbf{v})\, d\mathbf{x} = \langle \mathbf{F}, \mathbf{v}\rangle, \forall \mathbf{v} \in \mathbf{H}_0^1(\Omega, \partial\Omega_D). \tag{11.1}$$

Here G and β are material parameters which depend on the Young modulus $E > 0$ and the Poisson ratio $\nu \in (0, 1/2)$; we have $G = E/(1+\nu)$ and $\beta = \nu/(1-2\nu)$. In this article, we only consider the case of compressible elastucity, which means that the Poisson ratio ν is bounded away from $1/2$. Futhermore, $\epsilon/\epsilon_{ij}(\mathbf{u}) := \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)$ is the linearized strain tensor, and

$$\epsilon(\mathbf{u}) : \epsilon(\mathbf{v}) := \sum_{i,j=1}^{3} \epsilon_{ij}(\mathbf{u})\,\epsilon_{ij}(\mathbf{v}),\ \langle \mathbf{F}, \mathbf{v}\rangle := \int_\Omega \mathbf{f}^T \mathbf{v} d\mathbf{x} + \int_{\partial\Omega_N} \mathbf{g}^T \mathbf{v} d\mathbf{s}$$

For convenience we also intoduce the notation

$$(\epsilon(\mathbf{u}), \epsilon(\mathbf{v}))_{L_2(\Omega)} := \int_\Omega \epsilon(\mathbf{u}) : \epsilon(\mathbf{v})\, d\mathbf{x}$$

The bilinear form associated with linear elasticity is then

$$a(\mathbf{u}, \mathbf{v}) = (G\epsilon(\mathbf{u}), \epsilon(\mathbf{v}))_{L_2(\Omega)} + (G\beta\nabla(\mathbf{u}), \nabla(\mathbf{v}))_{L_2(\Omega)}$$

We also need to introduce the null space $\mathbf{Ker}(\epsilon)$ of ϵ which is the space of the six rigid body motions, which is spanned by the three translations $\mathbf{r}_i := \mathbf{e}_i, i = 1, 2, 3$, where the $\mathbf{e}_i$ are the three standard unit vectors, and the three rotations

$$\mathbf{r}_4 = \begin{bmatrix} x_2 - \hat{x}_2 \\ -x_1 + \hat{x}_1 \\ 0 \end{bmatrix}, \mathbf{r}_5 = \begin{bmatrix} -x_3 + \hat{x}_3 \\ 0 \\ x_1 - \hat{x}_1 \end{bmatrix}, \mathbf{r}_6 = \begin{bmatrix} 0 \\ x_3 - \hat{x}_3 \\ -x_2 + \hat{x}_2 \end{bmatrix}. \tag{11.2}$$

Here, $\hat{\mathbf{x}} \in \Omega$ shifts the origin to a point in Ω.

The linear system 11.1 is wellposed since it can be shown that the bilinear form $a(\cdot,\cdot)$ is uniformly elliptic and uniformly continuous. It is therefore sufficient to be discretized by low order, conforming finite elements, e.g. linear or trilinear elements. Let us assume that a triangulation τ^h of Ω is given which is shaped regular and has a typical diameter of h. We denote by $\mathbf{W}^h := \mathbf{W}^h(\Omega)$ the corresponding conforming finite element space of finite element functions. The discrete problem associated to the equation 11.1 is then

$$a(\mathbf{u}_h, \mathbf{v}_h) = \langle \mathbf{F}, \mathbf{v}_h \rangle, \forall \mathbf{v}_h \in \mathbf{W}^h \tag{11.3}$$

As there will be no risk of confusion, we will drop the subscript h

11.2 Problem formulation and description of interface conditions

Assuming that the structure is divided into N_s subdomains, each par $\Omega^{(s)}$ has internal degrees of freedom (d.o.f.) $\mathbf{u}_i^{(s)}$ and boundary d.o.f. $\mathbf{u}_b^{(s)}$ on the interface. Calling $\mathbf{f}_i^{(s)}$ and $\mathbf{f}_b^{(s)}$ the forces applied on the internal and boundary d.o.f. and $\mathbf{g}^{(s)}$ the connection forces to the neighboring domains, the local equilibrium of a subdomain $\Omega^{(s)}$ writes

$$\begin{bmatrix} K_{ii}^{(s)} & K_{ib}^{(s)} \\ K_{bi}^{(s)} & K_{bb}^{(s)} \end{bmatrix} \begin{bmatrix} \mathbf{u}_i^{(s)} \\ \mathbf{u}_b^{(s)} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_i^{(s)} \\ \mathbf{f}_b^{(s)} + \mathbf{g}^{(s)} \end{bmatrix} = \left[\mathbf{f}^{(s)} + \mathbf{g}^{(s)}\right]$$

and on an interface $\Gamma^{(s,t)}$ between subdomains s and t, the interface equilibrium and the interface compatibility respectively writes $\mathbf{g}_{\Gamma^{(s,t)}}^{(s)} + \mathbf{g}_{\Gamma^{(s,t)}}^{(t)} = 0$ and $\mathbf{u}_{-}^{(s)}{}_{\Gamma^{(s,t)}} = \mathbf{u}_{-}(t)_{\Gamma^{(s,t)}}$

To get an algebric expression of these interface conditions, we introduce :

for the interface equilibrium : L which is a Boolean matrix expressioning the assembly of the subdomains on the interface such that the local d.o.f. are identified to the global unique set $\mathbf{u}$ by

$$\begin{bmatrix} \mathbf{u}_i^{(s)} \\ \mathbf{u}_b^{(s)} \end{bmatrix} = \begin{bmatrix} 0 & \cdots & 0 & I & 0 & \cdots & 0 \\ 0 & & & \cdots & & 0 & L_b^{(s)} \end{bmatrix} \begin{bmatrix} \mathbf{u}_i^{(1)} \\ \vdots \\ \mathbf{u}_i^{(N_s)} \\ \mathbf{u}_b \end{bmatrix}$$

assuming for notation purpose that the boundary d.o.f are numbered last in every subdomain, or

$$\mathbf{u}^{(s)} = L^{(s)}\mathbf{u}$$

Then, $L^{(s)T} = \begin{bmatrix} 0 \\ L_b^{(s)T} \end{bmatrix}$ is the operator that projects the local set of d.o.f on the global set of interface d.o.f. We may finaly express the interface equilibrium as

$$\sum_{s=1}^{N_s} L^{(s)T}\left[\mathbf{f}^{(s)} + \mathbf{g}^{(s)}\right] = L^T\left[\mathbf{f} + \mathbf{g}\right]$$

, where

$$L^T = \begin{bmatrix} L^{(1)T} & \cdots & L^{(N_s)T} \end{bmatrix}$$

$$\left[\mathbf{f} + \mathbf{g}\right] = \begin{bmatrix} (\mathbf{f}+\mathbf{g})^{(1)T} & \cdots & (\mathbf{f}+\mathbf{g})^{(N_s)T} \end{bmatrix}^T$$

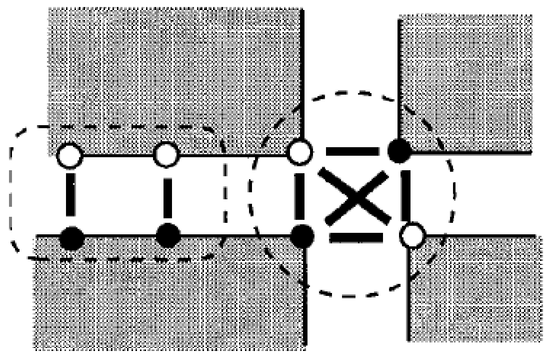
for the interface compatibility : B which is a signed Boolean matrice such that

$$\sum_{s=1}^{N_s} B^{(s)}\mathbf{u}^{(s)} = B\mathbf{u}$$

represents the gap displacement vector across interface and should be equal to null vector for the interface compatibility to be reached.

Remark on image spaces of these two operators : Contrary to the transposed assembly operator L^T which projects forces on a space which dimension is exactly the number of d.o.f on the interface, the operator B is needed to enforce continuity only for the dual displacement variables : those who have one different value for each subdomain they belong to. Futhermore, as one d.o.f on interface may be shared by more than two subdomains, we will use all possible constraint : we therefore choose a fully redundant set of gap variables. This will lead in section 1.4 to the definition of a fully redundant set of Lagrange multipliers. Thus, for a d.o.f common to four subdomains, we will use six gaps rather than as few as three.

Figure 2: Shematization of fully redundant constraints



11.3 Description of block operators notations

These notations are meant to lighten the following definitions of global operators that are constructed as block assembly of local operators. The usefullness of such notations can be discussed as some dimensions criterions can give clues to understand how local operators are assembled, anyway I will show that they can be used in a coherent manner in the rest of the report.

Block diagonal matrices $\underbrace{K}_{D} = diag_{i=1}^{N_s}(K^{(i)})$

Block row matrices $\underbrace{B}_{H} = \begin{bmatrix} B^{(1)} & \cdots & B^{(N_s)} \end{bmatrix}$

Block column matrices $\underbrace{L}_{V} = \begin{bmatrix} L^{(1)T} & \cdots & L^{(N_s)T} \end{bmatrix}^T$

Sum $\underbrace{BL}_{\sum} = \underbrace{B}_{H}\underbrace{L}_{V} = \sum_{i=1}^{N_s} B^{(i)}L^{(i)}$

11.4 Definitions related to nonoverlapping domain decomposition methods

Let the domain $\Omega \in I\!R^3$ be decomposed into nonoverlapping subdomains $\Omega_i, i = 1, ..., N$, each of which is the union of finite elements with matching finite element nodes on the boundaries of neighboring subdomains across the interface Γ. The interfaceΓ is the intersection of three open sets, namely, subdomain faces, edges, and vertices. We will denote individual faces, edges and vertices by *F*, *E*, and *V*, respectively. For the case of regular substructures such as cubes or tetrahedrons, we an use the standard faces, edges, and vertices. To define faces, edges and vertices more generallu, we introduce certain equivalence classes; c.f. [16] or [13]. Let us denote the sets of nodes on $\partial\Omega$, $\partial\Omega_i$, and Γ by $\partial\Omega_h$, $\partial\Omega_{i,h}$, and Γ_h, respectively. For any interface nodal point $x \in \Gamma_h$, we define $N_x := \{j \in \{1, ..., N\} : x \in \partial\Omega_j\}$, i.e. N_x is the set of indices of all subdomains with x in the enclosure of the subdomain. For a node we define the multiplicity as $|N_x|$, the cardinality of the set N_x. Associated with the nodes of the finite element mesh, we have a graph, the nodal graph, which represents the node-to-node adjacency. For a given node $x \in \Gamma_h$, we denote by $C_{con}(x)$ the connected component of the nodal subgraph, defined by N_x, to which x belongs. For two interface points $x, y \in \Gamma_h$, weintroduce an equivalence relation by

$$x \sim y \Leftrightarrow N_x = N_y \, and \, y \in C_{con}(x)$$

We now can describe faces, edges and vertices using their equivalence classes. We define

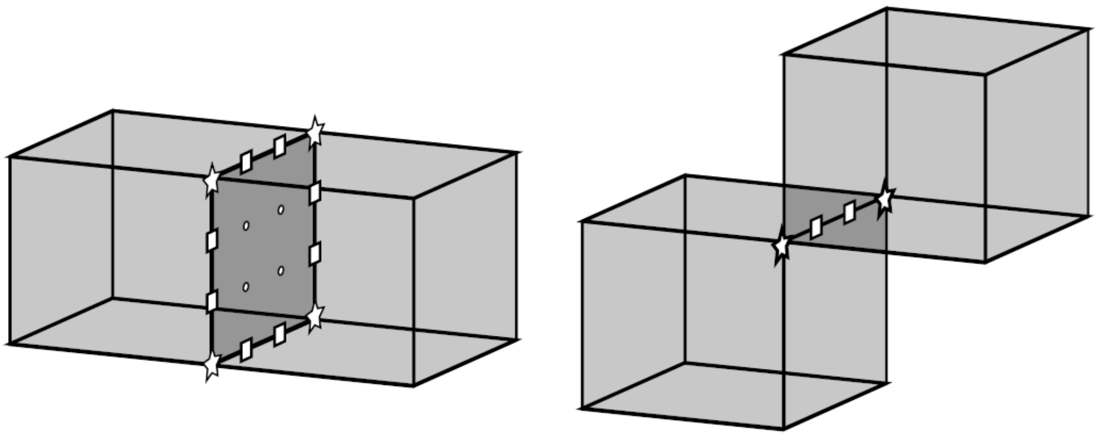
$$\begin{aligned} &x \in F \Leftrightarrow |N_x| = 2, \\ &x \in E \Leftrightarrow |N_x| \geq 3 \, and \, \exists y \in \Gamma_h, y \neq x, such that y \sim x, \\ &x \in V \Leftrightarrow |N_x| \geq 3 \, and \, \nexists y \in \Gamma_h, such that y \sim x. \end{aligned}$$

In the definition of dual-primal FETI method, we need the notion of edge average and also edge first order moments. We note that the rigid bod mdes $\mathbf{r}_1, ...\mathbf{r}_6$, restricted to a staight edge provide only five linearly dependent vectors, since one rotation is always linearly dependent on the other rigid body modes. For the following definition, we assume that we have used an appropriate change of coordinates such that the edge under consideration coincides with the x_1-axis and the special rotation is then $\mathbf{r}_6$. The edge averages and first order moments over this specific edge E are of the form.

$$\frac{\int_E \mathbf{r}_k^T \mathbf{u} d\mathbf{x}}{\int_E \mathbf{r}_k^T \mathbf{r}_k d\mathbf{x}}, k \in \{1, ..., 5\}, \mathbf{u} = \left(u_1^T, u_2^T, u_3^T\right)^T \in \mathbf{W}^h.$$

We note that on edges which are not straight we can use all six rigid body modes to construct six average and first order moment constraints.

Figure 3: Definition of vertices, edges, and faces. Vertices are denoted by stars, edge nodes by squares, and face nodes by small circles. Left: The intersection of the closure of the two subdomains consists of one face, four edges, and four vertices. Right: The intersection of the closure of the two subdomains consists of one edge and two vertices.



12 From sequential to parallel solvers : the parallel processing issue

12.1 Primal approach and direct solvers

For the primal approach, we assume that every d.o.f is primal. This means that displacement d.o.f are identical for all subdomains on the interface. the local d.o.f. are identified to the global set $\mathbf{u}$ by

$$\mathbf{u}^{(s)} = L^{(s)}\mathbf{u}$$

where L is the Boolean matrix defined in 11.2. The initial problem refref can now be expressed as if it results from the assembly of the substructures, namely

$$\begin{bmatrix} K_{ii}^{(1)} & 0 & K_{ib}^{(1)}L_b^{(1)} \\ & \ddots & \vdots \\ 0 & K_{ii}^{(N_s)} & K_{ib}^{(N_s)}L_b^{(N_s)} \\ L_b^{(1)T}K_{bi}^{(1)} \quad \cdots & L_b^{(N_s)T}K_{bi}^{(N_s)} & \sum_{s=1}^{N_s} L_b^{(s)T}K_{bb}^{(s)}L_b^{(s)} \end{bmatrix} \begin{bmatrix} \mathbf{u}_i^{(1)} \\ \vdots \\ \mathbf{u}_i^{(N_s)} \\ \mathbf{u}_b \end{bmatrix} = \begin{bmatrix} \mathbf{f}_i^{(1)} \\ \vdots \\ \mathbf{f}_i^{(N_s)} \\ \sum_{s=1}^{N_s} L_b^{(s)T}\mathbf{f}_b^{(s)} \end{bmatrix}$$

which can be rewritten :

$$\begin{bmatrix} K_{ii} & K_{ib}L_b \\ L_b^T K_{bi} & L_b^T K_{bb}L_b \end{bmatrix} \begin{bmatrix} \mathbf{u}_i \\ \mathbf{u}_b \end{bmatrix} = \begin{bmatrix} \mathbf{f}_i \\ L_b^T\mathbf{f}_b \end{bmatrix}$$

using new notations :

$$K_{ii} = \underbrace{K_{ii}}_{D}; K_{ib}L_b = \underbrace{K_{ib}L_b}_{V}; L_b^T K_{bb}L_b = \underbrace{L_b^T K_{bb}L_b}_{\sum}; \mathbf{u}_i = \underbrace{\mathbf{u}_i}_{V}; \mathbf{f}_i = \underbrace{\mathbf{f}_i}_{V}; L_b^T\mathbf{f}_b = \underbrace{L_b^T\mathbf{f}_b}_{\sum}$$

This expression indicates that the internal stiffness matrices $K_{ii}^{(s)}$ can be factorized independently, which is equivalent to eliminating all internal d.o.f. to build the condensed interface problem for $\mathbf{u}_b$. :

$$\mathbf{u}_i^{(s)} = K_{ii}^{(s)-1}\left(-K_{ib}^{(s)}L_b^{(s)}\mathbf{u}_b^{(s)} + \mathbf{f}_i^{(s)}\right)$$

The condensed interface problem is then obtained using interface conditions and proceeding to the schur fatorization of the global interface operator:

$$S\mathbf{u}_b = \left(\sum_{i=1}^{N_s} L_b^{(i)T}S_{bb}^{(i)}L_b^{(i)}\right)\mathbf{u}_b = \sum_{i=1}^{N_s} L_b^{(i)T}\mathbf{f}_b^{(i)\star} = \mathbf{f}_b^{\star}$$

which is equivalent to

$$S\mathbf{u}_b = \underbrace{L_b^T S_{bb} L_b}_{\sum} \mathbf{u}_b = \underbrace{L_b^T}_{H} \underbrace{S_{bb}}_{D} \underbrace{L_b}_{V} \mathbf{u}_b = \underbrace{L_b^T \mathbf{f}_b^\star}_{\sum} = \underbrace{L_b^T}_{H} \underbrace{\mathbf{f}_b^\star}_{V} = \underbrace{\mathbf{f}_b^\star}_{V}$$

where $S_{bb}^{(s)}$ is called the local Schur complement, and $\mathbf{b}_{bb}^{(s)}$ is the condensed right-hand-side

$$S_{bb}^{(s)} = K_{bb}^{(s)} - K_{bi}^{(s)} K_{ii}^{(s)-1} K_{ib}^{(s)}$$
$$\mathbf{f}_b^{(s)\star} = \mathbf{f}_b^{(s)} - K_{bi}^{(s)} K_{ii}^{(s)-1} \mathbf{f}_i^{(s)}$$

Thus, the decomposition should be such that the number of interface d.o.f is small in order to minimize the size of the global interface problem. But as the factorization of the condensed interface can not be efficiently factorized on parallel and that its size inscreases when the decomposition is refined, such an approach is not scalable on massively parallel computation.

12.2 Iterative methods

Iterative sovers search for approximations of all unknowns simultaneously and involve simple matrix operations. They are thus naturally parallel. For symetric positive definite systems, the most effective algorithm is the Conjugate Gradient method. It can be interpreted as successive Rayleigh-Ritz approximations : we seek for the best approximation of the solution in a subspace which size increases at each iteration. Conjugate gradient consists in building S-orthogonal basis $(w_i)_{0\leq i\leq m}$ of so-called Krylov subspace

$$\mathcal{K}_m\left(z_0, \tilde{S}^{-1}SQ\right) = \text{span}\left(z_0, \ldots, \left(\tilde{S}^{-1}SQ\right)^{m-1} z_0\right)$$

and finding approximation $\begin{cases} x_m \in x_0 + \mathcal{K}_m(S, r_0) \\ r_m \perp \mathcal{K}_m(S, r_0) \end{cases}$ so that at each iteration error $\|u - u_m\|_S$ is minimized. Because of the good conjugasion properties of basis (w_i) the optimization is decoupled and only one scalar coefficient α_m is seeked for at each iteration so that $\|x - x_{m-1} - \alpha_m w_m\|_S$ is minimized.

Key points of coupling domain decomposition methods and Krylov iterative solvers are the choice of the preconditioner $\tilde{S}^{-1}$ and the coarse problem represented by projector Q and intialization $\mathbf{u}_0$.

Table 1: preconditioned conjugate gradient

```
1: Set $Q = I - G(G^T SG)^{-1} G^T S$
2: Compute $u_0 = G(G^T SG)^{-1} G^T b$
3: Compute $r_0 = b - Su_0 = Q^T b$
4: $z_0 = \tilde{S}^{-1} r_0$ set $w_0 = z_0$
5: for $j = 0, \ldots, m$ do
6:     $p_j = SQw_j$ (notice $SQ = Q^T S = Q^T SQ$ )
7:     $\alpha_j = (w_j, r_j)/(p_j, w_j)$
8:     $u_{j+1} = u_j + \alpha_j w_j$
9:     $r_{j+1} = r_j - \alpha_j p_j$
10:    $z_{j+1} = \tilde{S}^{-1} r_{j+1}$
11:    $\beta_j = -(z_{j+1}, p_j)/(w_j, p_j)$
12:    $w_{j+1} = z_{j+1} + \beta_j w_j$
13: end for
```

13 The Classical Finite Element Tearing and Interconnecting method (FETI-1)

Reference [23]
The FETI algoritms (finite elements tearing and interconnecting) is a family of domain decomposition methods for which we distinguish primal and dual displacement variables by the way the continuity of the solution is established. Dual displacements variables are those, for which the continuity is enforced by the a set of Lagrange multipliers and thus the continuity is not established until convergence, as in the classical one-level FETI method. On the other hand, continuity of the primal displacement variables is enforced explicitely at each iteration step by subassembly of the local stiffness matrices at the primal displacement variables. This subassembly leads to a symmetric, positive definite matrix which is coupled at the primal displacement variables but block diagonal otherwise. This coupling yields a global problem which is necessary to obtain a numerically scalable algorithm.

13.1 FETI, the dual schur complement method

The original FETI method is distinguished from others by a fully dual assembly : all corresponding degrees of freedom on subdomain interface coincidence are enforced by the use of Lagrange multipliers. This implies that some subdomains stiffness operators may have a nullspace due to the lack of Dirichlet boundary conditions. We then have to deal with generalized inverse, and to add a linear combination of the nullspace components to the displacements, which is searched so that each subdomain is self-equilibrated after each iteration.

13.1.1 formalization

Introducing Lagrange multipliers λ to enforce the compatibility constraints, the inital problem can be expressed in the equivalent form

$$\begin{bmatrix} K^{(1)} & 0 & & B^{(1)T} \\ 0 & \ddots & & \vdots \\ & & K^{(N_s)} & B^{(N_s)T} \\ B^{(1)} & \dots & B^{(N_s)} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u}^{(1)} \\ \vdots \\ \mathbf{u}^{(N_s)} \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{f}^{(1)} \\ \vdots \\ \mathbf{f}^{(N_s)} \\ 0 \end{bmatrix}$$

To solve the interface problem, we solve this for the local d.o.f. $\mathbf{u}^{(s)}$:

$$\mathbf{u}^{(s)} = K^{(s)+}(\mathbf{f}^{(s)} - B^{(s)T}\lambda) - R^{(s)}\alpha^{(s)}$$

$K^{(s)+}$ is the inverse of $K^{(s)}$ for a subdomain with no rigid modes or a generalized inverse if subdomain s is floating, in which case $R^{(s)}$ are the rigid modes. The forces applied to a floating subdomain must be in self equilibrium, namely

$$R^{(s)T}(\mathbf{f}^{(s)} - B^{(s)T}\lambda) = 0$$

Substituting in the compatibility conditions the expression of $\mathbf{u}^{(s)}$ in terms of λ, and taking into account the subdomain self equilibrium, we find

$$\begin{bmatrix} F_I & G_I \\ G_I^T & 0 \end{bmatrix} \begin{bmatrix} \lambda \\ \alpha \end{bmatrix} = \begin{bmatrix} d \\ e \end{bmatrix}$$

where

$$
\begin{aligned}
F_I &= \sum_{s=1}^{N_s} B^{(s)} K^{(s)+} B^{(s)T} \\
d &= \sum_{s=1}^{N_s} B^{(s)} K^{(s)+} \mathbf{f}^{(s)} \\
G_I &= \left[B^{(1)} R^{(1)} \quad \dots \quad B^{(N_s)} R^{(N_s)} \right] \\
\alpha &= \begin{bmatrix} \alpha^{(1)} \\ \vdots \\ \alpha^{(N_s)} \end{bmatrix} and e = \begin{bmatrix} R^{(1)T} \mathbf{f}^{(1)} \\ \vdots \\ R^{(N_s)T} \mathbf{f}^{(N_s)} \end{bmatrix}
\end{aligned}
$$

F_I is the interface flexibility operator, d is the interface gap created by the applied loads. The interface problem expresses that the connecting forces should be such that they fill the interface gap created by the external loads, and such that, together with the applied loads, they are in equilibrium with respect to the local rigid body modes (self-equilibrium). This problem is called the dual interface problem because it is expressed un terms of the dual variables λ representing the interface connecting forces.

Assuming that the problem to be solved is symmetric positive definite, the interface problem () can be put into a symmetric positive form bu splitting the Lagrange multipliers a

$$\lambda = \lambda_0 + P\bar{\lambda}$$

where

$$
\begin{aligned}
P &= I - QG_I(G_I^T QG_I)^{-1} G_I^T \\
\lambda_0 &= QG_I(G_I^T QG_I)^{-1} e
\end{aligned}
$$

such that the self-equilibrium is satisfied for any $\bar{\lambda}$ and the interface problem finally writes

$$(P^T F_I P)\bar{\lambda} = P^T(d - F_I \lambda_0)$$

An efficient parallel solver then consists in solving the latter equation for λ by Conjugate Gradient iterations. Indeed, the projection operator P requires solving only a small coarse grid problem related to the rigid body modes whereas multiplication by F_I involves solving for $\mathbf{u}^{(s)}$ locally.

the Dirichlet preconditioner is expressed by

$$\Delta\lambda_{k+1} = \tilde{F}_I^{-1} \mathbf{r}_k$$

with $\tilde{F}_I^{-1} = \sum_{i=1}^{N_s} B^{(s)} \begin{bmatrix} 0 & 0 \\ 0 & S_{bb}^{(s)} \end{bmatrix} B^{(s)T}$

and where it is assumed for the notation purposes, that the boundary d.o.f are numbered last in every subdomain. Note that the so-called Lumped preconditioner expressed by

$$\Delta\lambda_{k+1} = \left(\tilde{F}_I^{-1} = \sum_{i=1}^{N_s} B^{(s)} \begin{bmatrix} 0 & 0 \\ 0 & K_{bb}^{(s)} \end{bmatrix} B^{(s)T} \right) \mathbf{r}_k$$

is a low cost variant of the Dirichlet preconditioner for which the internal d.o.f are supposed to remain fixed when prescribing interface displacement. Although it yields slower convergence of the iterations on the interface problem this low cost preconditioner is often used because it reduces the overall computing time.

These preconditioners are obviously naturally parallel since they involves solving local Dirichlet problems.

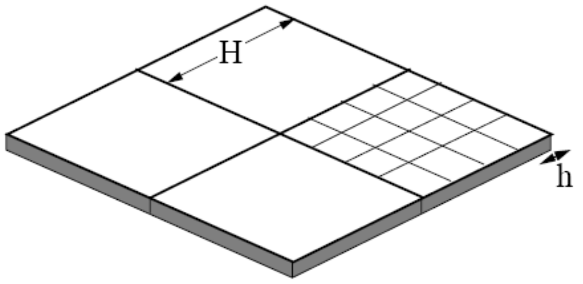
Mandel and Tezaur explained in reference [21] that :

Table 2: FETI or Dual Schur complement method

```
1: Set $k = 0$, $\lambda_0 = QG_I(G_I^T QG)^{-1}e$, $\mathbf{r}_0 = P^T(d - F_I\lambda_0)$
2: while $\|\mathbf{r}_k\| > \epsilon\|d\|$ do
3:     Preconditioning
4:     $\Delta\lambda_{k+1} = \tilde{F}_I^{-1}\mathbf{r}_k$
5:     $_{k+1} = \Delta\lambda_{k+1}$
6:     orthogonalisation of directions
7:     for $i = 1, \dots, k$ do
8:         $\beta_i = -\frac{y_i^T F_I \Delta\lambda_{k+1}}{y_i^T F_I y_i}$, $y_{k+1} = y_{k+1} + \beta_i y_i$
9:     end for
10:    $\eta_{k+1} = \frac{y_{k+1}^T \mathbf{r}_k}{y_{k+1}^T F_I y_{k+1}}$
11:    $\lambda_{k+1} = \frac{y_{k+1}^T \mathbf{r}_k}{y_{k+1}^T F_I y_{k+1}}$
12:    $\mathbf{r}_{k+1} = P^T(\mathbf{r}_k - \eta_{k+1} F_I y_{k+1})$
13:    $k = k + 1$
14: end while
15: $\alpha = (G_I^T QG_I)^{-1} G_I^T Q\mathbf{r}_k$
16: $\mathbf{u}^{(s)} = K^{(s)+}(\mathbf{f}^{(s)} - B^{(s)T}\lambda_k) - R^{(s)}\alpha^{(s)}$
```

Farhat, Mandel and Roux recognized that this small system plays the role of a coarse problem. They also replaced the diagonal preconditioner by a block preconditioner with the solution of independent Dirichlet problems in each subdomain and observed numerically that this Dirichlet preconditioner results in a very slow growth of the condition number with subdomain size. Mandel and Tezaur reference [20] proved that the condition number grows at most as $C(1 + log(H/h))^m$, with $m \leq 3$ and where H is subdomain size and h is element size, both in 2D and 3D.

Figure 4: Mesh size order definition

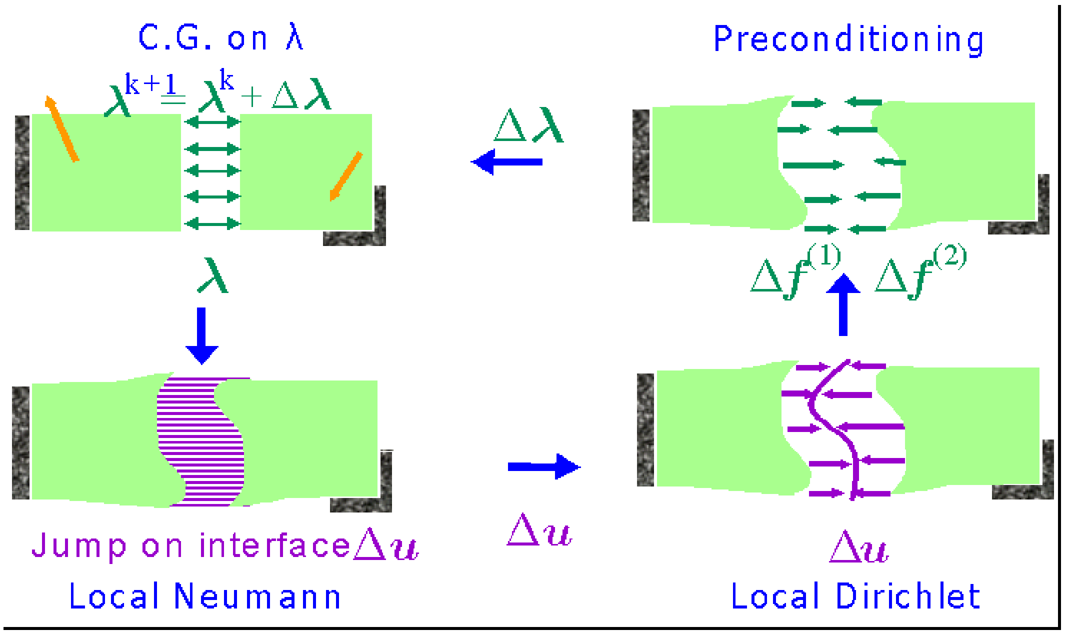


13.1.2 Mechanical considerations

Mechanical interpretation of the FETI method and the Dirichlet preconditioner : At each iteration k of the FETI solver, a new traction Lagrange multipliers field λ^k is computed. Then, for each substructure, a load field equal to $B_b^{(s)T}\lambda^k$ is imposed on its interface boundary, and the resulting substructure displacement field $\mathbf{u}^{(s)k}$ is obtained from the solution of the equilibrium equation. Except at convergence, the substructure solutions $\mathbf{u}^{(s)k}$ are not compatible on the substructure interface boundaries, and their jump is evaluated during the computation of the projected residual w^k

$$w^k = \sum_{s=1}^{N_s} B_b^s \mathbf{u}^{(s)k}$$

Figure 5: Mechanical interpretation of the FETI method and the Dirichlet preconditioner


C.G. on λ
$\lambda^{k+1} = \lambda^{k} + \Delta\lambda$
λ
Jump on interface Δu
Local Neumann
Preconditioning
$\Delta\lambda$
$\Delta f^{(1)}$ $\Delta f^{(2)}$
Δu
Δu
Local Dirichlet

How the preconditioner can improve the condition number for heterogeneities across interface ? From a mechanical viewpoint, the objective of a substructure-by-substructure preconditioner $\tilde{F}^{-1}$ can be stated as that of generating a Lagrange multiplier correction z that reduce as much as possible the displacement jump w. Preconditioning the projected residual w by the Dirichlet or lumped operators yields :

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

The above expression of z suggests that preconditioning by the substructure-by-substructure Dirichlet and lumped operators can be understood as a three-step procedure for building a correction of the Lagrange multiplier field

- First, displacement corrections $\Delta\mathbf{u}_b^{(s)}$ are imposed on the subdomain interfaces as follows

  $$\Delta\mathbf{u}_b^{(s)} = B_n^{(s)T} w$$

  This means that at every interface d.o.f., a displacement correction equal to the sum of the displacement jumps with neighboring d.o.f. is imposed.

- Next, the corresponding interface nodal forces $\Delta\mathbf{f}_b^{(s)}$ are evaluated as

  $$\Delta\mathbf{f}_b^{(s)} = \left(S_{bb}^{(s)} or K_{bb}^{(s)} \right) \Delta\mathbf{u}_b^{(s)}$$

  Note that when the lumped operator $K_{bb}^{(s)}$ is used instead of the Schur complement $S_{bb}^{(s)}$, the computation of the interface forces $\Delta\mathbf{f}_b^{(s)}$ assumes implicitly that the internal d.o.f. are fixed.

- Finally, the jump of interface nodal forces $\Delta\mathbf{f}_b^{(s)}$ are computed to obtain the Lagrange multiplier correction z

  $$z = \sum_{s=1}^{N_s} B_b^{(s)} \Delta\mathbf{f}_b^{(s)}$$

Rixen and Farhat defined in reference [24] a *Mechanically consistent* preconditioner such that z must be constructed such that as the $\Delta\mathbf{f}_b^{(s)}$ approach equilibrium, $B_b^{(s)T}z$ restores the interface force corrections $\Delta\mathbf{f}_b^{(s)}$. They therefore state that a mechanically consistent preconditioner implicitly results in

- displacement increment $\Delta\mathbf{u}_b^{(s)}$ such that the new gap is null :

$$\sum_{s=1}^{N_s} B_b^{(s)}\hat{\mathbf{u}}^{(s)} = \sum_{s=1}^{N_s} B_b^{(s)}\left(\mathbf{u}_b^{(s)} + \Delta\mathbf{u}_b^{(s)}(w)\right) = 0$$

- Lagrange multiplier correction z such that

$$B_b^{(s)T}z(\Delta\mathbf{f}_b^{(s)}) = \Delta\mathbf{u}_b^{(s)}$$

  if the sum of all forces acting on an interface d.o.f. is zero and is therefore the expression of interface equilibrium

$$\Delta\mathbf{f}_b^{(s)} - B_b^{(s)T}\sum_{\substack{r=1\\ r\neq s}}^{N_s} B_b^{(r)}\Delta\mathbf{f}_b^{(r)}$$

It has been shown that for iterative substructuring methods, mechanically consistent preconditioners deliver a better numerical performance than mechanically inconsistent ones.

as each row of $B_b^{(i)}$ with a non zero entry corresponds to a Lagrange multiplier connecting the subdomain $\Omega^{(i)}$ with a neighboring subdomain Ω_j at a point $x \in \partial\Omega_{i,h} \cap \partial\Omega_{j,h}$. The Dirichlet F^{D-1}, Lumped F^{L-1} and Super-Lumped F^{SL-1} preconditionner is then given in matrix form by

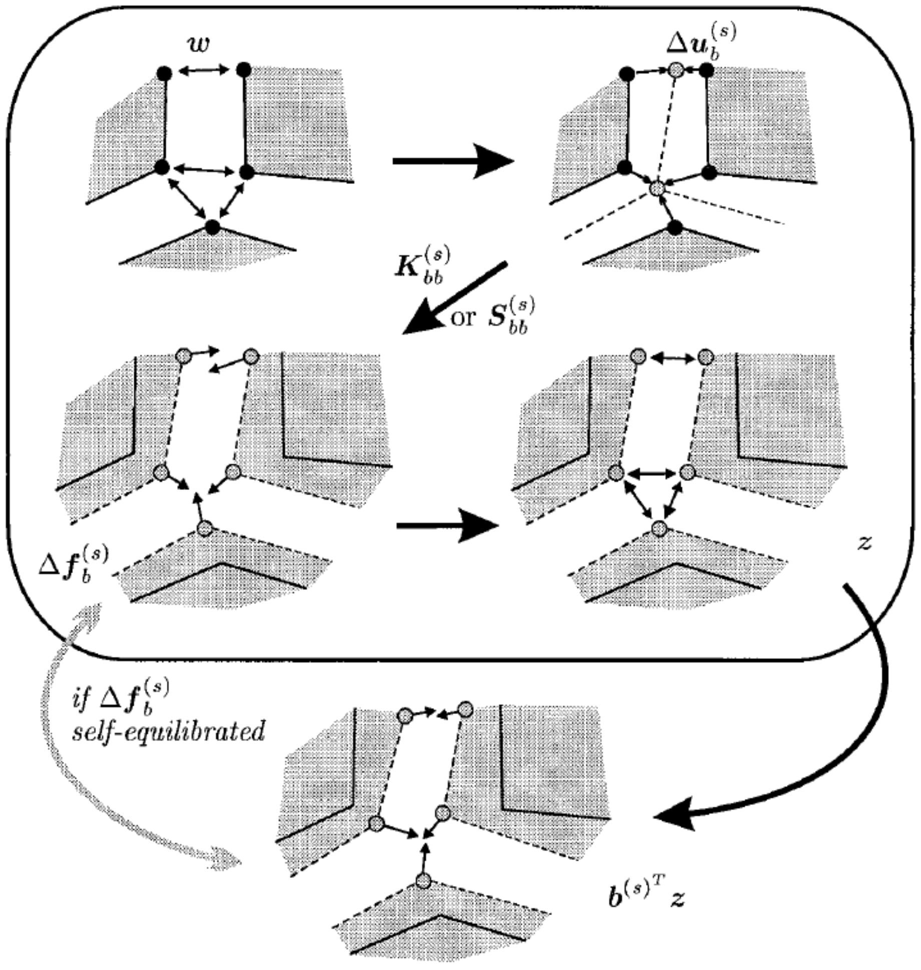
$$F^{D-1} = BDSDB^T = \sum_{i=1}^{N_s} B^{(i)}D^{(i)}S^{(i)}D^{(i)}B^{(i)T}$$

$$F^{L-1} = BDK_{bb}DB^T = \sum_{i=1}^{N_s} B^{(i)}D^{(i)}K_{bb}^{(i)}D^{(i)}B^{(i)T}$$

$$F^{SL-1} = BDdiag(K_{bb})DB^T = \sum_{i=1}^{N_s} B^{(i)}D^{(i)}diag(K_{bb}^{(i)})D^{(i)}B^{(i)T}$$

with the scaling parameter $D^{(i)}$ is choosen among :
-$D^{(i)} = diag\left(\frac{1}{multiplicity}\right) = diag\left(\frac{1}{\|N_x\|}\right)$ for homogeneous structures,
-$D^{(i)} = diag\left(\frac{diag\left(K_{bb}^{(i)}\right)_i}{\sum_{j\in N_i} diag\left(K_{bb}^{(j)}\right)_i}\right)$ for compressible heterogeneous structure (assuming s represents the same d.o.f shared by the set N_s of subdomains),
-$D^{(i)} = diag\left(\frac{\mu_i^{(i)}}{\sum_{j\in N_i}\mu_i^{(j)}}\right)$ for incompressible heterogeneous structures (assuming s represents the same d.o.f shared by the set N_s of subdomains)

Figure 6: Conditions for a mechanical consistent preconditioner


w
$\Delta u_b^{(s)}$
$K_{bb}^{(s)}$ or $S_{bb}^{(s)}$
$\Delta f_b^{(s)}$
z
if $\Delta f_b^{(s)}$ *self-equilibrated*
$b^{(s)^T} z$

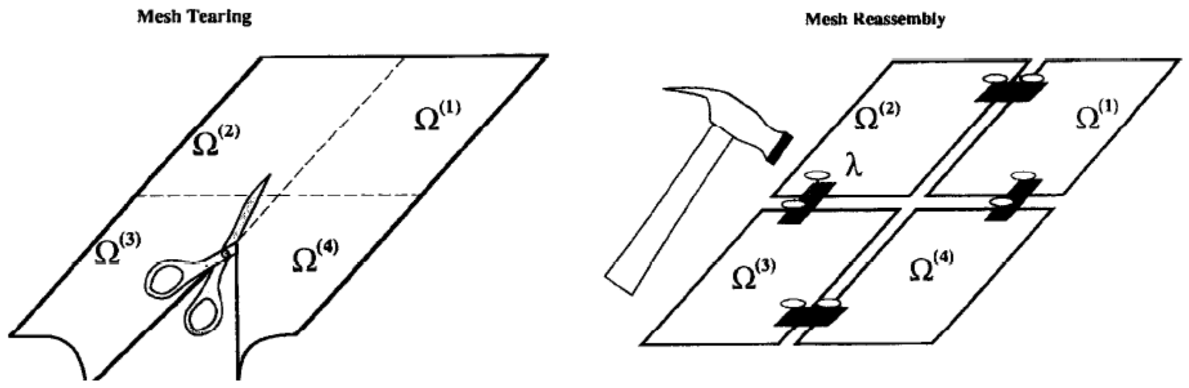
14 Variants to the FETI method

reference [11]
The basic FETI method has been successfully extended to modal analysis reference [8] transient response simulations reference [5], heterogeneous problems reference [4], and systems with multiple and/or repeated right-hand sides reference [7] [3]. However, for plate bending and shell problems, the condition number of the preconditioned FETI interface problem was observed to grow fast with the number of elements per substructure reference [2]. So far, curing this problem has been the major if not only obstacle between the FETI method and versatility, reliability, and universal high-performance.

14.1 The FETI 2 levels method (FETI-2)

The original method of reference [12] [2] does not converge well for plate and shell problems, and the existence and the form of the coarse space depend on the singularity of the subdomain matrices. Therefore, Mandel, Tezaur, and Farhat reference [22] and Farhat, Mandel, and Chen reference [11] proposed to project the Lagrange multipliers in each iteration onto an auxiliary space. With the auxiliary space chosen so that the corresponding primal solutions are continuous at the crosspoints, it is possible to prove that the condition number does not grow faster than $C(1 + log(H/h))^3$ for plate problems reference [22], and fast convergence was observed for plate reference [11] as well as shell problems reference [6]. This method is now called FETI-2. For related results for symmetric positive definite problems, see reference [?] [25] and references therein.

Figure 7: Shematization of tearing stage and FETI-2 reassembly stage



14.2 The Dual-Primal FETI method (FETI-DP)

The Dual-Primal FETI method (FETI-DP) was introduced by Farhat et al. reference [9]. This method enforces the continuity of the primal solution at the crosspoints directly in the formulation of the dual problem: the degrees of freedom at a crosspoint remain common to all subdomains sharing the crosspoint and the continuity of the remaining degrees of freedom on the interfaces in enforced by Lagrange multipliers : they are said to be primal variables. The degrees of freedom are then eliminated and the resulting dual problem for the Lagrange multipliers is solved by preconditioned conjugate gradients with a Dirichlet preconditioner. Evaluating the dual operator involves the solution of independent subdomain problems with nonsingular matrices and of a coarse problem based on the subdomain corners. The advantage of this method is a simpler formulation than those of reference [22] [11]; there is also no need to solve problems with singular matrices, and the method has been observed to be significantly faster in practice for 2D problems.

Jan Mandel and Radec Tezaur prove in reference [21] that the condition number of the FETI-DP method with the Dirichlet preconditioner does not grow faster than $C(1 + log(H/h))^2$ for both second order and fourth order problems in 2D. By spectral equivalence, the result for fourth order problems extends to a large class of Reissner-Mindlin elements for plate bending as in reference [22].

A condition number of the order of $\kappa(M^{-1}F) = O(1 + \log^2(\frac{H}{h}))$ reflects optimal convergence properties because :

when the number of substructures is fixed and the mesh is refined, h is decreased, and therefore the condition number and convergence rate of the FETI method deteriorates only as $O\left(\log^3\left(1/h\right)\right)$, whereas the condition number of the unpreconditioned stiffness matrix associated with a second-order elasticity problem grows asymptotically as $(1/h)$.

Suppose that a given mesh is fixed, one processor is assigned to every substructure, and the number of substructures is increased in order to increase parallelism. In that case, h is fixed and H is decreased. From reference [11], it follows that for second-order elasticity problems, the condition number of the preconditioned FETI interface problem decreases. This implies that the number of iterations for convergence can be expected to decrease with an increasing number of substructures.

On most distributed memory parallel processors, the total amount of available memory increases with the number of processors. When solving a certain class of problems on such parallel hardware, it is customary to define in each processor a constant subproblem size, and increase the total problem size with the number of processors. In such a case, h and H are decreased, but the ratio H/h is kept constant. It then follows that the FETI method can solve larger second-order elasticity problems with the same number of iterations as smaller ones, simply by increasing the number of substructures.

14.2.1 Formalization

Reference [14] We will use the subscripts I, Δ and Π to denote the interior dual and primal displacement variables, respectively and obtain for the local stiffness matrices, load vectors and solution vectors of nodal displacement

$$K^{(i)} = \begin{bmatrix} K_{II}^{(i)} & K_{\Delta I}^{(i)T} & K_{\Pi I}^{(i)T} \\ K_{\Delta II}^{(i)} & K_{\Delta\Delta}^{(i)} & K_{\Pi\Delta}^{(i)\,T} \\ K_{\Pi I}^{(i)} & K_{\Pi\Delta}^{(i)} & K_{\Pi\Pi}^{(i)} \end{bmatrix}, \mathbf{f}^{(i)} = \begin{bmatrix} \mathbf{f}_I^{(i)} \\ \mathbf{f}_\Delta^{(i)} \\ \mathbf{f}_\Pi^{(i)} \end{bmatrix}, \mathbf{u}^{(i)} = \begin{bmatrix} \mathbf{u}_I^{(i)} \\ \mathbf{u}_\Delta^{(i)} \\ \mathbf{u}_\Pi^{(i)} \end{bmatrix} \tag{14.1}$$

We also introduce the notation

$$\mathbf{u}_b = \begin{bmatrix} \mathbf{u}_I^T & \mathbf{u}_\Delta^T \end{bmatrix}^T, \mathbf{f}_b = \begin{bmatrix} \mathbf{f}_I^T & \mathbf{f}_\Delta^T \end{bmatrix}^T, \mathbf{f}_b^{(i)} = \begin{bmatrix} \mathbf{f}_I^{(i)T} & \mathbf{f}_\Delta^{(i)T} \end{bmatrix}^T, \mathbf{u}_b^{(i)} = \begin{bmatrix} \mathbf{u}_I^{(i)T} & \mathbf{u}_\Delta^{(i)T} \end{bmatrix}^T \tag{14.2}$$

Introducing local assembly operators $L_\Pi^{(i)T}$ which map from the space of local primal displacement variables to that of the global, assembled primal displacement variables writes

$$\tilde{K} = \begin{bmatrix} K_{BB}^{(1)} & 0 & 0 & K_{\Pi B}^{(1)\,T} L_\Pi^{(1)} \\ 0 & \ddots & 0 & \vdots \\ 0 & 0 & K_{BB}^{(N_s)} & K_{\Pi B}^{(N_s)T} L_\Pi^{(N_s)} \\ L_\Pi^{(1)T} K_{\Pi B}^{(1)} & \cdots & L_\Pi^{(N_s)T} K_{\Pi B}^{(N_s)} & \sum_{s=1}^{N_s} L_\Pi^{(s)T} K_{\Pi\Pi}^{(s)} L_\Pi^{(s)} \end{bmatrix}$$

we will denote by a tilde the subassembled matrices and

$$\tilde{K}_{\Pi B}^{(i)} = L_\Pi^{(i)T} K_{\Pi B}^{(i)}, \tilde{K}_{\Pi B}^{(i)} = \left[L_\Pi^{(1)T} K_{\Pi B}^{(1)}, \cdots, L_\Pi^{(N_s)T} K_{\Pi B}^{(N_s)} \right], \tilde{K}_{\Pi\Pi}^{(i)} = \sum_{i=1}^{N_s} L_\Pi^{(i)T} K_{\Pi\Pi}^{(i)} L_\Pi^{(i)},$$

$$K_{BB} = diag_{i=1}^{N_s}\left(K_{BB}^{(i)}\right), K_{BB}^{(i)} = \begin{bmatrix} K_{II}^{(i)} & K_{\Delta I}^{(i)\,T} \\ K_{\Delta II}^{(i)} & K_{\Delta\Delta}^{(i)} \end{bmatrix},$$

thus we obtain

$$\tilde{K} = \begin{bmatrix} K_{BB} & \tilde{K}_{\Pi B}^T \\ \tilde{K}_{\Pi B} & \tilde{K}_{\Pi\Pi} \end{bmatrix},$$

We note that K_{BB} is a block diagonal matrix.

for $i = 1, \ldots, N_s$. Due to the subassembly of the primal displacement variables, Lagrange Lagrange multipliers have to be used to enforce continuity only for the dual displacement variables $\mathbf{u}_\Pi$. We introduce a discrete jump operator $B = \begin{bmatrix} OB_D \end{bmatrix}$ such that the solution u_D, associated with more than one subdomain, coincides when $B\mathbf{u}_B = B_D\mathbf{u}_D = 0$. Since we assume pointwise matching grids across the interface C, the entries of the matrix B can be chosen as 0, 1, and -1. However, we will otherwise use all possible constraints and thus work with a fully redundant set of Lagrange multipliers as in reference [15]Section 5]; cf. also reference [24]. Thus, for an edge node common to four subdomains, we will use six Lagrange multipliers rather than choosing as few as three. We can now reformulate the problem as

$$\tilde{K} = \begin{bmatrix} K_{BB} & \tilde{K}_{\Pi B}^T & B^T \\ \tilde{K}_{\Pi B} & \tilde{K}_{\Pi\Pi} & 0 \\ B & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u}_B \\ \tilde{\mathbf{u}}_\Pi \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{f}_B \\ \tilde{\mathbf{f}_\Pi} \\ 0 \end{bmatrix},$$

where elimination of the primal variables and of the interior and dual displacement variables leads to a reduced linear system of the form :

$$F\lambda = d$$

formally obtained by block Gauss elimination i.e. we have

$$\begin{aligned} \tilde{S}_{\Pi\Pi} &= \tilde{K}_{\Pi\Pi} - \tilde{K}_{\Pi B} K_{BB}^{-1} \tilde{K}_{\Pi B}^T, \\ F &= BK_{BB}^{-1}B^T + BK_{BB}^{-1}\tilde{K}_{\Pi B}^T \tilde{S}_{\Pi\Pi}^{-1} \tilde{K}_{\Pi B} K_{BB}^{-1} B^T, \\ d &= BK_{BB}^{-1}\mathbf{f}_B + BK_{BB}^{-1}\tilde{K}_{\Pi B}^T \tilde{S}_{\Pi\Pi}^{-1} \left(\tilde{\mathbf{f}}_\Pi - \tilde{K}_{\Pi B} K_{BB}^{-1} \mathbf{f}_B \right), \end{aligned}$$

The matrix F is never built explicitly but in every interation appropriate systems are solved the first term of the sum on the right hand side of the representation of F applied to a vector can be computed completely in parallel since K_{BB} is a bock diagonal matrix, and the second term in that sum concernes the global problem needed for scalability.

As for the FETI methods, we then solve the $F^{-1}F\lambda = d$ preconditioned linear system with F^{-1} is the scaled Dirichlet, Lumped or Super-Lumped preconditioner.

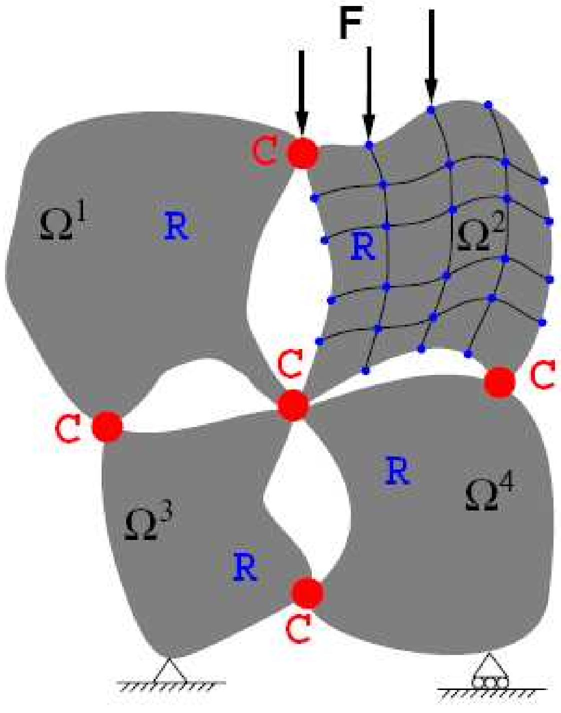
14.2.2 How the convergence of FETI-DP can be improved ?

The finite elements discretization of highly heterogeneous linear elasticity problems leads to large and ill-conditioned problems to solve. Thus they provide good examples to test the robustness of methods and see if they can converge independently of the material discontinuities.

The main point of the FETI-DP algorithm is the choice of the primal displacement variables to select for the coarse problem. So naturally the seek for a good 3D variant of the method led to the choice of the best displacement variables (primal variables) to choose : instead of corner displacement (Vertices in 3D), one can choose average displacements of edges or faces. In reference [18] where FETI-DP is described, some edge averages are first mentionned, then this method was experimented in 2003 when FETI-DP was implemented in Salinas, see reference [**?**] and is was shown scalable on massively parallel computers. Finally in reference [17]. Unfortunately this choice does not always lead to a good convergence results in three dimensions. To obtain better convergence, another coarse problem was suggested by introducing some additional constraints as averages or first order moments over selected edges and faces. these choices may lead to robust condition number bounds for highly heterogeneous materials

There are different ways of implementing these additional primal constraints. One is to use additional, optional Lagrange multipliers, see reference [10] or reference [16], another one is to apply a transformation of basis, see reference [10] [16] [19]. In [14], the use of a transformation of basis is explained. Let us note that this approach leads again to a mixed linear system of the form (*5) and

Figure 8: Dual-Primal Mesh partitions



that the same algorithmic form as for Algorithm A can be used; see reference [10] [16] [?] [19] for further details.

14.2.3 Theoretical implementation of additional primal constraints : with a transformation of basis

From the inner products of $\mathbf{u}_E$ with the translational and the rotational rigid body modes, we obtain three averages and two or three first order moments, respectively; cf. (*4). These averages and moments are explicitly introduced as new variables into the new basis and will form a part of the set of primal displacement variables. The dual displacement vectors in the new basis will have a zero edge average and will be orthogonal to the rotations on the fully primal edge under consideration; this can also be seen as having certain first order moments to be zero. We now describe how the transformation matrix for such a change of basis can be constructed. First, we consider the construction of the basis transformation for a single, fully primal edge. We consider the six rigid body modes $\mathbf{r}_i, i = 1, \ldots, 6$; cf. Section *2. Next, we orthogonalize the rigid body modes on the edge against each other using a stable formulation of the GramSchmidt process, e.g., modified GramSchmidt. We note that the translational rigid body modes are already orthogonal to each other and thus, we only have to start with the rotations in order to obtain an orthogonal basis of rigid body modes on the edge E. We denote the orthogonal basis obtained by this process by $(\hat{\mathbf{r}}_j)_{j=1,\ldots,l}$, with $l \in \{5, 6\}$. When restricted to straight edge E, one of the rotations is linearly dependent on the others and should vanish when modified GramSchmidt is used; cf. also the discussion at the end of Section *2. Then, we only have a five dimensional basis. Let us assume that the vector of nodal unknowns $\mathbf{u}_E$ has length n. We then consider the set of vectors $\{(\hat{\mathbf{r}}_j)_{j=1,\ldots,l}, (_i)_{i=1,\ldots,n}\}$, where e_i is the unit vector with one at the ith component and zero otherwise, which is associated with the ith d.o.f. on the fully primal edge. Starting with the orthogonalized rigid body modes $(\hat{\mathbf{r}}_j)_{j=1,\ldots,l}$, we orthogonalize the set of $n+l$ vectors, using modified GramSchmidt. We discard the l linearly dependent vectors and use the remaining n orthogonal vectors to define the column vectors of our transformation matrix T_E. The transformation matrix T_E performs the desired change of basis from the new basis to the original nodal basis. Denoting the edge unknowns in the new basis by $hat\mathbf{u}_E$, we have

$$\mathbf{u}_E = T_E \hat{\mathbf{u}}_E$$

A similar construction can be carried out for an edge where only averages are used as primal constraints. In this case,we orthogonalize only against the translational rigid body modes. Only edge averages are then introduced as new variables and the remaining new variables will have zero edge average. We note that in this case, we can also explicitly set up a transformation matrix for the basis transformation without using a GramSchmidt process; see, e.g., reference [16]. Next, we consider all edges on the boundary of a single subdomain Ω_i. The transformation matrix, which operates on all relevant edges of $\partial\Omega_i$, will be denoted by $T_E^{(i)}$. Then, the transformation $T^{(i)}$ for all variables of one subdomain Ω_i is of the form

$$T^{(i)} = \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & T_E^{(i)} \end{bmatrix}$$

Here, we assume that the variables are ordered interior variables first, interface variables not related to the (fully) primal edges second, and the variables on the (fully) primal edges last, i.e., a typical vector of nodal unknowns is of the form $\begin{bmatrix} \mathbf{u}_I^{(i)T} & \mathbf{u}_{\bar{\Gamma}}^{(i)T} & \mathbf{u}_E^{(i)T} \end{bmatrix}^T$. Here, we denote the interface variables not related to a (fully) primal edge by the subscript $\bar{\Gamma}$. We note that $T_E^{(i)}$ is a block-diagonal matrix where each block represents the transformation of a component of a (fully) primal edge. Decomposing the subdomain stiffness matrices $K^{(i)}$ in the same manner, we obtain

$$K^{(i)} = \left[\begin{array}{cc|c} K_{II}^{(i)} & K_{I\bar{\Gamma}}^{(i)} & K_{IE}^{(i)} \\ K_{\bar{\Gamma}I}^{(i)} & K_{\bar{\Gamma}\bar{\Gamma}}^{(i)} & K_{\bar{\Gamma}E}^{(i)} \\ \hline K_{EI}^{(i)} & K_{E\bar{\Gamma}}^{(i)} & K_{EE}^{(i)} \end{array}\right]$$

Using the transformation $\mathbf{u}_E = T_E \hat{\mathbf{u}}_E$, we obtain

$$T^{(i)T} K^{(i)} T^{(i)} = \left[\begin{array}{cc|c} K_{II}^{(i)} & K_{I\bar{\Gamma}}^{(i)} & K_{IE}^{(i)} T_E^{(i)} \\ K_{\bar{\Gamma}I}^{(i)} & K_{\bar{\Gamma}\bar{\Gamma}}^{(i)} & K_{\bar{\Gamma}E}^{(i)} T_E^{(i)} \\ \hline T_E^{(i)T} K_{EI}^{(i)} & T_E^{(i)T} K_{E\bar{\Gamma}}^{(i)} & T_E^{(i)T} K_{EE}^{(i)} T_E^{(i)} \end{array}\right]$$

where the upper left 2 x 2 block matrix is not affected by the basis transformation. The primal variables in the new basis consist now of averages and first order moments but we note that there might be also selected primal vertices as additional primal variables. As before, the primal variables belonging to Ω_i are denoted by $\mathbf{u}_\Pi^{(i)}$ and the remaining, dual displacement variables by $\mathbf{u}_\Delta^{(i)}$. By construction, the basis functions associated with the new dual displacement variables have zero edge average over primal edges. In the same manner the indices Δ_E and Π_E indicate the dual and primal displacement variables associated with the primal edge constraints. Denoting the transformed matrices by an overline and ordering the primal edge variables last, we obtain

$$T^{(i)T} K^{(i)} T^{(i)} = \left[\begin{array}{cc|cc} K_{II}^{(i)} & K_{I\bar{\Gamma}}^{(i)} & K_{I\Delta_E}^{(i)} & K_{IE}^{(i)} \\ K_{\bar{\Gamma}I}^{(i)} & K_{\bar{\Gamma}\bar{\Gamma}}^{(i)} & K_{\bar{\Gamma}\Delta_E}^{(i)} & K_{\bar{\Gamma}E}^{(i)} \\ \hline K_{\Delta_E I}^{(i)} & K_{\Delta_E\bar{\Gamma}}^{(i)} & K_{\Delta_E\Delta_E}^{(i)} & K_{\Delta_E E}^{(i)} \\ K_{\Pi_E I}^{(i)} & K_{\Pi_E\bar{\Gamma}}^{(i)} & K_{\Pi_E\Delta_E}^{(i)} & K_{\Pi_E\Pi_E}^{(i)} \end{array}\right]$$

Denoting the primal vertices by a subscript Π_V and the remaining dual displacement variables by a subscript Δ, we can then write $\mathbf{u}_{\bar{\Gamma}}^{(i)} = [\mathbf{u}_\Delta^{(i)T} \mathbf{u}_{\Pi_V}^{(i)\ T}]^T$. Using this splitting for the local stiffness matrices $K^{(i)}$ accordingly, ordering the primal variables $\mathbf{u}_{\Pi_V}^{(i)}$ and $\mathbf{u}_{\Pi_E}^{(i)}$ last, and combining them as primal variables $\mathbf{u}_\Pi^{(i)} = [\mathbf{u}_{\Pi_V}^{(i)\ T} \mathbf{u}_{\Pi_E}^{(i)\ T}]^T$, we obtain

$$T^{(i)T} K^{(i)} T^{(i)} = \begin{bmatrix} K_{II}^{(i)} & \bar{K}_{\Delta I}^{(i)T} & \bar{K}_{\Pi I}^{(i)T} \\ \bar{K}_{\Delta I}^{(i)} & \bar{K}_{\Delta\Delta}^{(i)} & \bar{K}_{\Pi\Delta}^{(i)\ T} \\ \bar{K}_{\Pi I}^{(i)} & \bar{K}_{\Pi\Delta}^{(i)} & \bar{K}_{\Pi\Pi}^{(i)} \end{bmatrix}$$

Assembling the primal contributions of each transformed $K^{(i)}$ and ordering the primal variables last, we obtain

$$\tilde{K} := \begin{bmatrix} K_{II}^{(1)} & \bar{K}_{\Delta I}^{(1)T} & & & & \tilde{K}_{\Pi I}^{(1)T} \\ \bar{K}_{\Delta I}^{(1)} & \bar{K}_{\Delta\Delta}^{(1)} & & & & \tilde{K}_{\Pi\Delta}^{(1)\,T} \\ & & \ddots & & & \vdots \\ & & & K_{II}^{(N_s)} & \bar{K}_{\Delta I}^{(N_s)T} & \tilde{K}_{\Pi I}^{(N_s)T} \\ & & & \bar{K}_{\Delta I}^{(N_s)} & \bar{K}_{\Delta\Delta}^{(N_s)} & \tilde{K}_{\Pi\Delta}^{(N_s)T} \\ \tilde{K}_{\Pi I}^{(1)} & \tilde{K}_{\Pi\Delta}^{(1)} & \dots & \tilde{K}_{\Pi I}^{(N_s)} & \tilde{K}_{\Pi\Delta}^{(N_s)} & \tilde{K}_{\Pi\Pi} \end{bmatrix} := \begin{bmatrix} \bar{K}_{BB} & \tilde{K}_{\Pi B}^{T} \\ \tilde{K}_{\Pi B} & \tilde{K}_{\Pi\Pi} \end{bmatrix}$$

.

The transformation of basis changes the sparsity pattern of the transformed matrices $T^{(i)T}K^{(i)}T^{(i)}$ compared to that of the original local stiffness matrices $K^{(i)}$ but only the matrix blocks related to the edge degrees of freedom are affected; cf. (*8). We note that the set of edge degrees of freedom is only a small subset of the overall set of interface degrees of freedom. Thus, the transformation of basis only slightly affects the sparsity pattern. Using local Lagrange multipliers, the sparsity of the transformed matrices can be further improved; cf. reference [16]Section 6.2]. In our FETI-DP algorithm in this article, we always assume that we have performed an appropriate change of basis. If there is no danger of confusion, we will drop the overline notation which indicates the dual displacement variables in the transformed basis. Using the transformation of basis, we again obtain a system of the same form as in (*5),

$$\begin{bmatrix} K_{BB} & \tilde{K}_{\Pi B}^{T} & B^T \\ \tilde{K}_{\Pi B} & \tilde{K}_{\Pi\Pi} & 0 \\ B & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u}_B \\ \tilde{\mathbf{u}}_\Pi \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{f}_B \\ \tilde{\mathbf{f}}_\Pi \\ 0 \end{bmatrix}$$

We note that, after the change of basis has been carried out, we can always use the same implementation since the algorithmic description earlier in this section is based on (*5) and thus does not depend on a specific choice of primal and dual variables. Note that the local problems as well as the Schur complement $\tilde{K}_{\Pi\Pi}$ remain symmetric positive definite. In the theory presented in Klawonn and Widlund reference [16], it is assumed that the subdomains are polytopes with good aspect ratios and that the edges are straight. Furthermore, large material discontinuities should be aligned with the interface. But condition number estimates have also been done in reference [16] for some case not predicted by the theory.

Part II
IMPLEMENTATION OF FETI METHOD IN OOFELIE CODE

21 Few things to know about the Oofelie toolbox : specificities and limitations

In my case, the commands interpretor seemed sufficient to introduce the FETI algorithm. In fact, but the domain partitionner that did not exist at the beginning, it allows the definition of the problem, of the resolution schemes and no compilation is needed to test the new algorithms.

specificities of ".d"-directories and ".e"-files New algorithms may be written in a file *name_of_the_file* .e which is compiled once called in the interpretor by the syntax *name_of_the_file*; or *name_of_the_file*.e; Oofelie looks then in a recursive way into every directories of working directory that may contain such a ".e" file. These special directories are those which name looks like *name_of_the_directory*.d . Once the file is found, it is parsed and the file is executed in the interpretor. As no compilation is needed to test algorithms in ".e" files, they are a very important tool in a first stage of implementing new methods. There are two different kind of ".e" :

- those who will only be executed once in the whole algorithm : they don't need any special heading and can contain only commands. They may use and modify any of the already declared variables. They may be called more than once in an algorithm but it is not adviced as it will be parsed and compiled each time by the command interpretor. Furthermore, for the time being, a bug appears when it is used in a loop : the file end acts like a "continue" command in C++, it means that every command between the end of the file call and the end of the loop is ignored, without stopping the loop.
- those who will be called two or more times : they must be declared as functions very similarly to C++ syntax. Every object used in such a function must be either defined as an input argument, or declared on the command list. Once they are called in the program, they are parsed and compiled by the interpretor but unlike the simple ".e" files, this happens only once and the interpretor keeps in memory how it behaves. Their syntax is as follows :

Table 3: function syntax in Oofelie's command interpretor

```
Function type_of_output_object name_of_function ( type_of_input_object name_of_input_object , ... )
{
commands_list
}
```

types of data Many types are available to store data and save space : some of them are written down below. In the interpretor *doubles* and *integers* have the same type named *scalar*. Let me define more precisely what are the specificities of some of the matrices storage :

- $Matr3$-type matrices are 3-dimensional matrices. they only use the amount of 9 scalars.
- $Matrix$-type matrices are n by m real matrices : they use an amount of $n * m + 2$ scalars
- $CMatrix$-type matrices are n by m complex matrices : they use an amount of $2 * n * m + 2$ scalars
- $SkyMatrix$-type matrices are sparse n by m matrices : they use an amount of $\sum_{i=1}^{m}(b_i - a_i + 1)$ scalars where $A_{ji} = 0$ for $j < a_i$ and $A_{ji} = 0$ for $j > b_i$
- $CSRMatrix$-type matrices are fully sparse n by m matrices that use an amount of $3 * p$ scalars for p declared values. every other value is considered as equal to the default value (0 for example)

all of these matrices have the same parent class $MotherMatrix$ but they may have specific functions that are not implemented in their common parent class. We always have to take care about what specific functions are available in parallel to willing save as much storage space as possible.

the std library Many types are available to store data and save space : some of them are written down below. In the interpretor *doubles* and *integers* Thanks to the accessibility to a C++ library called *standard* library, one can acces to a very usefull **std** function in the interpretor : the **std::vector**<*type* >dynamic array that contains a list of a specific type. the **std::vector**<*type* >function has three very important functions that are pop_up, push_back and get_dim. The following shows how works **std::vector**<*type* >declaration and use. Unlike in C++, it is not possible to define std::vector<std::vector<>> in the interpretor. so some of the strategies of data storage for the interpretor may have to be retought once we want to implement directly in the C++ code.

Table 4: std::vector<> declaration and use

```
1: std::vector<scalar >A (3); creates a std::vector of 3 scalars
2: A[0]$ the numbering starts at 0
3: >> scalar entry = 0
4: A.push_back(scalar);A.size$
5: >> scalar entry = 4
6: A.pop_back();A.size$
7: >> scalar entry = 3
```

The PhySet class In Oofelie, everything is based on a base class: the PhySet class. The name of this class comes from the contraction of Physical Set. This is the Oofelie's most important class and it is not possible to access the code without knowing how PhySet works. Most of Oofelie's objects are PhySet objects. For example, all the elements are PhySet, all the materials are PhySet, all the sets (Elemset, Positset, Fixaset, etc) are PhySets. Every PhySet has very practical base functions: for example the kind_of_set() function returns the name of PhySet, the print() function prints out to stdout the contents of PhySet, etc...

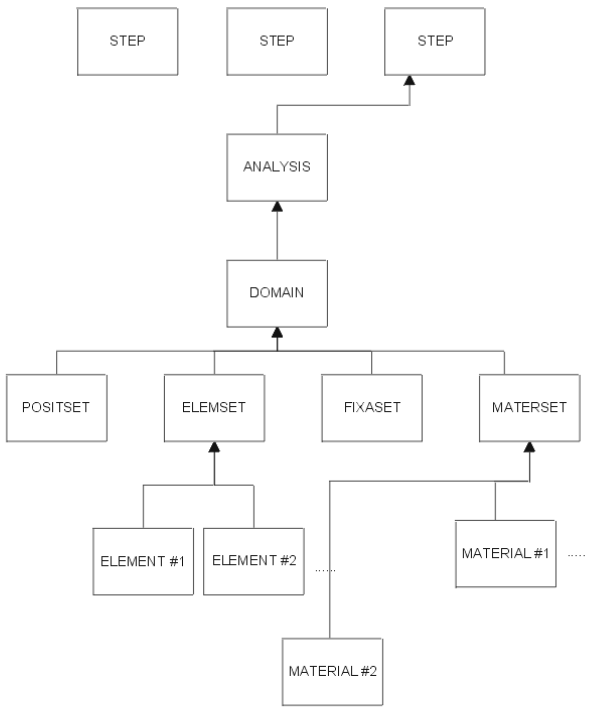
PhySets are only used to be identified: they allow to very freely create links between different Oofelie's objects.

Every PhySet can have one and only one "parent" (there exist the set_pere() function to define it) and a series of properties under a form of a PhySets list (use add_properties() function to define them). For example, an Element can have an Elemset as parent. This Element can also have a Material and a Propelem in its properties list.

This allows to create "the PhySet tree", an image of the Oofelie's memory organization. A classic PhySet tree is shown for example on figure 9.

We see that Domain is the axis of everything here. In fact, Domain is the main class that regroups the set of informations of the problem. Arrows represent the relationships defined by set_pere(). We see that Materset is the parent of two materials, the Elemset is the parent of two elements and the Domain is the parent of the Materset and of the Elemset. The Positset contains a set of positions

Figure 9: Father-Son scheme



((x,y,z) coordinates), for example nodes positions. The Fixaset contains the fixations and the Analysis manages the Step objects. The latter are time steps, load cases, etc... The Steps contain the database. The Analysis is always pointing to the current Step.

Another way of representing the preceding diagram is to indicate the properties of every PhySet. They are represented on figure 10 by an arrow that means "I'm a property of..." :

We remark that on this scheme, the Domain is the axis of everything. Therefore, every physet is a property of the Domain. Finally, there are a commands that should immediately be explained : the $Physet :: get\_properties(PHYSET\_2\_PO)$ command allows any communication within the three ($\_PO$ stands for Physical Object). In the interpretor, this command can be simply called by the $Physet[PHYSET\_2\_PO]$ function. Then, if the database $PHYSET\_2\_PO$ of Physet $Physet\_2$ is not a direct property of the Physet *Physet*, the interpretor asks the parent of *Physet* for the database, and so on.

In brief, The PhySets are the Oofelie's basic objects, they all have specific name and can be identified (ex: Domain). Every PhySet possess a parent and a list of properties (ex: DOMAIN_PO) that allow any object in memory to communicate with each other, using the get_properties(*_PO) or simply the [*_PO] function. This function allows a PhySet to do a request to another PhySet.

The multi-physic database and its possibilities The database is one of the keypoints that make Oofelie multi-physics. First of all because it can be accessed at whatever time and in whatever way. For example we can use previous computation results as new data for re-start an completly different computation. In a more generaly manner, the database may contain every value of the computation. The database is made of a set of objects *Set*. These objects are contained in the "Physet properties" of the *Step* objects. So every *Step* possess a set of *Sets* and, from this fact, a part of the database. The whole database is made of all the Sets of all the Steps, as shown on figure 11. Every *Set* makes up a set of different objects of different types, according to their dimension : scalar(temperature, electric potential), vectorial(displacement, force) or more(constraints))

The **Locks** help to recognize differences between one *Set* and another, i.e. how to distinguish between a forces *Set* and a displacements *Set*. A Lock is represented by a set of 64 bits (or flags)

Figure 10: Properties scheme

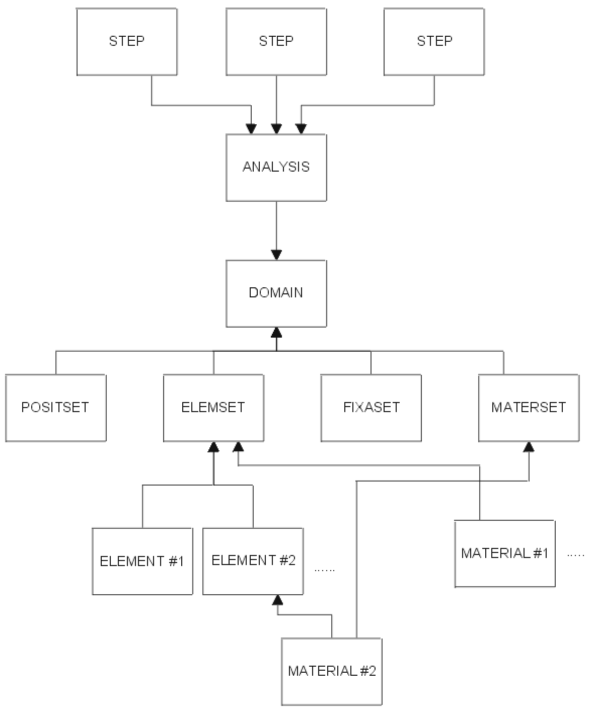
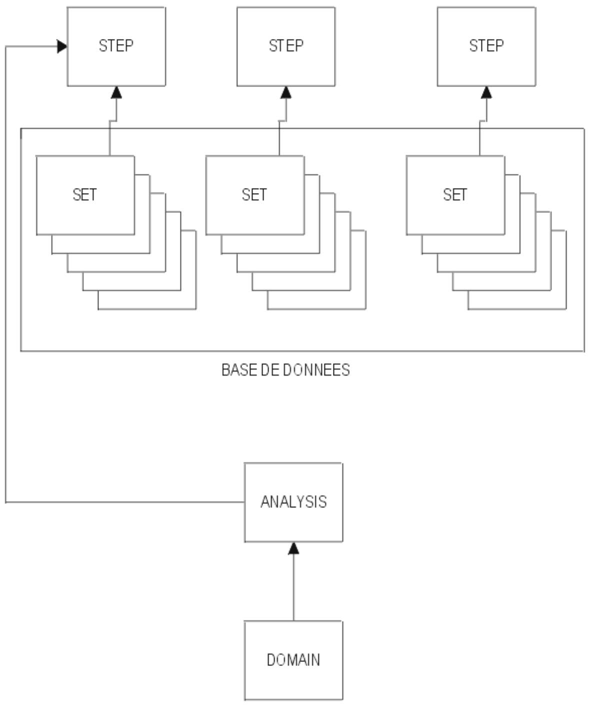


Figure 11: The database



taking the 0 or 1 value. Every bit is represented by a symbol made of two capital letters (TX, TO, GD, IM, etc). Some Lock categories are used to distinguish between different Sets, whereas other Lock categories allow to give attributes and are used at d.o.f management level, see figure 12 for some of the main fields.

Figure 12: Qualification of degrees of freedom

Main Fields		
Nature	: TX, TY, TZ, RX, RY, RZ, TO, EP, LM, TM, PR	
Reference	: AB, RE	Qualification
Complexe	: RL, IM	
Degree	: GD, GV, GA, GF	
Contribution	: I1, I2, I3	
Fixation	: FR, FI, IP	State
Interface	: NI, JN	
Boundary	: BO, IL	
Joker	: J1, J2, J3	

The Nature category gives the Lock type (for example TX for the translation along X coordinate), the Reference category is AB for absolute or RE for relative (allows multi-body management for example), the Degree category is GD for generalized displacement, GV ofr gen. velocity, GA for gen. acceleration, or GF for gen. force. The Fixation category is FR if the d.o.f is free and FI if it is fixed. Finaly one that will be very important for the following : the Interface category may be set to IN for a d.o.f on the interface or NI for a d.o.f not on the interface

The **Keys** are used to define criterions of validity of *Keys* (with some match function) ; it allows activation of multiple characteristics inside each sub-field, two examples out of the FETI.e program : (TX|TY|TZ|INTERFACE|FR).care means that all d.o.f, on the interface or not, provided that they are free, are valid ; (TX|TY|TZ|IN|FR).care means nearly the same, but there they have to be on the interface to be valid. The .care activates one default bit in every other field, which is necessary to obtain a *key*.

All this work on definition of degrees of freedom is used to **partition** the degrees of freedom set, so that we can divide structural components created into different blocks according to the partitioning.

Definition of a Domain. The Model definition must follow this order :

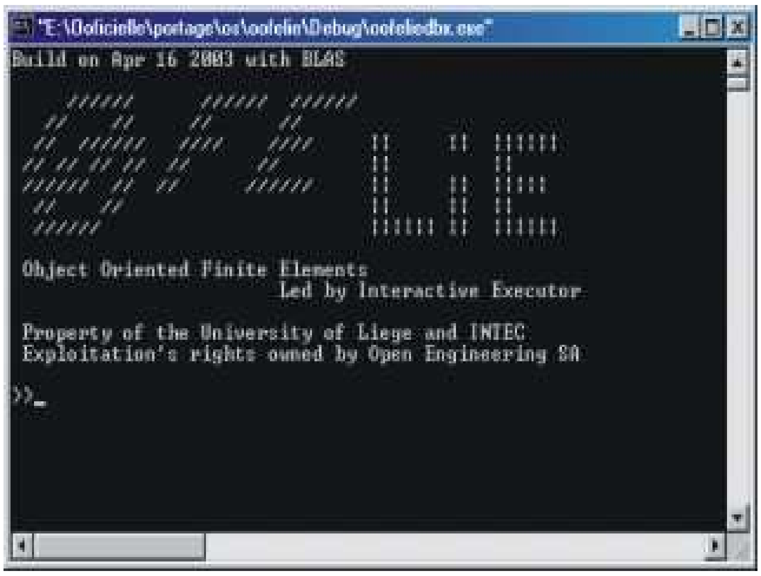
Table 5: Model definition

1: Positions
2: Materials : material or materialset +properties; element property “propelem” +properties+material
3: Elements : propelem+positions
4: Fixations : position+lock
5: Excitations : position+lock+value
6: Domain : add different properties

One remark about the positionset : this is the only *Set* that is also a PhySet, in fact, the *Lock* that corresponds actually to the *Set* PositionSet is (TX|AB|GD).

22 Programmation

Figure 13: Oofelie's command interpretor



```
"E:\Ooficielle\portage\os\oofelie\Debug\oofeliedbx.exe"
Build on Apr 16 2003 with BLAS

    //////     ////// //////
   //    //    //     //
  // //////   ////   ////    !!     !! !!!!!!
 // // // // //     //       !!        !!
////// //  //    //////      !!     !! !!!!!
  //    //                   !!     !! !!
  //////                     !!!!!! !! !!!!!!

Object Oriented Finite Elements
                       Led by Interactive Executor

Property of the University of Liege and INTEC
Exploitation's rights owned by Open Engineering SA

>>_
```

For the virtual parallelization , We decided to use *std* :: *vector* <> of the dimension of the number of subdomains for every PhySet needed : only for the database I did not really use *steps* and *sets*, but I extracted every structural matrix I needed into normal structures, and I used normal vectors to store displacement data. Oofelie has no tool implemented to make different processors work in collaboration such as the MPI (Message Passing Interface) so I tried to make programs the most virtualy parallelized as possible, mainly by the use of a loop on every subdomain as often as possible.

Structure of the project and Compilation As any C++ project, Oofelie must be compiled for the interpretor to take into account new modifications. I used Microsoft Visual C++ Studio to compile as it was the one installed on the faculty computers. It is a good software thanks to which we can easely navigate into the project, search for an expression in every files.

The I_class contents is related to :

- oeI_*class*.h :
  - the class name declaration
  - the eventual declaration of mother/parental classes (where the class to be interpreted comes from)
- oeI_*class*.cpp :
  - the different member functions declaration (with its own parameters)
  - the declaration of an interpreter documentation of all these functions
  - the execution management of member functions (by a big switch)

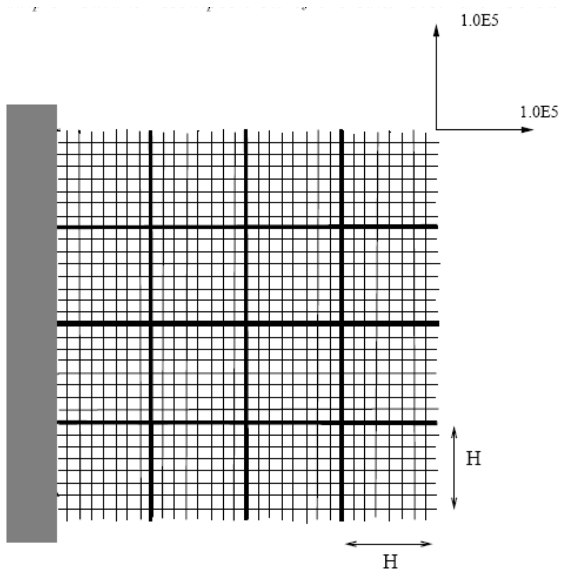
22.1 The first program

At the beginning of this internship, some of the keypoints for domain decomposition were not implemented on Oofelie : no domain partitioner were implemented. For a few weeks I kept on studying bibliography on different methods, and as the open-engineering team whished to implement theirselves a partitioner, I made a first attempt of the FETI algorithm on a simple "chessboard" 2D problem. This is a fully parameterized problem with rectangular elements. It is clamped on one side and some constraints can be imposed on the opposite side. This made me get familiar with the command interpretor avaliable functions, and with the architecture of the C++ project. Indeed, every class has particular commands and their application field may sometimes be limited.

For the model definition, I needed to limit data storage. One of the principles was to be as close as possible of a domain definition strategy that could be used for every FETI-family methods : as every interface degree of freedom may be different for each subdomain, they had to be part of a different set. Furthermore, the positionset should be defined only once and was to be shared by all subdomains.

The first program was made on the base of a Matlab program made by Mr Rixen, which was optimized to run fast on Matlab. I copied this program quite linearly and choose to maintain the use of connectivity matrices B. This lead to too much loops as lots of Matlab function do not exist in Oofelie, and the program was finaly very very slow.

Figure 14: Original partitioned domain for the first program

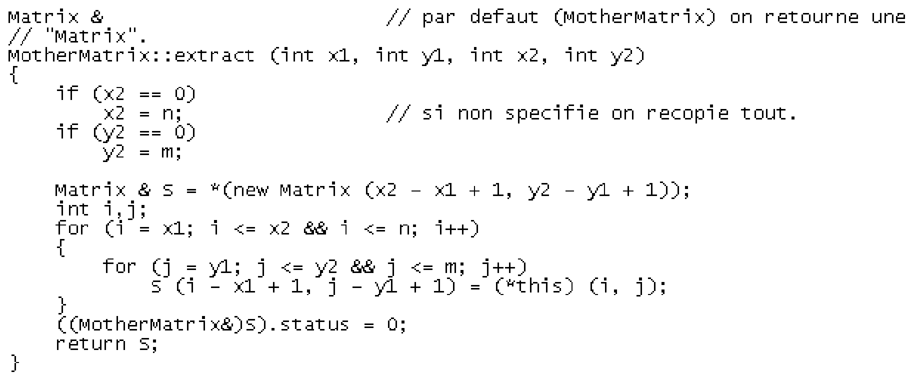


Nevertheless, thanks to this I have understood a lot about Oofelie, about the interpretor in particular but also about programing in the upper levels, and I made implemented some modifications in the project.

A Matrix extraction function At the beginning I didn't knew about partitions, how to label differently degrees of freedom and to extract a particular block from a structural matrix, I implemented a new function available in the interpretor that allows to extract a matrix from a bigger, taking two integer vectors as arguments. Before that, only block extraction was possible, with a min and a max value. This method is very similar to the one implemented in Matlab. This extraction method was implemented in the MotherMatrix class, which is a common parent of every matrix classes. Thus any kind of matrix can une this function.

Figures 15 and 16 are written in the ground level oeMotherMatrix.cpp file, and in order to authorize access to these functions in the interpretor, I also had to modify the second level I_class

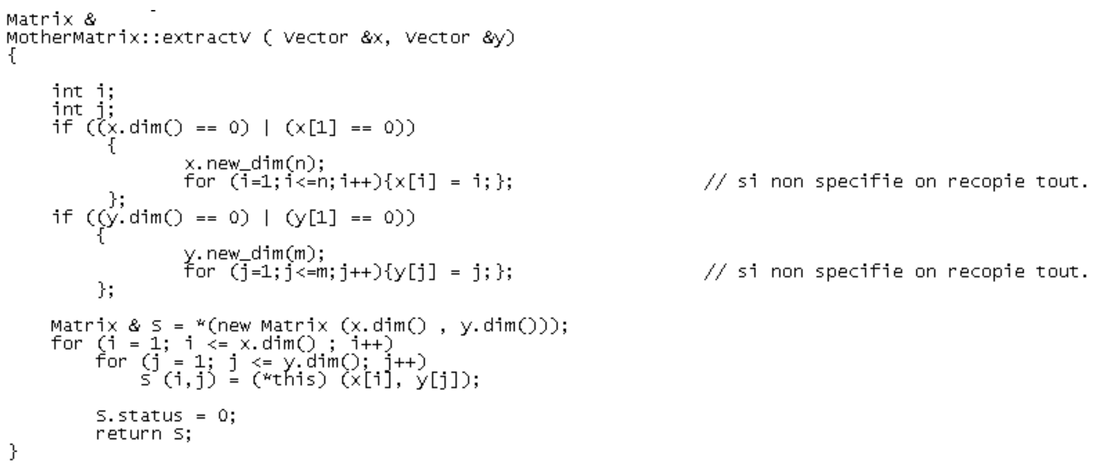
Figure 15: Original extract($scalar, scalar, scalar, scalar$) function



```
Matrix &                        // par defaut (MotherMatrix) on retourne une
// "Matrix".
MotherMatrix::extract (int x1, int y1, int x2, int y2)
{
    if (x2 == 0)
        x2 = n;                 // si non specifie on recopie tout.
    if (y2 == 0)
        y2 = m;

    Matrix & S = *(new Matrix (x2 - x1 + 1, y2 - y1 + 1));
    int i,j;
    for (i = x1; i <= x2 && i <= n; i++)
    {
        for (j = y1; j <= y2 && j <= m; j++)
            S (i - x1 + 1, j - y1 + 1) = (*this) (i, j);
    }
    ((MotherMatrix&)S).status = 0;
    return S;
}
```

Figure 16: Modified extractV($Vector, Vector$) function



```
Matrix &
MotherMatrix::extractV ( Vector &x, Vector &y)
{

    int i;
    int j;
    if ((x.dim() == 0) | (x[1] == 0))
         {
                x.new_dim(n);
                for (i=1;i<=n;i++){x[i] = i;};               // si non specifie on recopie tout.
         };
    if ((y.dim() == 0) | (y[1] == 0))
        {
                y.new_dim(m);
                for (j=1;j<=m;j++){y[j] = j;};               // si non specifie on recopie tout.
        };

    Matrix & S = *(new Matrix (x.dim() , y.dim()));
    for (i = 1; i <= x.dim() ; i++)
        for (j = 1; j <= y.dim(); j++)
            S (i,j) = (*this) (x[i], y[j]);

        S.status = 0;
        return S;
}
```

oeI_MotherMatrix.cpp : a line in the description of the member function list that declares a new function available for the I_MotherMatrix class, and a case in the switch loop of the I_MotherMatrix::exec() function that links the new function available in the interpretor and the one recently added in the ground C++ level.

Making rigid body modes available in the interpretor In the class SymMatrix, the LDL^T factorization was already coded and detected null pivots of singular matrices. A Matrix containing the rigid body modes was also created but it was not available in the interpretor. I needed this information thought for the elaboration of the small coarse grid problem. Thus I had to modify the oeI_SymMatrix.cpp file and it is now possible to extract the rigid body modes matrix, the number of such modes. On the opposite of the Matlab code for the nullspace of a Matrix which seeks for the best d.o.f. to block, the LDL^T factorization is inherently sequential and often the last d.o.f. are considered as fixed.

22.2 The metis partitionner and the DomainPartitioning class

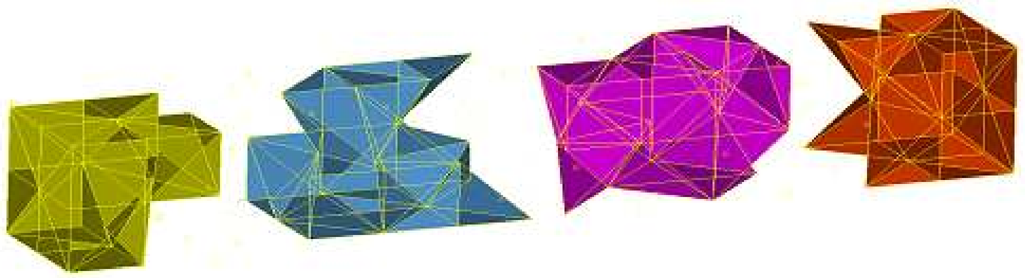
Metis [1] is a freeware package commonly used. It is a graph partitioner and can also be used as a mesh partitioner. There are two methods implemented to partition meshes slightly different : the one minimizes the latter information exchange by minimizing the number of nodes on the interface, the second one just minimizes the number of 2D elements on the interface. The difference of efficacity between them may be discussed in terms of speed of computation but we do not have enough data to make our choice now.

Using this mesh partitioner, Mr Paquay from Open-Engineering created new classes named oeDomainPartitioning and oeI_DomainPartitioning. It is first declared taking as argument the initial global domain, then we set the number of subdomains for the decomposition and finaly the partitioning is performed using Metis. After this, informations about interface are stored in the InterfaceInfos structure and the interface flags are set to IN=1 (on the interface) for corresponding d.o.f while others have the default flag NI set to 1. The commands in the interpretor are as follows :

Table 6: Domain partitioning

```
DomainPartitioning domDec(dom_FETI);
domDec.setNbParts(Ns);
domDec.performDecomposition();
```

Figure 17: dom_FETI domain example partitioned with Metis



The InterfaceInfos is a std::vector<InterfaceNodeInfoByDomain> ; The InterfaceNodeInfoByDomain is a std::vector<InterfaceNodeInfo> ; The InterfaceNodeInfo is a structure that possess an int nodeId, a std::vector<int> domainsIds, and a std::vector<int> localInterfaceIdOfConnectedNode ; All this means that the InterfaceInfos has every information a subdomain could need to rapidly pick from other subdomains the exact information they need to compare any of their interface d.o.f. .

Concretely,

interfaceInfos[$subdomainId$][$LocalInterfaceNodeId$].nodeId() is the identity number of the node to which this d.o.f. belongs in the positionSet.

interfaceInfos[$subdomainId$][$LocalInterfaceNodeId$].domainsIds contains the domainsId of the subdomains that are connected to this interface node.

Thus, interfaceInfos[$subdomainId$][$LocalInterfaceNodeId$].domainsIds.size is equal to the multiplicity of the interface node minus one.

Limitations For the moment, some limitations exist for the decomposition : It works only for Tetra4 elements, and for continuous nodeId and elementId numerotation (starting from 1 to ...) There are also some problems about the repartition of node fixations and excitations which are defined in FixationSet or ExcitationSet by for example the use of the following command : excitationset.define(nodeId,Lock,value); . Indeed, after partitioning, every subdomain has the whole information about fixations and excitations, even if it does not have any elements connected to the excited nodes. This is not really a problem but it appears that data are lost once we try to get the structural vector of generalized forces. To avoid loos of data for a simple domain, one commonly use the special command domain.get_properties(Lock).put_val_NFD(nodeId,Lock,value); but I didn't succeed in getting those values transmitted to subdomains in case of a domain decomposition. To get some constraints for one or more subdomains, one has to modify arbitrarly one of the generalized forces structural vector.

22.3 The final program

The main FETI.e file is composed as follows :

Table 7: The main FETI.e file algorithm and different files related : description of the preprocessing

```
1: $exempledom.e$; //creates the global dom_FETI domain
2: define different parameters;
3: declare variables that are not $Sets$;
4: partition the domain as in table ??
5: $extract\_fact\_local\_op.e$; //extract $K, u, f$
6: $extract\_fact\_int\_op.e$; // extract $K_{bb}; K_{bi}; K_{ii}$; and factorize $K_{ii}$ if needed
7: $comp\_force\_norm.e$; //compute force norm
8: $compute\_k\_scaling.e$; //compute scaling for preconditioner if needed
9: $coarsegrid\_.e$; //build natural coarse grid data
10: $FETI\_initialisation.e$;
11: Preconditioned Conjugate Gradient Iterations
```

But the use of the DomainPartitioning class, The main difference between this program and the first one is that no connectivity matrice B is defined. Thanks to the InterfaceInfos structure, every subdomain have direct acces to the Id of the domains and of their local d.o.f. connected. Most of the functions that the B and the L matrices yield to may be done in a simple a completely parallelized manner. For the assembly operations using B, I made the $LocalScaledAssembly.e$ function. This function is equivalent to the operation $B^{(s)T} \sum_{i=1}^{N_s} B^{(i)}\mathbf{u}^{(i)}$ and is sufficient for nearly everything in the FETI method. The different values may be ponderated, for example if the preconditioner takes into account heterogeneities across the interface. We then should rather store the facing stiffness of the connected domain at the connected node in the preoprocessing to avoid decelerations each time it is needed. I then added to the InterfaceInfos structure the std::vector<double> localFacingStiffTX (and TY and TZ);. For the convergence criterion, I made the $LocalPrimalAssembly.e$ which has no real algebraic equivalence with L or B operations.

Improvements possible For the moment, almost all the specificities of the Oofelie code have been used in this algorithm, but one should still implement this algorithm in a Analysis Class, and this could enable the use of steps to store data in different Sets. In this algorithm, the sets exist but are not used, and data for one step are stored in different colums of a Matrix. But there still is a problem, because for the moment the set stores only one value for each displacement variable and only one value for each force variable. This could fit for primal variables, but as dual variables need one specific value for each subdomain and $multiplicity - 1$ lagrange multipliers values, one should maybe use the "I1toI9 contributions Locks", combinated with the GD lock, and each subdomain should know specificly which one of the variables is his, and maybe the "I1toI9 contributions Locks", combinated with the GF locks, for the lagrange multipliers. This would limit the number of lagrange multipliers and there could never be a node where more than four subdomains could meet : if so, the lagrange multipliers could not be fully redundant.

This problem could be solved with the actual parallelization of the Oofelie code : maybe each processor will posses its own database, then there would be no problem for the displacement variables and the lagrange multipliers would be limited to 9 per subdomain, which would mean that 9 subdomains could meet in one node without any problem, which is enought to test the algorithms on some very simple cubic "3D chessboard" problems.

CONCLUSION

The achieved work To conclude the work I have done, I first have to go back to the differents algorithms and their Indeed, our first work during this internship was to make understand links and differences between different FETI-family methods : The different preconditioners and its mechanical interpretation, The difference between FETI-2 and FETI-DP, How to add average constraints to improve convergence.
After having read a lot of the Oofelie documentation on the community website $Oofelie.org$, and talking about strategies of where to store the different data with Mr Rixen and Mr Paquay, and before that the *DomainPartitioning* class was created, I started to implement FETI in a first manner on the simple chessboard problems. This was the occasion to get used with the command interpretor functions, and to constainsly seek in differents lower levels for the available functions and their meaning. This problems has permitted us to understand a lot about how this code works and how to modify the interpretor. I then implemented a first little Matrix extraction function and modified the LDL^T factor function in order to have acces to some informations in the interpretor.
Once the *DomainPartitioning* class was availabe and that the first program worked, I started the new implementation, avoiding connectivity matrices and using Locks and partitions to separate degrees of freedom and to extract easely Blocks from structural Matrices.
Concerning the objectives that I had at the beginning, that is to say:

- To read and learn as much as possible about the FETI-family algorithms
- To learn a lot about how Oofelie works
- To implement at least the original FETI algorithm

they are all nearly completly reached.

The perspectives As I have mentionned it, There are a lot of perspective for the implementation of FETI methods in the Oofelie program. One has to completely integrate it into an Analysis class and to use Steps and different Sets to save memory. There is also a lot to do in order to take into account average primal variables for the coarse problems of the next algorithms, finaly, the *DomainPartitioning* class should to be modified to distribute the minimum of information to each subdomain, in particular for the excitations. One keypoint in the futur is to make a real paralelized version of these algorithms, with communcations rules between processors and to make some choice about what data they will independently own.

References

[1] *http://glaros.dtc.umn.edu/gkhome/views/metis.*

[2] J. Mandel C. Farhat and F.X. Roux. Optimal convergence properties of the feti domain decomposition method. *Comput. Methods Appl. Mech. Engrg.*, 115:367–388, 1994.

[3] Charbel Farhat and Po-Shu Chen. Tailoring domain decomposition methods for efficient parallel coarse grid solution and for systems with many right hand sides. *Contemporary mathematics*, 180:401–406, 1994.

[4] Charbel Farhat, Po-Shu Chen, and Jan Mandel. A new coarsening operator for the optimal preconditioning of the dual and primal domain decomposition methods: application to problems with severe coefficient jumps. In *Proc. Copper Mountain Conference on Multigrid Methods*, April 3-7 1995.

[5] Charbel Farhat, Po-Shu Chen, and Jan Mandel. A scalable lagrange multiplier based domain decomposition method for implicit time-dependent problems. *Int. J. Numer. Methods Engrg.*, 38:3831–3854, 1995.

[6] Charbel Farhat, Po-Shu Chen, Jan Mandel, and Francois Xavier Roux. The two-level FETI method part II: Extension to shell problems, parallel implementation and performance results. *Computational Methods in Applied Mechanical Engineering*, 155:153–179, 1998.

[7] Charbel Farhat, L. Crivelli, and F.X. Roux. Extending substructure based iterative solvers to multiple load and repeated analyses. *Computational Methods in Applied Mechanical Engineering*, 117:195–209, 1994.

[8] Charbel Farhat and M. Geradin. On a component mode synthesis method and its application to incompatible substructures. *Comput. Struct.*, 51:459–473, 1994.

[9] Charbel Farhat, Michel Lesoinne, Patrick LeTallec, Kendall Pierson, and Daniel Rixen. FETI–DP : a dual–primal unified feti method part i : A faster alternative to the two-level FETI method. *international journal for numerical methods in engineering*, 50:1523–1544, 2001. Tech. Rep. CU-CAS-99-15, Center for Aerospace Structures, University of Colorado at Boulder, August 1999.

[10] Charbel Farhat, Michel Lesoinne, and Kendall Pierson. A scalable dual–primal domain decomposition method. *Numerical Linear Algebra with Applications*, 7:687–714, 2000.

[11] 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. *Computational Methods in Applied Mechanical Engineering*, 155:129–151, 1998.

[12] Charbel Farhat and François-Xavier Roux. An unconventional domain decomposition method for an efficient parallel solution of large scale finite element systems. *SIAM J. Sci. Stat. Comput.*, 13:379–396, 1992.

[13] Axel Klawonn and Oliver Rheinbach. A parallel implementation of dual-primal FETI methods for three-dimensional linear elasticity using a transformation of basis. *Society for industrial and applied mathematics j. sci. comput.*, 28(5):1886–1906, 2006. AMS subject classification 65F10 65N30 65N55 ; DOI 10.1137/050624364.

[14] Axel Klawonn and Oliver Rheinbach. Robust FETI–DP methods for heterogeneous three dimensional elasticity problems. *computational methods for applied mechanical engineering*, 196:1400–1414, 2007.

[15] Axel Klawonn and Olof B. Widlund. FETI and neumann-neumann iterative substructuring methods: Connections and new results. *Communications on Pure and Applied Mathematics*, 54:57–91, 2001.

[16] Axel Klawonn and Olof B. Wildund. Dual–primal FETI mathods for linear elasticity. *TR2004-855*, 2004.

[17] Michel Lesoinne. A FETI–DP corner selection algorithm for three-dimensional problems. In *Fourteenth International Conference on Domain Decomposition Methods*, number 19, pages 217–224, 2003.

[18] Michel Lesoinne and Kendall Pierson. FETI–DP : An efficient, scalable and unified dual-primal (feti) method. In *Twelveth International Conference on Domain Decomposition Methods*, number 44, pages 221–228, 2001.

[19] Jing Li and Olof Widlund. FETI-DP, BDDC, and block Cholesky methods. *TR2004-857*, 2004.

[20] J. Mandel and R. Tezaur. Convergence of a substructuring method with lagrange multipliers. *Numerische Mathematik*, 73:473–487, 1996.

[21] Jan Mandel and Radek Tezaur. On the convergence of a dual-primal substructuring method. 2000.

[22] Jan Mandel, Radek Tezaur, and Charbel Farhat. A scalable substructuring method by lagrange multipliers for plate bending problems. *SIAM Journal on Numerical Analysis*, 1998.

[23] Daniel J. Rixen. Computational methods : Parallel processing.

[24] Daniel J. Rixen and Charbel Farhat. A simple and efficient extension of a class of substrubturing based preconditioners to heterogeneous structural mechanics problems. *INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING*, 44:489–516, 1999.

[25] DANIEL J. RIXEN, CHARBEL FARHAT, RADEK TEZAUR, and JAN MANDEL. Theoretical comparison of the FETI and algebraically partitioned FETI methods, and performance comparisons with a direct sparse solver. *Int. J. Numer. Meth. Engng.*, 46:501–533, 1999.