

VSB – Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Applied Mathematics

Parallel input data generation in FETI methods

Paralelizace generování vstupních dat ve FETI metodách

2014

Pavla Jirůtková

Zadání diplomové práce

Student: **Bc. Pavla Jirůtková**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 1103T031 Výpočetní matematika

Téma: **Paralelizace generování vstupních dat ve FETI metodách**
Parallel input data generation in FETI methods

Zásady pro vypracování:

FETI (Finite Element Tearing and Interconnecting) metody rozložení oblastí jsou vysoce účinné algoritmy k masivní paralelizaci celé řady úloh výpočetní mechaniky. Jejich hlavní myšlenka spočívá v rozdělení úlohy definované na původní oblasti na úlohy menší definované na jednotlivých podoblastech, které lze vyřešit paralelně a výsledná dílčí řešení slepit pomocí Lagrangeových multiplikátorů. Jedním z velkých problémů ovšem stále zůstává efektivní paralelní generování vstupních dat pro případ složitých geometrií výpočetních oblastí.

Součástí práce bude hlubší nastudování metody konečných prvků a efektivní implementace FETI metod. Předpokládá se zapojení do týmu v rámci IT4I zabývajícím se vývojem knihovny škálovatelných algoritmů kvadratického programování FLLOP a zaměření se na knihovnu LibMesh podporující paralelní generování vstupních dat. Hlavním cílem pak bude přizpůsobit knihovnu LibMesh pro import ze zvolených komerčních balíčků a vytvořit rozhraní ke knihovně FLLOP. Díky těmto modifikacím bude možno počítat rozsáhlé úlohy o 100 milionech až miliardách neznámých.

Seznam doporučené odborné literatury:

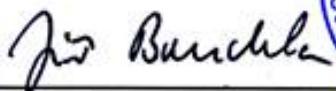
Podle doporučení školitele.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **prof. Ing. Tomáš Kozubek, Ph.D.**

Datum zadání: 01.09.2013

Datum odevzdání: 07.05.2014



doc. RNDr. Jiří Bouchala, Ph.D.
vedoucí katedry



prof. RNDr. Václav Snášel, CSc.
děkan fakulty

I declare that I have worked up this thesis by myself. I have referenced all the sources and publications I have used.

Ostrava, May 7, 2014

.....

I would like to express my thanks to prof. Ing. Tomáš Kozubek, Ph.D., the leader of my diploma thesis, for very interesting and inspiring theme of the thesis and for supervision of my work.

My thanks also belong to Ing. Oldřich Vlach, Ph.D. for helping me with libMesh library and for his helpful advice and suggestions. I would also like to thank Ing. Marta Jarošová, Ph.D. for her worthwhile suggestions and recommendations about FETI methods and for helping me with English.

Furthermore, I would like to thank Ing. Václav Hapla and Ing. Alexandros Markopoulos, Ph.D. for their cooperation at the connection of the libMesh and FLLOP library.

This thesis has been elaborated with the support of the project New creative teams in priorities of scientific research, reg. no. CZ.1.07/2.3.00/30.0055, supported by Operational Programme Education for Competitiveness and co-financed by the European Social Fund and the state budget of the Czech Republic.

This thesis was supported by the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070), funded by the European Regional Development Fund and the national budget of the Czech Republic via the Research and Development for Innovations Operational Programme, as well as Czech Ministry of Education, Youth and Sports via the project Large Research, Development and Innovations Infrastructures (LM2011033).

This thesis is also partially supported by Grant of SGS No. SP2014/204, VŠB - Technical University of Ostrava, Czech Republic.

Abstrakt

V této práci se zabýváme paralelním generováním vstupních dat pro FETI (Finite Element Tearing and Interconnecting) metody. Ve FETI metodách je původní úloha, definovaná na rozsáhlé oblasti, rozdělena na řadu menších úloh definovaných na jednotlivých podoblastech, které jsou pak řešeny paralelně. Pro efektivní paralelní generování vstupních dat používáme knihovnu `libMesh`. Tato softwarová knihovna je nástroj využívající metodu konečných prvků pro numerické simulace parciálních diferenciálních rovnic definovaných na diskretizované oblasti (síti) a spustitelných sériově nebo paralelně. Naše implementace generování dat je připojena ke knihovně FLLOP a numerické experimenty jsou pak počítány na superpočítači Anselm. Škálovatelnost naší implementace je testována na akademické úloze lineární elasticity.

Klíčová slova: FETI-DP, libMesh, metoda konečných prvků, Newmarkova metoda, paralelní implementace, problém elasticity, škálovatelnost, TFETI

Abstract

In this diploma thesis we deal with an effective parallel input data generation for FETI (Finite Element Tearing and Interconnecting) methods. In FETI methods the original problem, defined on a large body, is partitioned into a number of smaller problems defined on the subdomains. These smaller problems are solved in parallel. To generate the input data effectively in parallel we use the `libMesh` library. This open-source software library is a preprocessing tool for the numerical mesh-based simulation of partial differential equations on serial and parallel platforms, using the finite element method. Our implementation of data generation is connected to FLLOP library to perform the numerical experiments on Anselm supercomputer. The scalability of our implementation is tested on the academic benchmark from the linear elasticity.

Keywords: FETI-DP, libMesh, finite element method, Newmark method, parallel implementation, elasticity problem, scalability, TFETI

List of Used Abbreviations and Symbols

QP	– Quadratic Programming
FETI	– Finite Element Tearing and Interconnecting
$C^n(\Omega)$	– Set of all n times continuously differentiable maps on Ω
\mathbb{R}	– Set of all real numbers
\mathbb{R}^d	– Set of all d -dimensional real vectors
I	– Identity matrix
O	– Null matrix
o	– Null vector
∇	– Gradient

Contents

1	Introduction	4
2	Elasticity problem	5
2.1	Weak formulation	5
2.2	Finite element discretization	6
2.3	Assembling the mass and the stiffness matrix and the load vector	10
2.4	Discretization in time	14
2.5	Static elasticity problem	16
3	FETI methods	17
3.1	TFETI (FETI)	17
3.2	FETI-DP	23
4	Parallel input data generation by the libMesh library	27
4.1	LibMesh library	27
4.2	Key data structures	27
4.3	Description of the implementation	30
5	Numerical experiments	39
6	Conclusion	44
7	References	45

List of Tables

5.1	Numerical (weak) scalability for ≈ 2500 elements of the subdomain.	40
5.2	Numerical (weak) scalability for ≈ 10000 elements of the subdomain.	40
5.3	Numerical (weak) scalability for ≈ 22500 elements of the subdomain.	40
5.4	Parallel (strong) scalability for 1000000 elements.	41
5.5	Parallel (strong) scalability for 4000000 elements.	42

List of Figures

2.1	Model problem.	5
2.2	Finite element discretization of the domain in space.	7
2.3	Detail of the domain discretization (zoomed part of the Figure 2.2).	7
2.4	Base function corresponding to the part of the Figure 2.3 and shape function (dark grey area) corresponding to the marked part of the base function. . .	9
2.5	Reference triangle element.	12
2.6	Gauss points qp of the triangle of different polynomial degrees p	13
2.7	The constant average acceleration scheme [9].	15
3.1	Domain decomposition by TFETI [13].	18
3.2	The block structure of matrices and vectors of primal formulation [13], [12].	19
3.3	Domain decomposition by FETI-DP with all types of variables.	23
4.1	The example of mesh and <code>pid_mesh</code> for 16 elements and 4 processors.	28
4.2	The <code>Elem</code> class hierarchy.[26]	29
4.3	The <code>System</code> class hierarchy. [27]	30
4.4	Scheme of the implementing flow.	31
5.1	Geometry for the given example.	39
5.2	Number of iterations.	41
5.3	Total time and preprocessing time for 1000000 elements.	42
5.4	Total time and preprocessing time for 4000000 elements.	42
5.5	Time of the <code>assemble_elasticity</code> function.	43

1 Introduction

The FETI domain decomposition methods are ranked among an efficient algorithms for a massively parallelization of many problems of a computational mechanics. These methods are based on a principle of „divide and conquer“. It means the original problem solved on a large body is partitioned into a number of smaller problems defined on the nonoverlapping subdomains. Thanks to that the original problem can be solved in parallel. The partial solutions are then glued via Lagrange multipliers. However the one of great problems, an efficient parallel input data generation for large and complicated geometries of computational bodies, still remains to solve. The main goals of this thesis is to implement this input data generation in `libMesh` library for the elasticity problem on mesh imported from selected commercial software packages and make an interface to `FLLOP` library.

Now we look at an organization of sections in the thesis and at contents of these sections. In Section 2 at first we introduced the dynamic elasticity problem and its weak formulation. The next part of this section deals with the spatial discretization by the finite element method, while the time dimension remains continuous. We describe an assembling the mass and the stiffness matrix and the load vector, whereat we use the concept of the reference element and the Gaussian quadrature rule. After the spatial discretization we present the time discretization by Newmark method. The last part describes the static elasticity problem in brief.

In Section 3 we show two FETI methods - the Total-FETI (FETI) method and the FETI-DP method. In both cases the domain is decomposed into nonoverlapping subdomains and the continuity of displacement on the intersubdomain interface is enforced via Lagrange multipliers. In first part we describe the TFETI (FETI) method and its primal and dual problem. In second part we present FETI-DP method and its differences in comparison to TFETI (FETI).

The main part of the thesis is the Section 4, where we deal with the `libMesh` library and describe our implementation. First we shortly present the open-source software library `libMesh` developed in C++. In the next part the description of some key data structures used in library follows. The last part deals with the description of main functions of our implementation. There we derive the process of the assembling the stiffness matrix in the `libMesh`.

Finally in the last Section 5 we present results of the parallel and numerical scalability of our implementation for the given example of a static linear elasticity problem. For numerical experiments the Anselm supercomputer is used.

2 Elasticity problem

We will study the dynamic elasticity problem illustrated in Figure 2.1. Let us consider an elastic body represented (at time $t = 0$) by a bounded domain $\Omega \subset \mathbb{R}^d$, $d = 2, 3$ with the Lipschitz boundary $\partial\Omega$. This boundary consists of two non-overlapping parts Γ_D and Γ_N such that

$$\begin{aligned}\partial\Omega &= \bar{\Gamma}_D \cup \bar{\Gamma}_N, \\ \bar{\Gamma}_D \cap \bar{\Gamma}_N &= \emptyset.\end{aligned}$$

The Γ_D represents the Dirichlet boundary, the part of the boundary with prescribed displacements, and the Γ_N represents the Neumann boundary, the part with prescribed tractions $g : \Gamma_N \rightarrow \mathbb{R}^d$. The body forces are denoted by $f : \Omega \rightarrow \mathbb{R}^d$ and the unknowns displacements by $u : \Omega \times \mathbb{T} \rightarrow \mathbb{R}^d$. In this section we will follow especially [1] and [2], for more information see also [3].

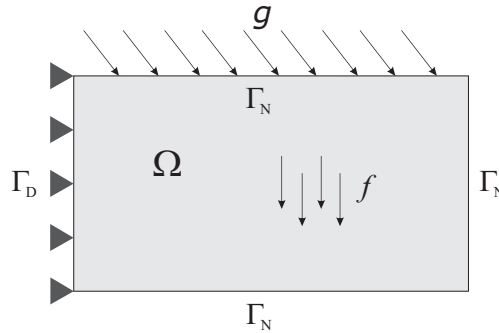


Figure 2.1: Model problem.

2.1 Weak formulation

In this section we introduce the weak formulation of the elasticity problem. First let us define for all times $t \in \mathbb{T} := [0, T]$ the solution set

$$\mathcal{U}(t) := \left\{ u \in H^1(\Omega)^d : \text{Tr}(u) = u_D(t) \text{ on } \Gamma_D \right\}$$

and the space of test functions

$$\mathcal{V} := \left\{ v \in H^1(\Omega)^d : \text{Tr}(v) = 0 \text{ on } \Gamma_D \right\},$$

where $H^1(\Omega)$ denotes the Sobolev space (also denoted as $W^{1,2}(\Omega)$), u_D denotes the prescribed solution on the Dirichlet boundary and $\text{Tr} : H^1(\Omega)^d \rightarrow L^2(\partial\Omega)^d$ is trace operator. The weak formulation of the dynamic elasticity problem can be expressed in the form

$$\begin{cases} \text{Find } u : \mathbb{T} \rightarrow H^1(\Omega)^d \text{ such that } u(\cdot, t) \in \mathcal{U}(t) \quad \forall t \in \mathbb{T} \text{ and} \\ m(\ddot{u}, v) + a(u, v) = f(v), \quad \forall v \in \mathcal{V} \quad \forall t \in \mathbb{T}, \\ u(x, 0) = u_0, \quad \dot{u}(x, 0) = \dot{u}_0, \quad \forall x \in \Omega, \end{cases} \quad (2.1)$$

with the bilinear forms

$$m(\ddot{u}, v) = \int_{\Omega} \rho \ddot{u} v \, dx, \quad (2.2)$$

$$a(u, v) = \int_{\Omega} \sigma(u) : \varepsilon(v) \, dx \quad (2.3)$$

and the linear form

$$f(v) = \int_{\Omega} f v \, dx + \int_{\Gamma_N} g v \, ds. \quad (2.4)$$

The $\rho \geq 0$ is a material density, the f denotes the body forces, the g denotes the forces on Γ_N and the terms σ and ε are the second-order tensors. The σ denotes the stress tensor and the ε denotes the strain tensor. The linear relationship between them is described by Hooke's law of linear elasticity

$$\sigma(u) = C : \varepsilon(u), \quad (2.5)$$

where C is the fourth-order stiffness tensor and $\varepsilon(u) = \frac{1}{2} (\nabla u + (\nabla u)^\top)$. By using summation convention we can write

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

where

$$\varepsilon_{kl} = \frac{1}{2} \left(\frac{\partial u_k}{\partial x_l} + \frac{\partial u_l}{\partial x_k} \right),$$

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

The formula for C holds for homogeneous isotropic material. The constants λ and μ are the Lamé coefficients and the δ_{ij} is the Kronecker's delta, i.e., the function with value 1 if $i = j$, and with 0 otherwise.

2.2 Finite element discretization

For numerical computation of dynamic elasticity problem, the weak formulation (2.1) have to be discretized. First we will make a discretization in space, while the time dimension remains continuous.

Let us consider the polygonal domain $\Omega \subset \mathbb{R}^d$, $d = 2$, and split it into finite number of non-overlapping elements $\Omega_{(el)}$, the triangles in this section, see Figure 2.2. The elements can generally be polygons (in 2D are used only triangles and quadrilaterals), in this section we will talk only about triangles, but the reader should have in mind that the description here can be generalized. The set of all triangular elements is denoted $\tilde{\mathcal{T}} := \{\Omega_{(el)}\}_{el=1}^{n^{el}}$, where n^{el} is the number of the elements. The number of the nodes will be denoted by n^{nod} and the number of the nodes without nodes lying on the Dirichlet boundary will be

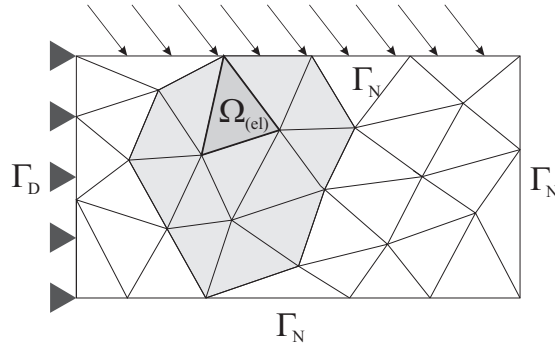


Figure 2.2: Finite element discretization of the domain in space.

denoted by n^{Vnod} . The local numbering of nodes is $x^{[1]}, x^{[2]}, x^{[3]}$ counterclockwise for every element and the global numbering is presented by $x^{(1)}, \dots, x^{(n^{nod})}$, see Figure 2.3. We will denote $\mathcal{D} = \{i \in \{1, \dots, n^{nod}\}, x^{(i)} \in \bar{\Gamma}_D\}$ the set of indices of the nodes lying on the Dirichlet boundary. The displacement u is approximated by polynomials \tilde{P} on triangles.

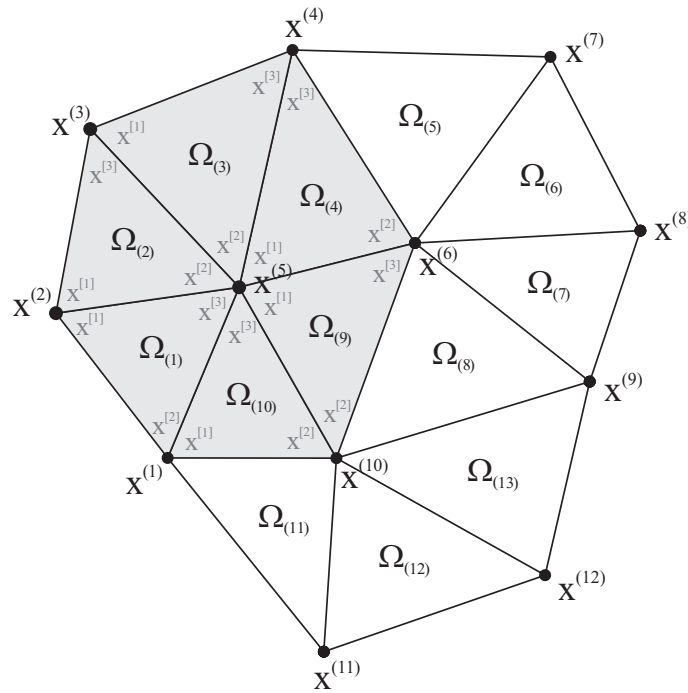


Figure 2.3: Detail of the domain discretization (zoomed part of the Figure 2.2).

The solution set $\mathcal{U}(t)$ is discretized by set $\mathcal{U}^h(t)$, which contains piecewise linear functions $u_k^h(x, t)$ over $\tilde{\mathcal{T}}$ satisfying prescribed Dirichlet boundary conditions and representing the displacement in direction of the x_k , $k = 1, 2$. The discretization of the space $H^1(\Omega)$

and \mathcal{V} is defined by

$$\begin{aligned}\mathcal{V}_{H^1(\Omega)}^h &= \left\{ v^h \in C(\bar{\Omega}) : \forall \Omega_{(el)} \in \tilde{\mathcal{T}} : v^h|_{\Omega_{(el)}} \in \tilde{P}(\Omega_{(el)}) \right\}, \\ \mathcal{V}^h &= \left\{ v^h \in \mathcal{V}_{H^1(\Omega)}^h : v^h = 0 \text{ on } \Gamma_D^h \right\}.\end{aligned}$$

Now we can express the finite dimensional spaces $\mathcal{V}_{H^1(\Omega)}^h$ and \mathcal{V}^h as the sets of all linear combinations of the basis functions, i.e., the linear spans of the basis functions

$$\begin{aligned}\mathcal{V}_{H^1(\Omega)}^h &= \text{span}_{i=1, \dots, n^{nod}} \{ \varphi_i \}, \\ \mathcal{V}^h &= \text{span}_{\substack{i=1, \dots, n^{nod} \\ i \notin \mathcal{D}}} \{ \varphi_i \},\end{aligned}$$

where $\varphi_i \in \mathcal{V}_{H^1(\Omega)}^h$ and $\varphi_i(x^{(j)}) = \delta_{ij}$ for all $x^{(j)} \in \mathbb{R}^2$, $j = 1, \dots, n^{nod}$, is the basis function at the i -th node, see example in Figure 2.4a. Thanks to the $u_k^h(x, t) \in \mathcal{U}^h(t) \subset \mathcal{V}_{H^1(\Omega)}^h$ we can write $u_k^h(x, t)$ in the form

$$u_k^h(x, t) = \sum_{i=1}^{n^{nod}} u_{g_{k,i}}(t) \varphi_i(x), \quad (2.6)$$

where $u_{g_{k,i}}(t) \in \mathbb{R}$ are unknowns of the function $u_k^h(x, t)$ at the nodes $x^{(i)}$. Let us choose the basis functions $\varphi_i \in \mathcal{V}^h$, $i \in \{1, \dots, n^{nod}\} \setminus \mathcal{D}$, as the test functions $v^h \in \mathcal{V}^h$. Substituting that and $u^h(x, t) = (u_1^h(x, t), u_2^h(x, t))^T$ to the weak formulation of the problem we obtain the system of linear equations with solution vector

$$u_g(t) = (u_{g_{1,1}}(t), u_{g_{2,1}}(t), \dots, u_{g_{1,n^{nod}}}(t), u_{g_{2,n^{nod}}}(t))^T. \quad (2.7)$$

The discrete initial displacement vector is define as

$$u_{g_0} = \left(u_0(x^{(1)}), \dots, u_0(x^{(n^{nod})}) \right)^T, \quad u_0(x^{(i)}) \in \mathbb{R}^2.$$

The assembling of the matrices and vectors we will perform over individual elements. For each element $\Omega_{(el)}$ we have three vertices $x^{(i)}, x^{(j)}, x^{(k)} \in \mathbb{R}^2$, where $i, j, k \in \{1, \dots, n^{nod}\}$. These vertices we will describe in local numbering (el, v) , $v \in \{1, 2, 3\}$, such that $x^{[1]} = x^{(i)}$, $x^{[2]} = x^{(j)}$, $x^{[3]} = x^{(k)}$. Over each element $\Omega_{(el)} \in \tilde{\mathcal{T}}$ we define the local basis functions $N_{el,v}$ as a restriction of the φ_i over $\Omega_{(el)}$.

In engineering books are these local basis functions called shape functions and holds $N_{el,v}(x^{[w]}) = \delta_{vw}$, see Figure 2.4b. The form of the basis functions for the one triangular element $\Omega_{(el)}$ with three vertices $x^{[w]} = (x_1^{[w]}, x_2^{[w]})$ is following [4]

$$N_{el,v}(x^{[w]}) := \frac{a_v + b_v x_1^{[w]} + c_v x_2^{[w]}}{2\Delta}, \quad v, w = 1, 2, 3, \quad (2.8)$$

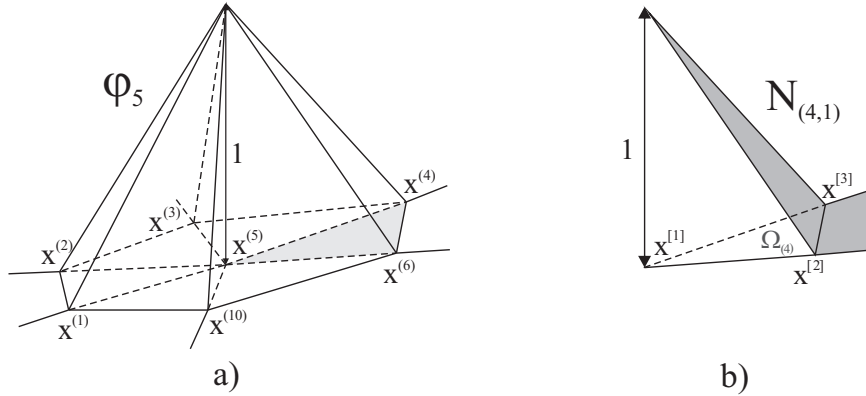


Figure 2.4: Base function corresponding to the part of the Figure 2.3 and shape function (dark grey area) corresponding to the marked part of the base function.

where¹

$$\begin{aligned} a_v &= x_1^{[v+1]} x_2^{[v+2]} + x_1^{[v+2]} x_2^{[v+1]}, \\ b_v &= x_2^{[v+1]} - x_2^{[v+2]}, \\ c_v &= x_1^{[v+2]} - x_1^{[v+1]} \end{aligned}$$

and

$$\Delta = \frac{1}{2} \begin{vmatrix} 1 & x_1^{[1]} & x_2^{[1]} \\ 1 & x_1^{[2]} & x_2^{[2]} \\ 1 & x_1^{[3]} & x_2^{[3]} \end{vmatrix}$$

define the element area. Since the element has three nodes, linear approximation of displacement $u(x, t) = (u_1(x, t), u_2(x, t))^T$ over $\Omega_{(el)}$ can be define by

$$u(x, t)|_{\Omega_{(el)}} \approx u^h(x, t)|_{\Omega_{(el)}} = \sum_{v=1}^3 N_{el,v}(x) I_d \hat{u}_{el,v}(t), \quad (2.9)$$

where I_d is the identity matrix of order d (in this case $d = 2$). The coefficients $\hat{u}_{el,v}(t) = (\hat{u}_{1,el,v}(t), \hat{u}_{2,el,v}(t))^T$ represent the displacement of the vertices of the element $\Omega_{(el)}$ in direction of the coordinate axis. It remains to define the approximation of the test functions $v(x, t) = (v_1(x, t), v_2(x, t))^T$ over $\Omega_{(el)}$ as

$$v^h(x, t)|_{\Omega_{(el)}} = \sum_{v=1}^3 N_{el,v}(x) I_d \hat{v}_{el,v}(t). \quad (2.10)$$

¹ $[k], k \in \mathbb{N} : [k] = ((k - 1) \bmod 3) + 1; [v + 1] \mapsto 2, 3, 1$ and $[v + 2] \mapsto 3, 1, 2$

2.3 Assembling the mass and the stiffness matrix and the load vector

Let us discretize the terms of the weak formulation, i.e. (2.2), (2.3), (2.4). Note that the integral over Ω can be split into the sum of integrals over all $\Omega_{(el)} \in \tilde{\mathcal{T}}$.

$$m(\ddot{u}^h, v^h) = \int_{\Omega^h} \rho \ddot{u}^h v^h dx = \sum_{el=1}^{n^{el}} \int_{\Omega_{(el)}} \rho \ddot{u}_{|\Omega_{(el)}}^h \cdot v_{|\Omega_{(el)}}^h dx, \quad (2.11)$$

$$a(u^h, v^h) = \int_{\Omega^h} \varepsilon(u^h) : C : \varepsilon(v^h) dx = \sum_{el=1}^{n^{el}} \int_{\Omega_{(el)}} \varepsilon(u_{|\Omega_{(el)}}^h) : C : \varepsilon(v_{|\Omega_{(el)}}^h) dx, \quad (2.12)$$

$$f(v^h) = \sum_{el=1}^{n^{el}} \left(\int_{\Omega_{(el)}} f^h(x) v^h_{|\Omega_{(el)}} dx + \sum_{face_{el} \subset \Gamma_N} \int_{face_{(el)}} g^h(x) v^h_{|\Omega_{(el)}} ds \right). \quad (2.13)$$

Substituting the approximation (2.9) and (2.10) to the (2.11), (2.12) and (2.13) we obtain

$$\begin{aligned} m(\ddot{u}^h, v^h) &= \sum_{el=1}^{n^{el}} \int_{\Omega_{(el)}} \rho \left(\sum_{v=1}^3 N_{el,v}(x) I_d \ddot{u}_{el,v}(t) \right) \cdot \left(\sum_{w=1}^3 N_{el,w}(x) I_d \hat{v}_{el,w}(t) \right) dx, \\ a(u^h, v^h) &= \sum_{el=1}^{n^{el}} \int_{\Omega_{(el)}} \varepsilon \left(\sum_{v=1}^3 N_{el,v}(x) I_d \hat{u}_{el,v}(t) \right) : C : \varepsilon \left(\sum_{w=1}^3 N_{el,w}(x) I_d \hat{v}_{el,w}(t) \right) dx, \\ f(v^h) &= \sum_{el=1}^{n^{el}} \left(\int_{\Omega_{(el)}} \left(f^h(x) \cdot \sum_{v=1}^3 N_{el,v}(x) I_d \hat{v}_{el,v}(t) \right) dx + \right. \\ &\quad \left. + \sum_{face_{el} \subset \Gamma_N} \int_{face_{(el)}} \left(g^h(x) \cdot \sum_{v=1}^3 N_{el,v}(x) I_d \hat{v}_{el,v}(t) \right) ds \right). \end{aligned}$$

The reader should have in mind that the upper limit of the sum over vertices is different for different than triangular elements. Now we denote $\bar{N}_{el,v}(x) := N_{el,v}(x) I_d$ and $\bar{N}_{el,w}(x) := N_{el,w}(x) I_d$ and write

$$\begin{aligned} m(\ddot{u}^h, v^h) &= \sum_{el=1}^{n^{el}} \sum_{v=1}^3 \sum_{w=1}^3 \hat{v}_{el,w}(t) \left(\ddot{u}_{el,v}(t) \right)^\top \int_{\Omega_{(el)}} \rho \bar{N}_{el,v}^\top(x) \bar{N}_{el,w}(x) dx = \\ &= \sum_{el=1}^{n^{el}} \hat{v}_{el}(t) M_{el} \ddot{u}_{el}(t), \\ a(u^h, v^h) &= \sum_{el=1}^{n^{el}} \sum_{v=1}^3 \sum_{w=1}^3 \hat{v}_{el,w}(t) \left(\hat{u}_{el,v}(t) \right)^\top \int_{\Omega_{(el)}} \left(\varepsilon(\bar{N}_{el,v}(x)) \right)^\top : C : \varepsilon(\bar{N}_{el,w}(x)) dx = \\ &= \sum_{el=1}^{n^{el}} \hat{v}_{el}(t) K_{el} \hat{u}_{el}(t), \end{aligned}$$

$$f(v^h) = \sum_{el=1}^{n^{el}} \sum_{v=1}^3 \left((\hat{v}_{el,v}(t))^\top \int_{\Omega_{(el)}} f^h(x) \bar{N}_{el,v}(x) dx + \sum_{face_{el,\Gamma_N} \subset \Gamma_N} (\hat{v}|_{face_{el,\Gamma_N}}(t))^\top \int_{face_{el,\Gamma_N}} g^h(x) \bar{N}_{el,v}(x) ds \right) = \sum_{el=1}^{n^{el}} \left(F_{el} \hat{v}_{el}(t) + \sum_{face_{el,\Gamma_N} \subset \Gamma_N} F_{face_{el,\Gamma_N}} \hat{v}|_{face_{el,\Gamma_N}}(t) \right),$$

where M_{el} , K_{el} , F_{el} denote the element (local) mass matrix, the element (local) stiffness matrix and the element (local) load vector, respectively. The $\hat{u}_{el}(t)$ is the element (local) vector $\hat{u}_{el}(t) = (\hat{u}_{el,1}(t), \dots, \hat{u}_{el,3}(t))^\top$, the $\hat{\ddot{u}}_{el}(t)$ and the $\hat{v}_{el}(t)$ are defined similarly. The $F_{face_{el}}$ is the face (local) vector containing contribution of the prescribed tractions on Γ_N . For chosen indices v and w we can write $M_{el,(v,w)}$, $K_{el,(v,w)}$, $F_{el,v}$ and $F_{face_{el,\Gamma_N}}$ as follows

$$M_{el,(v,w)} = \int_{\Omega_{(el)}} \rho \bar{N}_{el,v}^\top(x) \bar{N}_{el,w}(x) dx, \quad (2.14)$$

$$K_{el,(v,w)} = \int_{\Omega_{(el)}} (\varepsilon(\bar{N}_{el,v}(x)))^\top : C : \varepsilon(\bar{N}_{el,w}(x)) dx, \quad (2.15)$$

$$F_{el,v} = \int_{\Omega_{(el)}} f^h(x) \bar{N}_{el,v}(x) dx, \quad (2.16)$$

$$F_{face_{el,\Gamma_N}} = \int_{face_{el,\Gamma_N}} g^h(x) \bar{N}_{el,v}(x) ds. \quad (2.17)$$

As we mention at the beginning of the section 2.2, in 2D the elements can be also quadrilaterals as well as higher order triangles. In these cases the construction of shape functions becomes more complicated and with it also the complexity of the integration process that appear in the M_{el} , K_{el} , F_{el} and $F_{face_{el,\Gamma_N}}$ becomes higher.

Therefore, we will continue with the concept of the reference element $\tilde{\Omega}_{(el)}$ presented in [5], see Figure 2.5. We define mapping $\mathcal{X}_{el} : \tilde{\Omega}_{(el)} \rightarrow \Omega_{(el)}$, so that $x = \mathcal{X}_{el}(\xi)$, $\xi \in \tilde{\Omega}_{(el)}$. For the mesh that consists only from one type of finite elements the subscript (el) can be omitted, i.e., $\tilde{\Omega}$. We consider the shape functions $\phi_{el,v}$ on the $\tilde{\Omega}_{(el)}$ satisfy $\phi_{el,v} = N_{el,v} \circ \mathcal{X}_{el}$. For triangle element (illustrated in Figure 2.5) we have

$$\phi_{el,1}(\xi) = 1 - \xi_1 - \xi_2,$$

$$\phi_{el,2}(\xi) = \xi_1,$$

$$\phi_{el,3}(\xi) = \xi_2.$$

The Jacobian of the mapping $\mathcal{X}_{el}(\xi)$ is denoted by $J_{el}(\xi)$. Last we have to define following rule

$$\varepsilon = (J_{el}(\xi))^{-1} \varepsilon_\xi =: \tilde{\varepsilon}_\xi.$$

Using the concept of the reference element we can rewrite (2.14), (2.15) and (2.16) into

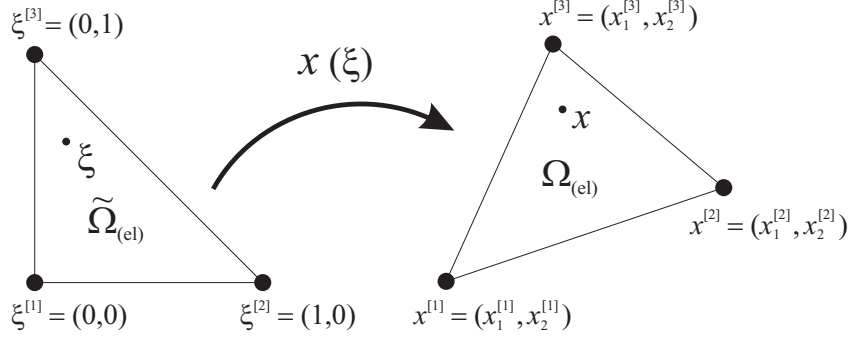


Figure 2.5: Reference triangle element.

the form

$$M_{el,(v,w)} = \int_{\tilde{\Omega}_{(el)}} \rho \bar{\phi}_{el,v}(\xi) \bar{\phi}_{el,w}(\xi) |J_{el}(\xi)| d\xi, \quad (2.18)$$

$$K_{el,(v,w)} = \int_{\tilde{\Omega}_{(el)}} \tilde{\varepsilon}_{\xi}(\bar{\phi}_{el,v}(\xi)) : C : \tilde{\varepsilon}_{\xi}(\bar{\phi}_{el,w}(\xi)) |J_{el}(\xi)| d\xi, \quad (2.19)$$

$$F_{el,v} = \int_{\tilde{\Omega}_{(el)}} f^h(\mathcal{X}_{el}(\xi)) \bar{\phi}_{el,v}(\xi) |J_{el}(\xi)| d\xi, \quad (2.20)$$

where $\bar{\phi}_{el,v}(\xi) := \phi_{el,v}(\xi)I_d$ and $\bar{\phi}_{el,w}(\xi) := \phi_{el,w}(\xi)I_d$. Note that for the (2.17) is necessary to use other reference element than triangle, namely the line, because there we integrate over the face of the element lying on the Γ_N . Let us denote $\phi_{el,v}^{face}(\xi)$ the shape function over this reference element and $\bar{\phi}_{el,v}^{face}(\xi) := \phi_{el,v}^{face}(\xi)I_d$.

As an approximation of the integrals we use the Gaussian quadrature rule [6], so the integrals over the reference element are approximated by the weighted sum over the quadratic (Gauss) points qp with the weights w_{qp} . We also need introduced the notation qp_{face} for the Gauss points of the face reference element. We denote n_{el}^{qp} , $n_{face_{el},\Gamma_N}^{qp_{face}}$ the number of quadratic points on the element and the number of quadratic points on the face of the element lying on the Γ_N , respectively. For each point qp , qp_{face} is defined the corresponding weight w_{qp} , $w_{qp_{face}}$, respectively. The points and weights for different elements of different order are table values [7], some examples of the points of the triangular element are illustrated in Figure 2.6. The matrices (2.18), (2.19) and vector (2.20) can be written as

$$M_{el,(v,w)} \approx \sum_{qp=1}^{n_{el}^{qp}} \rho \bar{\phi}_{el,v}(\xi_{qp}) \bar{\phi}_{el,w}(\xi_{qp}) |J_{el}(\xi_{qp})| w_{qp}, \quad (2.21)$$

$$K_{el,(v,w)} \approx \sum_{qp=1}^{n_{el}^{qp}} \tilde{\varepsilon}_{\xi,ij}(\bar{\phi}_{el,v}(\xi_{qp})) C_{ijkl} \tilde{\varepsilon}_{\xi,kl}(\bar{\phi}_{el,w}(\xi_{qp})) |J_{el}(\xi_{qp})| w_{qp}, \quad (2.22)$$

$$F_{el,v} \approx \sum_{qp=1}^{n_{el}^{qp}} f^h(\mathcal{X}_{el}(\xi_{qp})) \bar{\phi}_{el,v}(\xi_{qp}) |J_{el}(\xi_{qp})| w_{qp}, \quad (2.23)$$

$$F_{face_{el},\Gamma_N} \approx \sum_{qp_{face}=1}^{n_{face_{el},\Gamma_N}^{qp_{face}}} g^h(\mathcal{X}_{el}(\xi_{qp_{face}})) \bar{\phi}_{el,v}^{face}(\xi_{qp_{face}}) |J_{el}(\xi_{qp_{face}})| w_{qp_{face}}. \quad (2.24)$$

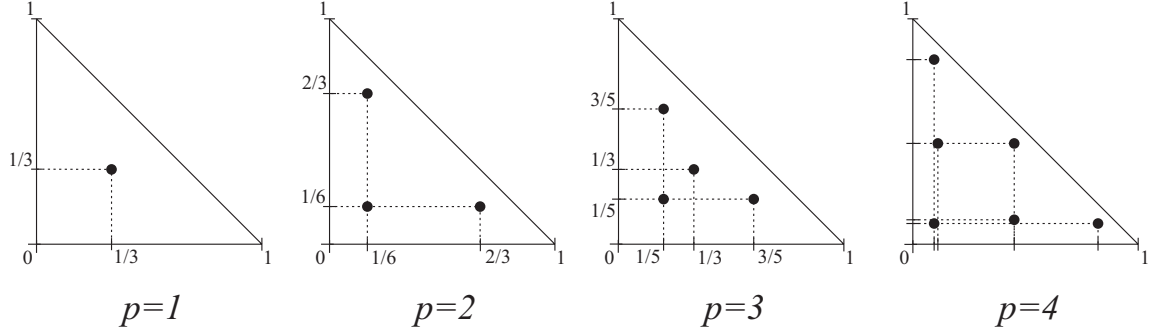


Figure 2.6: Gauss points qp of the triangle of different polynomial degrees p .

The global mass and stiffness matrices M and K , respectively, are assembled by summing the element (local) matrices to appropriate positions in global matrices. The global load vector $F(t)$ is assembled similarly by summing the element (local) vector F_{el} and $F_{face_{el}}$ to appropriate positions in global vector.

It remains show how the Dirichlet boundary conditions can be enforced. Now we have following matrix form of the equation system of the weak problem (2.1) after discretization

$$v_g^\top (K u_g + M \ddot{u}_g) = v_g^\top F, \quad \forall v_g \in \mathbb{R}^{2 \cdot n^{nod}} : i \in \mathcal{D} \Rightarrow v_{gk,i} = 0, \quad (2.25)$$

where v_g and $\ddot{u}_g \in \mathbb{R}^{2 \cdot n^{nod}}$ are defined as well as (2.7). The (2.25) can be block re-arranged into

$$\begin{aligned} & \left(v_{gF}^\top \ v_{gD}^\top \right) \left(\begin{pmatrix} K_{FF} & K_{FD} \\ K_{DF} & K_{DD} \end{pmatrix} \begin{pmatrix} u_{gF} \\ u_{gD} \end{pmatrix} + \begin{pmatrix} M_{FF} & M_{FD} \\ M_{DF} & M_{DD} \end{pmatrix} \begin{pmatrix} \ddot{u}_{gF} \\ \ddot{u}_{gD} \end{pmatrix} \right) = \\ & = \left(v_{gF}^\top \ v_{gD}^\top \right) \begin{pmatrix} F_F \\ F_D \end{pmatrix}, \quad \forall v_{gF} \in \mathbb{R}^{2 \cdot n^{V^{nod}}}, v_{gD} = o, \end{aligned} \quad (2.26)$$

where the subscript D , F denotes values corresponding to the nodes lying on the Dirichlet boundary and values corresponding to the nodes not lying on the Dirichlet boundary, respectively. We consider the homogeneous Dirichlet boundary condition, it means that $u_{gD} = \ddot{u}_{gD} = o$. There is a number of methods to enforce this Dirichlet boundary condition.

We make a modification of matrices K and M to satisfy this condition such that the (2.26) can be written as

$$v_g^\top \left(\underbrace{\begin{pmatrix} K_{FF} & O \\ O & \alpha I \end{pmatrix}}_{=:K} \begin{pmatrix} u_{gF} \\ o \end{pmatrix} + \underbrace{\begin{pmatrix} M_{FF} & O \\ O & \alpha I \end{pmatrix}}_{=:M} \begin{pmatrix} \ddot{u}_{gF} \\ o \end{pmatrix} \right) = v_g^\top \underbrace{\begin{pmatrix} F_F \\ o \end{pmatrix}}_{=:F}, \quad \forall v_g \in \mathbb{R}^{2 \cdot n^{nod}},$$

where $\alpha \in \mathbb{R}$. Note that the notation of new modified matrices and vector remain the same as the original ones.

The space discretized form of the problem now arrive at

$$\begin{cases} \text{Find } u_g(t) : \mathbb{T} \rightarrow \mathbb{R}^{2 \cdot n^{nod}}, \quad \forall t \in \mathbb{T} \text{ such that} \\ M\ddot{u}_g(t) + Ku_g(t) = F(t), \quad \forall t \in \mathbb{T}, \\ u_g(0) = u_{g0}, \quad \dot{u}_g(0) = \dot{u}_{g0}. \end{cases} \quad (2.27)$$

2.4 Discretization in time

Now we make a discretization of time. Let us split the time interval \mathbb{T} into n_t non-overlapping subintervals such that

$$\mathbb{T} = \bigcup_{k=0}^{n_t-1} \langle t_k, t_{k+1} \rangle,$$

where $t_0 = 0$, $t_k < t_{k+1}$, $\Delta t := t_{k+1} - t_k$ and $t_{n_t} = n_t \Delta t$. We denote $u^k := u_g(t_k)$ and $f^k := F(t_k)$ for the simplicity of notation. We will find the solution in each time step t_k , so the space and time discretized form of the problem is:

$$\begin{cases} \forall k \in \{0, 1, \dots, n_t\} \text{ find } u^{k+1} \in \mathbb{R}^{d \cdot n^{nod}} \text{ such that} \\ M\ddot{u}^{k+1} + Ku^{k+1} = f^{k+1}, \\ u^0 = u_{g0}, \quad \dot{u}^0 = \dot{u}_{g0}. \end{cases} \quad (2.28)$$

We will solve this problem by Newmark integration method, we will follow [8].

2.4.1 Newmark integration method

This method proposed in 1959 by Nathan M. Newmark is one of the most popular time integration method for solving problems in structural dynamics. For the computation of unknowns at time t_{k+1} is only necessary to know values in the previous time t_k . Let us introduce a new notation for approximations to the velocity and acceleration vectors at time t_k :

$$\begin{aligned} v^k &:= \dot{u}^k, \\ a^k &:= \ddot{u}^k. \end{aligned}$$

Now we can compute u^{k+1} and v^{k+1} by

$$u^{k+1} = u^k + \Delta t v^k + \Delta t^2 \left[\left(\frac{1}{2} - \beta \right) a^k + \beta a^{k+1} \right], \quad 0 \leq 2\beta \leq 1, \quad (2.29)$$

$$v^{k+1} = v^k + \Delta t \left[(1 - \gamma) a^k + \gamma a^{k+1} \right], \quad 0 \leq \gamma \leq 1, \quad (2.30)$$

where β and γ are real parameters that affects the integration accuracy and the stability of the method. The scheme is unconditionally stable for $2\beta \geq \gamma \geq 1/2$. Setting $2\beta = \gamma = 1/2$ yields the constant average acceleration method (also called trapezoidal rule, see Figure 2.7).

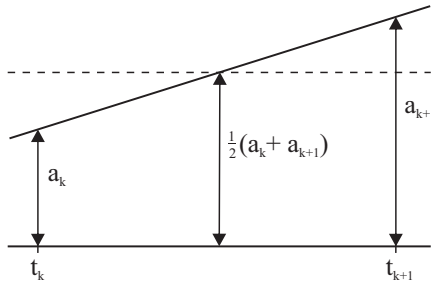


Figure 2.7: The constant average acceleration scheme [9].

In addition to (2.29) and (2.30) we also consider the equilibrium equation at time t_{k+1} with the notation v^{k+1} and a^{k+1} :

$$M a^{k+1} + K u^{k+1} = f^{k+1}. \quad (2.31)$$

We express the acceleration vector a^{k+1} from (2.29) as follows:

$$a^{k+1} = \frac{1}{\Delta t^2 \beta} (u^{k+1} - u^k) - \frac{1}{\Delta t \beta} v^k - \frac{\frac{1}{2} - \beta}{\beta} a^k. \quad (2.32)$$

Substituting (2.32) into (2.30) we obtain

$$v^{k+1} = \frac{\gamma}{\Delta t \beta} (u^{k+1} - u^k) - \frac{\gamma - \beta}{\beta} v^k - \frac{\frac{\gamma}{2} - \beta}{\beta} \Delta t a^k$$

and substituting (2.32) into the equilibrium equation (2.31) we arrive at

$$\underbrace{\left(K + \frac{1}{\Delta t^2 \beta} M \right)}_{=: \hat{K}} u^{k+1} = f^{k+1} + \underbrace{\left(\frac{1}{\Delta t^2 \beta} M u^k + \frac{1}{\Delta t \beta} M v^k + \frac{\frac{1}{2} - \beta}{\beta} M a^k \right)}_{=: \hat{f}^{k+1}}, \quad (2.33)$$

where \hat{K} is so called effective stiffness matrix and \hat{f}^{k+1} is the effective load vector. For the choice of parameters $\beta = 1/4$ and $\gamma = 1/2$ we have a^{k+1} and v^{k+1} of the form

$$a^{k+1} = \frac{4}{\Delta t^2} (u^{k+1} - u^k) - \frac{4}{\Delta t} v^k - a^k, \quad v^{k+1} = \frac{2}{\Delta t} (u^{k+1} - u^k) - v^k$$

and so the equilibrium equation of the form

$$\underbrace{\left(K + \frac{4}{\Delta t^2} M\right)}_{=: \widehat{K}} u^{k+1} = \underbrace{f^{k+1} + \frac{4}{\Delta t^2} M u^k + \frac{4}{\Delta t} M v^k + M a^k}_{=: \widehat{f}^{k+1}}.$$

2.5 Static elasticity problem

In previous section we dealt with the formulation of the dynamic problem. For lack of the time caused by complicated debugging of the code we decided to switch to the static elasticity problem. The reason was also that there are more experiences with static benchmarks solved by using FLLOP than with dynamic benchmarks. Contrariwise the assembly process for the mass matrix is prepared in the code and the implementation of dynamic problems will be not difficult from the side of `libMesh`. In this subsection and in further text we will deal with static elasticity problems, to the dynamic problem we will return eventually in future work.

We can derive the static elasticity problem from the dynamic elasticity problem (2.1). In the static problem all terms are independent on time t so the term m will vanish ($\ddot{u} = 0$). The weak formulation of the static elasticity problem can be written in the form

$$\begin{cases} \text{Find } u \in \mathcal{U} \text{ such that} \\ a(u, v) = f(v), \quad \forall v \in \mathcal{V}, \end{cases} \quad (2.34)$$

where $a(u, v)$, $f(v)$ are the same as (2.3), (2.4), respectively, and

$$\mathcal{U} := \{u \in H^1(\Omega)^2 : \text{Tr}(u) = u_D \text{ on } \Gamma_D\}.$$

We can derived the discretized form of the (2.34) in similar way as the space discretized form of dynamic problems, see Section 2.2 and 2.3. So the discretized form of the static problem can be expressed as

$$\begin{cases} \text{Find } u_g \in \mathbb{R}^{2 \cdot n^{nod}} \text{ such that} \\ K u_g = F, \end{cases} \quad (2.35)$$

where u_g , F and K are the same as in (2.27), only u_g and F are not dependent on time variable t in this case.

3 FETI methods

The FETI (Finite Element Tearing and Interconnecting) methods are nonoverlapping domain decomposition methods. These methods are based on a principle of „divide and conquer“. It means the original large domain, on which the given problem is solved, is partitioned into nonoverlapping smaller subdomains on which local Neumann problems are introduced. Thanks to that the original problem can be solved in parallel. In this section we show two FETI methods - the Total-FETI method and the FETI-DP method. Let us point out that we will describe these methods only for the static problem.

3.1 TFETI (FETI)

The TFETI (Total Finite Element Tearing and Interconnecting) method described by Dostál, Horák and Kučera in [10] is a variant of the FETI method introduced by Farhat and Roux [11] in the early nineties of the last century. In both cases the domain is decomposed into nonoverlapping subdomains and the continuity of displacement on the intersubdomain interface is enforced via Lagrange multipliers. The difference is in approach how to apply the Dirichlet boundary conditions. In the FETI method the Dirichlet boundary conditions are included in the stiffness matrix and the load vector and in the TFETI are enforced by Lagrange multipliers. It causes that all subdomain stiffness matrices are singular and they have a-priori known kernels. In the FETI method we have some positive definite local stiffness matrices and so some subdomain stiffness matrices will have not the same kernels as others. For more information about TFETI, see also [12].

3.1.1 Domain decomposition

Let us decompose domain Ω into N non-overlapping subdomains Ω_i (see Figure 3.1):

$$\overline{\Omega} = \bigcup_{i=1}^N \overline{\Omega}_i, \quad \Omega_i \cap \Omega_j = \emptyset \quad \text{if } i \neq j.$$

We denote the interface between the subdomains by Γ :

$$\Gamma = \bigcup_{i=1}^N \Gamma_i = \bigcup_{i \neq j} \partial\Omega_i \cap \partial\Omega_j,$$

where $\partial\Omega_i, \partial\Omega_j$ are the boundaries of Ω_i, Ω_j . The Dirichlet boundary is denoted by $\partial\Omega_D$ and the Neumann boundary by $\partial\Omega_N$. We introduce new „gluing“ conditions on the interface Γ and on the Dirichlet boundaries to enforce continuity of the displacements.

We consider the globally assembled discretized problem

$$K_g u_g = f_g. \tag{3.1}$$

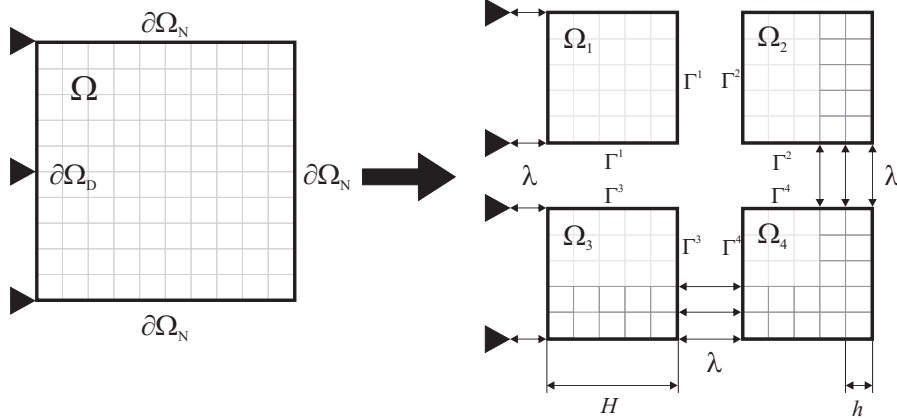


Figure 3.1: Domain decomposition by TFETI [13].

Now we introduce a restriction operators for a transformation from global problem (3.1) to the local problem on the subdomains Ω_i and a prolongation operators for a reverse transformation. Let n , n_{loc_i} denote the total number of degrees of freedom for all nodes and the number of degrees of freedom for nodes in the Ω_i , respectively. The restriction operator Re_i for the subdomain Ω_i is sparse matrix $n_{loc_i} \times n$ with zero entries except 1's at appropriate positions. The local subdomain solution vector u_i and the local subdomain stiffness matrix K_i can be written as:

$$u_i = Re_i u_g, \quad K_i = Re_i K_g Re_i^\top.$$

The prolongation operator Pr_i for the subdomain Ω_i is sparse matrix $n \times n_{loc_i}$ with zero entries except 1's at appropriate positions. The vector \hat{u}_g can be assemble as

$$\hat{u}_g = \sum_{i=1}^N Pr_i u_i.$$

We obtain the global vector u_g by the averaging of values of \hat{u}_g at the positions corresponding to the nodes on the interface Γ .

3.1.2 Primal problem

After the decomposition of discretized Ω we obtain the primal formulation of our problem:

$$Ku = f \quad \text{subject to } Bu = c \quad (3.2)$$

or its equivalent energy formulation

$$\min_u \frac{1}{2} u^\top Ku - f^\top u \quad \text{subject to } Bu = c. \quad (3.3)$$

The matrix $K = \text{diag}(K_1, \dots, K_N)$ is a symmetric positive semidefinite block-diagonal singular stiffness matrix of order n , $f = (f_1, \dots, f_N)^\top$ is a load vector of order n , $u = (u_1, \dots, u_N)^\top$ is a solution vector of order n . The condition $Bu = c$ presents a continuity constraint, so that $B = (B_1, \dots, B_N)$ denotes a full rank sparse constraint matrix $m \times n$ and c denotes a constraint vector of order m . The structure of these matrices and vectors is illustrated in Figure 3.2.

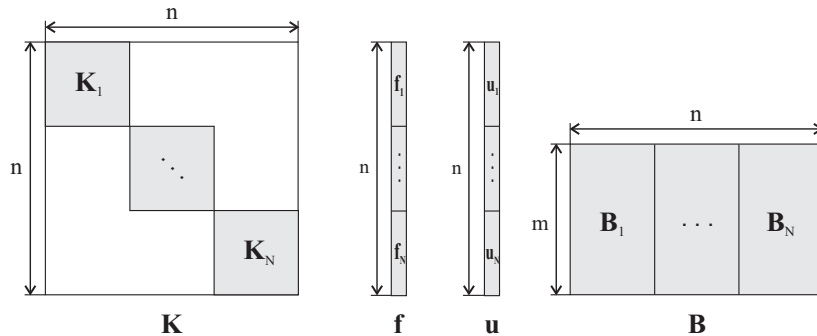


Figure 3.2: The block structure of matrices and vectors of primal formulation [13], [12].

The diagonal blocks of the matrix K are sparse matrices with known kernels, so we can effectively regularize all of them (see more [14], [15]). The regularized matrices can then be decompose using a standard Cholesky type factorization for nonsingular matrices.

The matrix B , also called the jump operator, and the vector c enforce the continuity of the displacements across the interface Γ and the prescribed displacements on the Dirichlet boundary $\partial\Omega_D$. Note that the jump condition across the interface can be written as $u_i - u_j = o$ on $\Omega_i \cap \Omega_j$. The matrix B has zero entries except 1 and -1 at appropriate positions and the vector c has zero entries at all positions except positions where the Dirichlet boundary conditions are given. Let us note that in the FETI method the vector c is obviously a zero vector.

We assemble the matrix B directly in the form with the orthonormal rows. There is a problem with the rows of B corresponding to the nodes shared by more than two subdomains. These rows are linearly dependent, so we have to remove one of them and then we orthonormalize them.

3.1.3 Dual problem

The formulation of the problem (3.3) is not suitable for a numerical solution, because the K is ill-conditioned, singular and very large matrix. These complications may be reduced by applying the theory of duality of convex programming [16].

Let us introduce Lagrange multipliers λ enforcing all the constraints and the Dirichlet boundary conditions, then the Lagrangian associated with (3.3) is

$$L(u, \lambda) = \frac{1}{2}u^\top K u - f^\top u + \lambda^\top (B u - c). \quad (3.4)$$

Derivating of the 3.4 we obtain so called Karush-Kuhn-Tucker (KKT) system or KKT conditions [16]:

$$\begin{cases} \nabla_u L(u, \lambda) = K u - f + B^\top \lambda = 0, \\ \nabla_\lambda L(u, \lambda) = B u - c = 0. \end{cases}$$

This is the dual formulation of our problem, which can be written as

$$\begin{cases} K u + B^\top \lambda = f, \\ B u = c \end{cases}$$

and in the matrix form as

$$\begin{pmatrix} K & B^\top \\ B & O \end{pmatrix} \begin{pmatrix} u \\ \lambda \end{pmatrix} = \begin{pmatrix} f \\ c \end{pmatrix}. \quad (3.5)$$

The problem (3.5) has unique solution $(\hat{u}, \hat{\lambda}) \in \mathbb{R}^n \times \mathbb{R}^m$ which is guaranteed by the following necessary and sufficient conditions [17]:

$$Ker B^\top = o, \quad (3.6)$$

$$Ker K \cap Ker B = o. \quad (3.7)$$

The condition (3.6) mean that matrix B^\top has full column-rank or that matrix B has full row-rank. Note that the first equation in (3.5) is satisfied if

$$f - B^\top \lambda \in Im K$$

and because of $Ker K^\top = Ker K$ we get

$$(f - B^\top \lambda) \perp Ker K. \quad (3.8)$$

Let us gather a basis of $Ker K$ in a block-diagonal matrix $R \in \mathbb{R}^{n \times l}$ (for $l = n - rank(K)$) defined as follows

$$R = \begin{pmatrix} R_1 & & O \\ & \ddots & \\ O & & R_N \end{pmatrix}, \quad (3.9)$$

where the columns of the matrices R_1, \dots, R_N are created by the basis of the $Ker K_i$, i.e., the subdomain rigid body modes. In 2D each matrix $R_i \in \mathbb{R}^{n_{loc_i} \times 3}$ is composed from the following blocks

$$\begin{pmatrix} -x_2^{(j)} & 1 & 0 \\ x_1^{(j)} & 0 & 1 \end{pmatrix} \quad (3.10)$$

associated with the nodes $x^{(j)} \in \Omega_i$. If a local subdomain problem is positive definite, which happens in the FETI method, the corresponding matrix R_i will be empty. Now we can defined $Ker K$ as

$$Ker K = \{R\alpha | \alpha \in \mathbb{R}^l\} \quad (3.11)$$

and from the combination of (3.8) and (3.11) we can write

$$R^\top(f - B^\top \lambda) = 0. \quad (3.12)$$

Let us denote by K^+ a generalized inverse of K satisfying $KK^+K = K$ and note that definition of $Ker K$ (3.11) implies $KR\alpha = 0$. The first equation of (3.5) can be written as

$$Ku + B^\top \lambda = f + KR\alpha$$

and multiplying this equation by K^+ we arrive at

$$u = K^+(f - B^\top \lambda) + R\alpha. \quad (3.13)$$

Reducing of the matrix in (3.5) to row echelon form by applying basic row operation $r_2 - (BK^+)r_1 \rightarrow r_2$ give us

$$\begin{pmatrix} K & B^\top \\ O & -BK^+B^\top \end{pmatrix} \begin{pmatrix} u \\ \lambda \end{pmatrix} = \begin{pmatrix} f + \overbrace{KR\alpha}^{=0} \\ c - BK^+f - BR\alpha \end{pmatrix}. \quad (3.14)$$

Summarizing the second equation in (3.14) and (3.12) we obtain a system

$$\begin{cases} BK^+B^\top \lambda - BR\alpha = BK^+f - c, \\ -R^\top B^\top \lambda = -R^\top f. \end{cases} \quad (3.15)$$

Now we introduce new notation

$$F := BK^+B^\top, \quad G := -R^\top B^\top$$

$$d := BK^+f - c, \quad e := -R^\top f$$

and re-write (3.15) into the matrix form

$$\begin{pmatrix} F & G^\top \\ G & O \end{pmatrix} \begin{pmatrix} \lambda \\ \alpha \end{pmatrix} = \begin{pmatrix} d \\ e \end{pmatrix}. \quad (3.16)$$

The problem to find the solution $(\hat{u}, \hat{\lambda}) \in \mathbb{R}^n \times \mathbb{R}^m$ to the system (3.5) is transformed to the problem to find the pair $(\hat{\lambda}, \hat{\alpha}) \in \mathbb{R}^m \times \mathbb{R}^l$ satisfying (3.16). Then the solution \hat{u} is obtaining from (3.13). Let us note that (3.16) has formally the same structure as that of (3.5), however, its size $m \times l$ is considerably smaller. Moreover, the first diagonal block $F = BK^+B^\top$ is much better conditioned than K .

Now we need solve the first equation in (3.16), but there are two unknown variables. We eliminate the variable α by introducing a projection to the subspace orthogonal to the space spanned by $G^\top \alpha$. This subspace is so-called natural coarse space and we will denote it by $Ker G$. Then the projector onto the $Ker G$ is given by

$$P = P^\top = I - G^\top (GG^\top)^{-1} G.$$

Applying P on the first equation in (3.16) gives us:

$$PF\lambda + PG^\top \alpha = Pd,$$

where $PG^\top \alpha = G^\top \alpha - G^\top (GG^\top)^{-1} GG^\top \alpha = 0$. So we obtain

$$PF\lambda = Pd. \quad (3.17)$$

In order to arrange (3.17) and $G\lambda = e$ (i.e., the second equation of (3.5)) as one equation on the vector space $Ker G$ we split the variable λ into $\lambda_{Im G^\top} \in Im G^\top$ and $\lambda_{Ker G} \in Ker G$ as

$$\lambda = \lambda_{Im G^\top} + \lambda_{Ker G}. \quad (3.18)$$

Now we substitute (3.18) into $G\lambda = e$:

$$\begin{aligned} G(\lambda_{Im G^\top} + \lambda_{Ker G}) &= e \\ G\lambda_{Im G^\top} + \underbrace{G\lambda_{Ker G}}_{=0} &= e, \end{aligned}$$

where $\lambda_{Im G^\top}$ is one particular solution solving $G\lambda_{Im G^\top} = e$ and $\lambda_{Ker G}$ represents all homogenous solutions solving $G\lambda_{Ker G} = 0$. Then $\lambda_{Im G^\top}$ can be identified easily as

$$\lambda_{Im G^\top} = G^\top (GG^\top)^{-1} e. \quad (3.19)$$

One part of λ is known and now we show how to get $\lambda_{Ker G}$. Substituing (3.18) into (3.17) we obtain

$$\begin{aligned} PF(\lambda_{Im G^\top} + \lambda_{Ker G}) &= Pd \\ PF\lambda_{Im G^\top} + PF\lambda_{Ker G} &= Pd \\ PF\lambda_{Ker G} &= P(d - F\lambda_{Im G^\top}). \end{aligned} \quad (3.20)$$

The equation (3.20) is solved with a conjugate gradient method on the subspace $Ker G$. If we know λ , it remains to compute the second unknown variable α . We obtain α by solving the first equation in (3.16) as follows

$$\begin{aligned} F\lambda + G^\top \alpha &= d / \cdot (GG^\top)^{-1} G \\ (GG^\top)^{-1} GF\lambda + \alpha &= (GG^\top)^{-1} Gd \\ \alpha &= (GG^\top)^{-1} G(d - F\lambda). \end{aligned} \quad (3.21)$$

3.2 FETI-DP

We will now talk about the FETI-DP (Dual-primal FETI) method [18]. In this method all local subdomain stiffness matrices are positive definite thanks to the so called primal constraints. So the computation of generalized inverse is not necessary and the matrix R is empty. For more information, see also [19], [20].

3.2.1 Matrix and vector assembling

Let us describe the index sets of nodes used in FETI-DP method, see Figure 3.3. We have the finite element set N_D splitted into the interior set I with nodes inside the subdomains and the interface set Γ with nodes on the subdomains interface. Note that all remaining nodes on the Neumann boundary are considered to be interior nodes of the subdomains. However any nodes on the Dirichlet boundary are not need to be considered, because there are variables prescribed. The Dirichlet boundary conditions are included in the stiffness matrix and in the load vector as well as in the FETI method. Now we decompose Γ into the primal set Π with nodes situated in the corners of the subdomains and the dual set Δ with remaining nodes on the subdomains interface, so we obtain

$$N_D = I \cup \Gamma = I \cup \Pi \cup \Delta,$$

where $I \cap \Gamma = \Pi \cap \Delta = \emptyset$. We also introduce a notation of the nonprimal set $B = I \cup \Delta$.

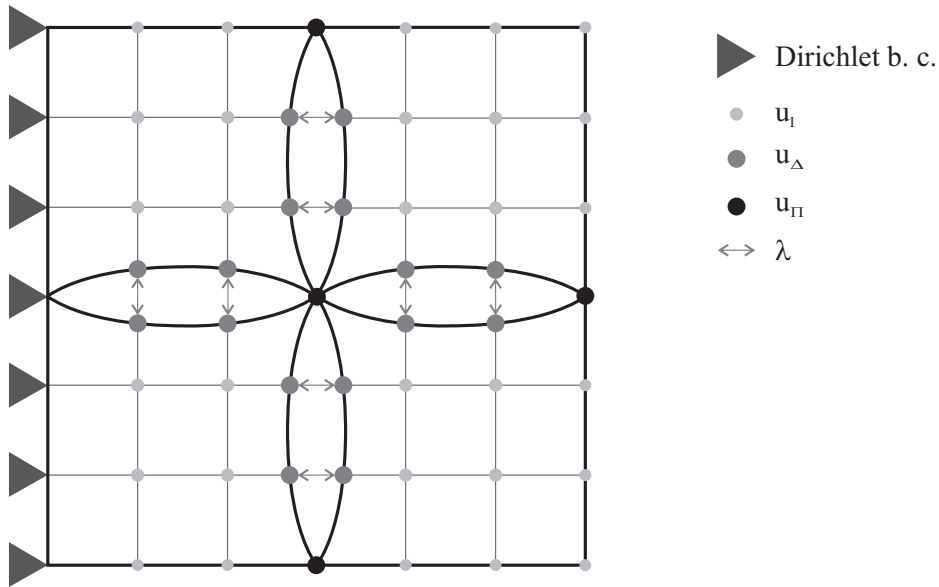


Figure 3.3: Domain decomposition by FETI-DP with all types of variables.

Let us partition the local stiffness matrices, load vectors and solution vectors as follows:

$$K_i = \begin{pmatrix} (K_{BB})_i & (K_{\Pi B})_i^\top \\ (K_{\Pi B})_i & (K_{\Pi\Pi})_i \end{pmatrix}, \quad f_i = \begin{pmatrix} (f_B)_i \\ (f_\Pi)_i \end{pmatrix}, \quad u_i = \begin{pmatrix} (u_B)_i \\ (u_\Pi)_i \end{pmatrix},$$

where components with the subscripts BB and B can be partitioned as follows:

$$(K_{BB})_i = \begin{pmatrix} (K_{II})_i & (K_{\Delta I})_i^\top \\ (K_{\Delta I})_i & (K_{\Delta\Delta})_i \end{pmatrix}, \quad (f_B)_i = \begin{pmatrix} (f_I)_i \\ (f_\Delta)_i \end{pmatrix}, \quad (u_B)_i = \begin{pmatrix} (u_I)_i \\ (u_\Delta)_i \end{pmatrix}.$$

Now we have the global stiffness matrix and load vector of the form

$$K = \begin{pmatrix} K_{BB} & K_{\Pi B}^\top \\ K_{\Pi B} & K_{\Pi\Pi} \end{pmatrix}, \quad f = \begin{pmatrix} f_B \\ f_\Pi \end{pmatrix}, \quad (3.22)$$

where $K_{BB} = \text{diag}((K_{BB})_1, \dots, (K_{BB})_N)$ is a symmetric positive definite block-diagonal matrix, $K_{\Pi\Pi} = \text{diag}((K_{\Pi\Pi})_1, \dots, (K_{\Pi\Pi})_N)$ is a symmetric positive semidefinite block-diagonal matrix and $K_{\Pi B} = \text{diag}((K_{\Pi B})_1, \dots, (K_{\Pi B})_N)$ is a block-diagonal matrix and $f_B = ((f_B)_1, \dots, (f_B)_N)^\top$, $f_\Pi = ((f_\Pi)_1, \dots, (f_\Pi)_N)^\top$ are block vectors.

We introduce assembly operator B_Π^\top for the primal variables which map the local variables u_Π to the global \tilde{u}_Π :

$$\tilde{u}_\Pi = B_\Pi^\top u_\Pi = \sum_{i=1}^N (B_\Pi)_i^\top (u_\Pi)_i. \quad (3.23)$$

The matrix

$$B_\Pi = \begin{pmatrix} (B_\Pi)_1 \\ \vdots \\ (B_\Pi)_N \end{pmatrix}$$

is a sparse boolean block matrix with ones entries at positions of primal nodes (for the example of B_Π , see [21]). Because of $B_\Pi B_\Pi^\top = I$ we can write a mapping from the global variables \tilde{u}_Π to the local u_Π as

$$u_\Pi = B_\Pi \tilde{u}_\Pi.$$

The continuity of the dual variables u_Δ is enforced via Lagrange multipliers λ . We introduce a jump operator $B_B = (O \ B_\Delta)$ such that

$$B_B u_B = B_\Delta u_\Delta = o, \quad (3.24)$$

where B_Δ is matrix with zero entries except 1 and -1 at appropriate positions so that the continuity $(u_\Delta)_i - (u_\Delta)_j = o$ on $\Omega_i \cap \Omega_j$ is satisfied.

We can now define the partial assembled global stiffness matrix as

$$\tilde{K} = \begin{pmatrix} K_{BB} & \tilde{K}_{\Pi B}^\top \\ \tilde{K}_{\Pi B} & \tilde{K}_{\Pi\Pi} \end{pmatrix} = \begin{pmatrix} I_B & O \\ O & B_\Pi^\top \end{pmatrix} \begin{pmatrix} K_{BB} & K_{\Pi B}^\top \\ K_{\Pi B} & K_{\Pi\Pi} \end{pmatrix} \begin{pmatrix} I_B & O \\ O & B_\Pi \end{pmatrix}, \quad (3.25)$$

where $\tilde{K}_{\Pi B}^\top := K_{\Pi B}^\top B_\Pi$, $\tilde{K}_{\Pi B} := B_\Pi^\top K_{\Pi B}$ and $\tilde{K}_{\Pi\Pi} := B_\Pi^\top K_{\Pi\Pi} B_\Pi$. The corresponding global load vector is defined as

$$\tilde{f} = \begin{pmatrix} f_B \\ \tilde{f}_\Pi \end{pmatrix} = \begin{pmatrix} I_B & O \\ O & B_\Pi^\top \end{pmatrix} \begin{pmatrix} f_B \\ f_\Pi \end{pmatrix}. \quad (3.26)$$

and the global solution vector as

$$\tilde{u} = \begin{pmatrix} u_B \\ \tilde{u}_\Pi \end{pmatrix} = \begin{pmatrix} I_B & O \\ O & B_\Pi^\top \end{pmatrix} \begin{pmatrix} u_B \\ u_\Pi \end{pmatrix}. \quad (3.27)$$

The matrix I_B is the identity operator on u_B and the tilde indicates that the continuity in the primal variables is enforced. For a sufficient number of primal variables we obtain a symmetric positive definite matrix \tilde{K} , where $\tilde{K}_{\Pi\Pi}$ is a symmetric positive definite matrix. The structure of \tilde{K} is

$$\tilde{K} = \begin{pmatrix} K_{BB} & \tilde{K}_{\Pi B}^\top \\ \tilde{K}_{\Pi B} & \tilde{K}_{\Pi\Pi} \end{pmatrix} = \left(\begin{array}{cc|c} (K_{BB})_1 & O & (\tilde{K}_{\Pi B})_1^\top \\ & \ddots & \vdots \\ O & (K_{BB})_N & (\tilde{K}_{\Pi B})_N^\top \\ \hline (\tilde{K}_{\Pi B})_1 & \dots & (\tilde{K}_{\Pi B})_N & \tilde{K}_{\Pi\Pi} \end{array} \right). \quad (3.28)$$

3.2.2 Dual problem

The FETI-DP saddle-point problem can be written in matrix form as

$$\begin{pmatrix} K_{BB} & \tilde{K}_{\Pi B}^\top & B_B^\top \\ \tilde{K}_{\Pi B} & \tilde{K}_{\Pi\Pi} & O \\ B_B & O & O \end{pmatrix} \begin{pmatrix} u_B \\ \tilde{u}_\Pi \\ \lambda \end{pmatrix} = \begin{pmatrix} f_B \\ \tilde{f}_\Pi \\ 0 \end{pmatrix}. \quad (3.29)$$

Let us eliminate the primal variables \tilde{u}_Π and non-primal variables u_B from (3.29). First we express the u_B from the first equation as:

$$u_B = K_{BB}^{-1} \left(f_B - B_B^\top \lambda - \tilde{K}_{\Pi B}^\top \tilde{u}_\Pi \right) \quad (3.30)$$

and then the \tilde{u}_Π from the second equation as:

$$\tilde{u}_\Pi = \tilde{S}_{\Pi\Pi}^{-1} \left(\tilde{f}_\Pi - \tilde{K}_{\Pi B} K_{BB}^{-1} (f_B - B_B^\top \lambda) \right), \quad (3.31)$$

where

$$\tilde{S}_{\Pi\Pi} = \tilde{K}_{\Pi\Pi} - \tilde{K}_{\Pi B} K_{BB}^{-1} \tilde{K}_{\Pi B}^\top \quad (3.32)$$

represents the coarse problem of FETI-DP method.

It remains to show how to get λ . If we look at (3.29) as at TFETI problem (3.5), we can make an elimination of the vector \tilde{u} in the same way as in the previous section and then we multiply it by $(B_B \ O)$:

$$\begin{pmatrix} u_B \\ \tilde{u}_\Pi \end{pmatrix} = \begin{pmatrix} K_{BB} & \tilde{K}_{\Pi B}^\top \\ \tilde{K}_{\Pi B} & \tilde{K}_{\Pi\Pi} \end{pmatrix}^{-1} \begin{pmatrix} f_B \\ \tilde{f}_\Pi \end{pmatrix} - \begin{pmatrix} K_{BB} & \tilde{K}_{\Pi B}^\top \\ \tilde{K}_{\Pi B} & \tilde{K}_{\Pi\Pi} \end{pmatrix}^{-1} \begin{pmatrix} B_B^\top \\ O \end{pmatrix} \lambda,$$

$$0 = (B_B \ O) \begin{pmatrix} K_{BB} & \tilde{K}_{\Pi B}^\top \\ \tilde{K}_{\Pi B} & \tilde{K}_{\Pi\Pi} \end{pmatrix}^{-1} \begin{pmatrix} f_B \\ \tilde{f}_\Pi \end{pmatrix} - (B_B \ O) \begin{pmatrix} K_{BB} & \tilde{K}_{\Pi B}^\top \\ \tilde{K}_{\Pi B} & \tilde{K}_{\Pi\Pi} \end{pmatrix}^{-1} \begin{pmatrix} B_B^\top \\ O \end{pmatrix} \lambda.$$

Let us introduce a new notation

$$F := (B_B \ O) \begin{pmatrix} K_{BB} & \tilde{K}_{\Pi B}^\top \\ \tilde{K}_{\Pi B} & \tilde{K}_{\Pi\Pi} \end{pmatrix}^{-1} \begin{pmatrix} B_B^\top \\ O \end{pmatrix}, \quad (3.33)$$

$$d := (B_B \ O) \begin{pmatrix} K_{BB} & \tilde{K}_{\Pi B}^\top \\ \tilde{K}_{\Pi B} & \tilde{K}_{\Pi\Pi} \end{pmatrix}^{-1} \begin{pmatrix} f_B \\ \tilde{f}_\Pi \end{pmatrix}, \quad (3.34)$$

so we obtain the symmetric positive definite linear system:

$$F\lambda = d. \quad (3.35)$$

For computing F and d we need to know the inverse matrix \tilde{K}^{-1} :

$$\begin{pmatrix} K_{BB} & \tilde{K}_{\Pi B}^\top \\ \tilde{K}_{\Pi B} & \tilde{K}_{\Pi\Pi} \end{pmatrix}^{-1} = \begin{pmatrix} K_{BB}^{-1} + K_{BB}^{-1} \tilde{K}_{\Pi B}^\top \tilde{S}_{\Pi\Pi}^{-1} \tilde{K}_{\Pi B} K_{BB}^{-1} & -K_{BB}^{-1} \tilde{K}_{\Pi B}^\top \tilde{S}_{\Pi\Pi}^{-1} \\ -\tilde{S}_{\Pi\Pi}^{-1} \tilde{K}_{\Pi B} K_{BB}^{-1} & \tilde{S}_{\Pi\Pi}^{-1} \end{pmatrix}. \quad (3.36)$$

We know that this inverse matrix exists, because K_{BB} and $\tilde{K}_{\Pi\Pi}$ are symmetric positive definite matrices. Note that the (3.36) and the inverse of (3.32) are never computed explicitly. We only need them in terms of matrix vector multiplications.

Now we multiply the first component $(\tilde{K}^{-1})_{11}$ by B_B and B_B^\top and we obtain the matrix F , the vector d similarly:

$$\begin{aligned} F &= B_B K_{BB}^{-1} B_B^\top + B_B K_{BB}^{-1} \tilde{K}_{\Pi B}^\top \tilde{S}_{\Pi\Pi}^{-1} \tilde{K}_{\Pi B} K_{BB}^{-1} B_B^\top, \\ d &= B_B K_{BB}^{-1} f_B - B_B K_{BB}^{-1} \tilde{K}_{\Pi B}^\top \tilde{S}_{\Pi\Pi}^{-1} (\tilde{f}_\Pi - \tilde{K}_{\Pi B} K_{BB}^{-1} f_B). \end{aligned}$$

The solution of the system (3.35) is computing by the conjugate gradient method.

4 Parallel input data generation by the libMesh library

4.1 LibMesh library

The `libMesh` is an open-source software library in C++ [22] developed originally at The University of Texas at Austin in the CFDLab in March 2002. However the contributions have made elsewhere in the U.S. and abroad. The `libMesh` library is a preprocessing tool for the numerical mesh-based simulation of partial differential equations (PDEs) on serial and parallel platforms, using the finite element method. The library supports discretization of one, two and three dimensional steady and transient problems using various element types. Note that for a parallel computing is used the MPI standard. A major goal of the library is to provide a tool in which the users can focus on the specifics of a given physical problem without considering the complexities of parallel and adaptive computing. To come true this goal the library hides parallel communication from the user, so basic MPI calls are not necessary in most applications. The `libMesh` provides interfaces to high performance existing software, such as PETSc, LAMMPS or SLEPc, whenever possible. A description of the installation process of the library is presented in [23].

4.2 Key data structures

In this section we describe some key data structures used in `libMesh`. We follow the article [24] and the presentation [25].

Mesh

The `Mesh` class is main class in `libMesh`. The `Mesh` describes a discrete form of a body in d -dimensional space, where $d = 1, 2$ or 3 . It includes elements and nodes, which are described in following parts.

The `libMesh` supports reading and writing a few of mesh formats, for example Exodus II or GMSH. Unfortunately we have had some problems with input of a mesh from an external software package. First at all the software packages used by engineers at VŠB don't support mesh formats included in `libMesh`. It has been possible to use GMSH software as a mesh generator. However there has been a problem with correct reading of the input file in `libMesh` and mainly a problem of addition the setting of physic to mesh with following correct reading in `libMesh`. We spent some time over solving these problems, but the right solution would require much more time. So we decided for mesh generation by `libMesh` (for now).

The `Mesh` is partitioned into non-overlapping subdomains such that each subdomain with corresponding elements is assigned to an individual processor. In the library the default partitioning algorithm is METIS. In the `Mesh`, each node lying on border between subdomains is assigned to processor with lower index. The example of partitioned `Mesh` is illustrated in Figure 4.1a), where nodes 4, 5, 10, 11, 12 are assigned to processor 0, nodes

17, 22 to processor 1 and nodes 13, 14 to processor 2. To create of a new mesh containing all the elements which are assigned to a given processor is used a function `create_pid_mesh`. So we have one `pid_mesh` for each processor. Note that the `pid_mesh` is useful just for the FETI methods. An example of `pid_mesh` created from `Mesh` shown in Figure 4.1a) for 4 processors is illustrated in Figure 4.1b).

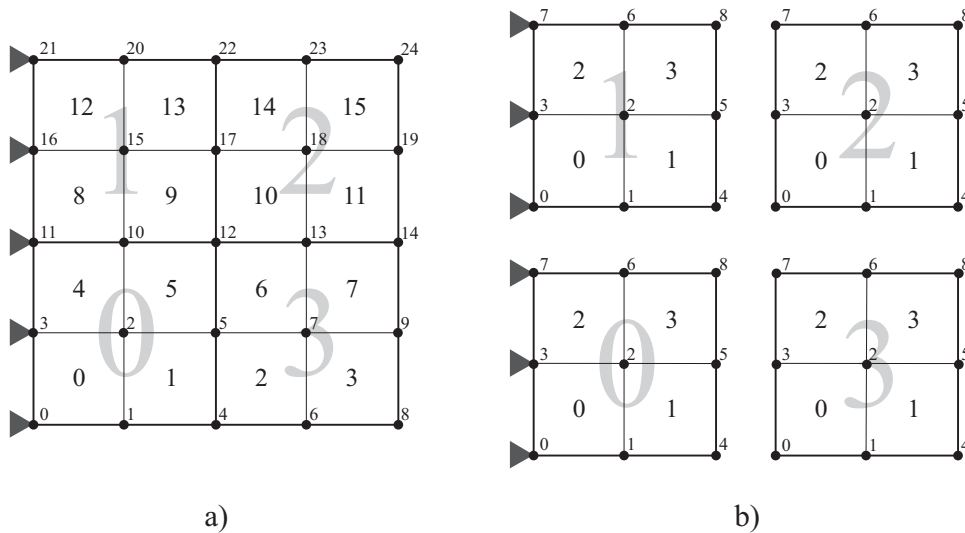


Figure 4.1: The example of mesh and `pid_mesh` for 16 elements and 4 processors.

The access to the elements and nodes from a `Mesh` is possible through iterators. An iterator is a mechanism that enables to traverse a field of objects. We can use iterators to access all elements/nodes in the mesh or only some subset of them, for example, only the local nodes in the mesh.

Elements

The `Elem` base class defines a geometric element. All geometric element types provided in `libMesh` are shown in rightmost column of Figure 4.2. Note that an element for 1D, 2D and 3D is denoted by `Edge`, `Face` and `Cell`, respectively. For finite element analysis the `Elem` class includes

- quadrilaterals (`Quad`) and triangles (`Tri`) for 2D,
- hexahedra (`Hex`), tetrahedra (`Tet`), prisms and pyramids for 3D.

Elements store information as a unique identification number (ID), degree of freedom information and a processor ownership. They also store pointers to their face neighbors and to their nodes. The face neighbors are neighbor elements shared a side, where a side is a `Node` in 1D, an `Edge` in 2D and a `Face` in 3D.

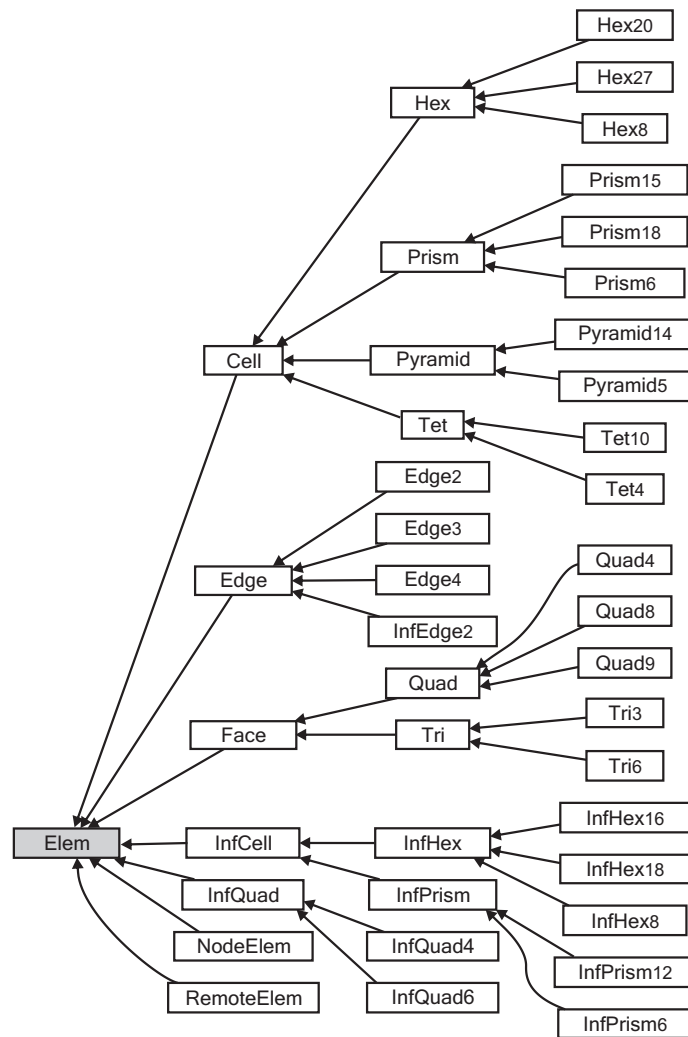


Figure 4.2: The Elem class hierarchy.[26]

Nodes

Nodes are points in arbitrary dimensional space defined by its spatial location. Each object of the `Node` class stores its coordinates and additional information such as a global ID, degree of freedom indices and a processor ownership. To `Nodes` we can access directly via iterators, or indirectly through elements.

Systems

For one `Mesh` object only one `EquationSystems` object, which represents one or more systems of equations defined on the `Mesh`, can be given. The `EquationSystems` object

can include many **Systems**, each corresponds to a PDE system of one or more equations. The library provides at least implementation of explicit, implicit, steady, transient, linear, nonlinear and eigen systems, see Figure 4.3.

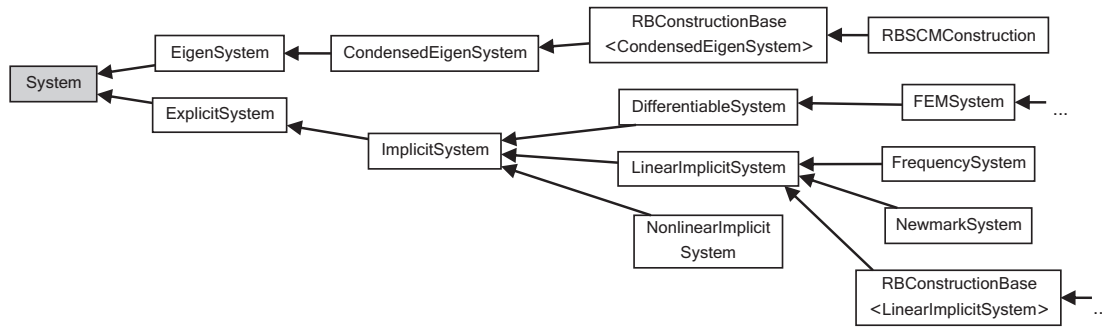


Figure 4.3: The **System** class hierarchy. [27]

Adding a **System** to the **EquationSystems** is performed by function `add_system<T_sys>("name")`, where `T_sys` is **System** type and `name` is an arbitrary name of the **System**. Then we can use the function `get_system("name")` to return the reference to the **System** named `name`.

Each **System** object stores the solution vector of degrees of freedom values and may contain additional information such as a **SparseMatrix** and a **NumericVector**. For example, in our implementation a **SparseMatrix** is the stiffness matrix K and a **NumericVector** is the load vector F , for dynamic elasticity problem a **SparseMatrix** is the effective stiffness matrix and a **NumericVector** is the effective load vector.

4.3 Description of the implementation

In this part we focus on a description of major functions included in our implementation, especially on the assembling of elasticity stiffness matrix K and load vector F . We have to remark that the implementation is prepared only for static elasticity problem without mass matrix M , see Subsection 2.5. However the assembling of matrix M in `libMesh` is not difficult and we are able to do it. The dynamic elasticity problem will be added in future. Still we can follow a description of the assembling of the matrices and vectors in the Subsection 2.3, i.e., we consider the dynamic elasticity problem only in one time step t .

The structure of the implementing flow is illustrated in Figure 4.4. The code can run in parallel as well as in serial without any changes of the implementation. A switching between serial and parallel computing is very easy, for serial, parallel computing we set run command to `./example-dbg`, `mpirun -np X ./example-dbg`, respectively, where X is a number of processors. Let us note that the number of subdomains, how we defined in section 3.1.1, is equal to a number of processors and the implementation is a completely functional for dimension equal 2 (for 1D and 3D will be finished in future).

For solving process the package FLLOP (FETI Light Layer On top of PETSc) [28], originally developed by Václav Hapla and David Horák, is used. The assembling of the matrix B is implemented by Václav Hapla, Lubomír Říha and Alexandros Markopoulos.

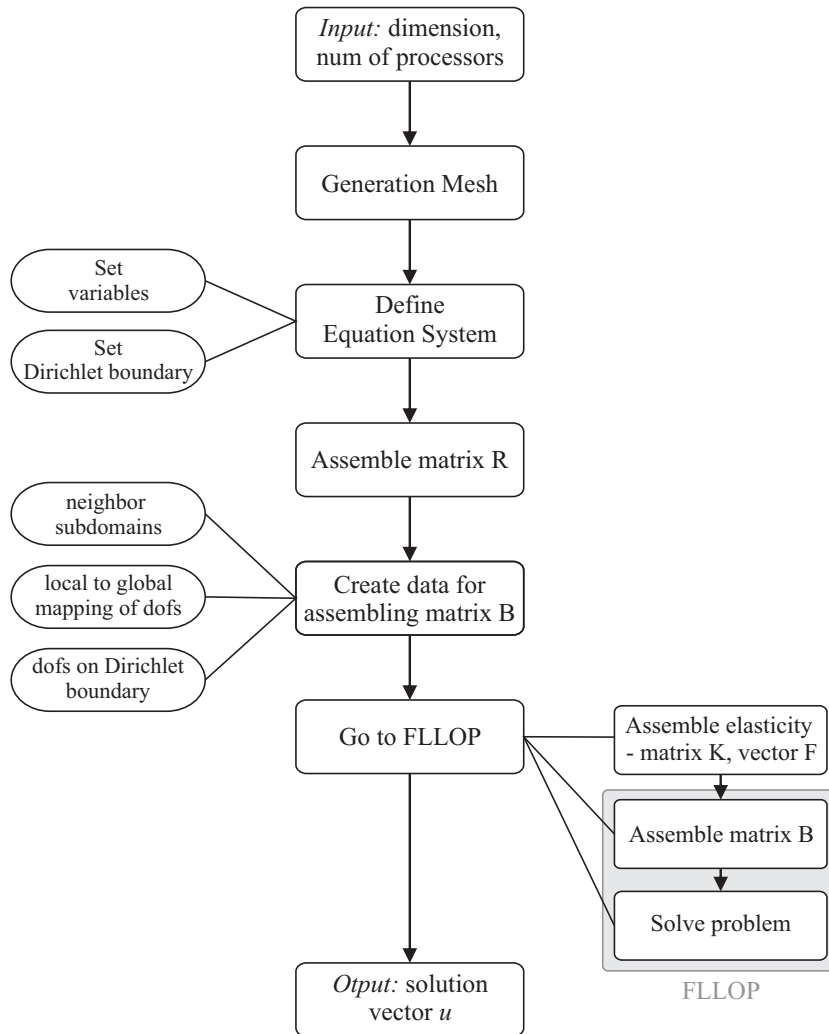


Figure 4.4: Scheme of the implementing flow.

4.3.1 Function `assemble_R`

For assembling the matrix of the rigid body modes R , i.e. (3.9), we implemented the function `assemble_R`. This function builds the submatrix R_i for each processor i and supports 1D, 2D and 3D problem. The matrix is stored as a dense matrix. Let us show the assembling part of code for 2D case that is included inside a for cycle through local

nodes appropriate to the i -th processor, see Part of code 1. Note that the `dim` is equal to 2 and `point(j)` returns j -th coordinate of the current node.

Part of code 1 Assembling R_i in 2D.

```

1: for ( ; nod != end_nod; nod++)
2: {
3:   const Node* node = *nod;
4:   const Point point = *static_cast<const Point*> (node);
5:   switch (dim)
6:   {
7:     ...
8:     case 2:
9:       R(dim*node->id() ,0) = point(1); /rotation about x axis
10:      R(dim*node->id()+1 ,0) = -point(0); /rotation about y axis
11:      R(dim*node->id() ,1) = 1; /displacement in the x-direction
12:      R(dim*node->id()+1 ,2) = 1; /displacement in the y-direction
13:      break;
14:     ...
15:   }
16: }
```

4.3.2 Function `create_data_for_B`

To assemble the jump matrix B defined in section 3.1.2 we need some information and these are created in the function `create_data_for_B`. For each processor the following information is needed:

- vector of neighbor subdomains (processors)
- mapping vector from local degrees of freedom to global ones
- vector of degrees of freedom corresponding to the Dirichlet boundary.

How we can see in scheme 4.4, the assembling of the B itself is executed inside FLLOP package. Let us show an example of the creation of data for B . We consider the mesh illustrated in Figure 4.1. For processors p_i , where $i = 0, \dots, 3$, we obtain the following vectors of neighbor subdomains

$$p_0 : \begin{pmatrix} 3 \\ 1 \end{pmatrix}, \quad p_1 : \begin{pmatrix} 0 \\ 2 \end{pmatrix}, \quad p_2 : \begin{pmatrix} 3 \\ 1 \end{pmatrix}, \quad p_3 : \begin{pmatrix} 0 \\ 2 \end{pmatrix}.$$

For 2D elasticity problem we have two degrees of freedom (for each direction in 2D coordinate system) so two indices of dof's, i.e., $2j$ and $2j + 1$ corresponds to each node j . For four processors we obtain the following mapping vectors from local degrees of freedom (see

pid_mesh in Figure 4.1) to global

$$p_0 : \begin{pmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 20 \\ 21 \\ 22 \\ 23 \\ 24 \\ 25 \end{pmatrix}, \quad p_1 : \begin{pmatrix} 22 \\ 23 \\ 20 \\ 21 \\ 30 \\ 31 \\ 32 \\ 33 \\ 24 \\ 25 \\ 34 \\ 35 \\ 40 \\ 41 \\ 42 \\ 43 \\ 44 \\ 45 \end{pmatrix}, \quad p_2 : \begin{pmatrix} 24 \\ 25 \\ 26 \\ 27 \\ 36 \\ 37 \\ 34 \\ 35 \\ 28 \\ 29 \\ 38 \\ 39 \\ 46 \\ 47 \\ 44 \\ 45 \\ 48 \\ 49 \end{pmatrix}, \quad p_3 : \begin{pmatrix} 8 \\ 9 \\ 12 \\ 13 \\ 14 \\ 15 \\ 10 \\ 11 \\ 16 \\ 17 \\ 18 \\ 19 \\ 26 \\ 27 \\ 24 \\ 25 \\ 28 \\ 29 \end{pmatrix}$$

and the following vectors of degrees of freedom corresponding to the Dirichlet boundary

$$p_0 : \begin{pmatrix} 0 \\ 1 \\ 6 \\ 7 \\ 22 \\ 23 \end{pmatrix}, \quad p_1 : \begin{pmatrix} 22 \\ 23 \\ 32 \\ 33 \\ 42 \\ 43 \end{pmatrix}, \quad p_2 : \begin{pmatrix} \\ \\ \\ \\ \\ \end{pmatrix}, \quad p_3 : \begin{pmatrix} \\ \\ \\ \\ \\ \end{pmatrix},$$

where two empty vectors are appeared.

4.3.3 Function *go_to_fflop*

In this function we call the assembling function at the beginning, namely `assemble_elasticity` (more about in the next subsection). Then we continue only by PETSc and FLLOP functions. At first we have to convert `libMesh` data to the PETSc format and then rearrange/set them by FLLOP commands to the form needed for the assembling matrix B and for solving the QP problem.

The key function for the assembling of jump matrix B is `QPFetiSetUp`. Inside this function we call `QPFetiAssembleGluing` for the assembling of the gluing part of the B and `QPFetiAssembleDirichlet` for the assembling of the part of the B enforcing the Dirichlet boundary conditions. The FETI process preparing the input data for solving process itself

is included in the function `QPFetiPrepare` and the solving of the QP problem is executed by the `QPSSolve`.

4.3.4 Function `assemble_elasticity`

This function implements the parallel assembly process of the stiffness matrix K and the load vector F in 2D [29]. The assembling operates on the (active) elements assigned to each processor, i.e., (active) local elements. First the local element matrix and vector are computed and then these are added into the global matrix and vector. Let us remind the numerical approximation of the $K_{el,(v,w)}$, the $F_{el,v}$ and the F_{face_{el},Γ_N} from the end of the section 2.3.

$$K_{el,(v,w)} \approx \sum_{qp=1}^{n_{el}^{qp}} \tilde{\varepsilon}_{\xi,ij}(\bar{\phi}_{el,v}(\xi_{qp})) C_{ijkl} \tilde{\varepsilon}_{\xi,kl}(\bar{\phi}_{el,w}(\xi_{qp})) |J_{el}(\xi_{qp})| w_{qp}, \quad (4.1)$$

$$F_{el,v} \approx \sum_{qp=1}^{n_{el}^{qp}} f^h(\mathcal{X}_{el}(\xi_{qp})) \bar{\phi}_{el,v}(\xi_{qp}) |J_{el}(\xi_{qp})| w_{qp}, \quad (4.2)$$

$$F_{face_{el},\Gamma_N} \approx \sum_{qp_{face}=1}^{n_{face_{el},\Gamma_N}^{qp_{face}}} g^h(\mathcal{X}_{el}(\xi_{qp_{face}})) \bar{\phi}_{el,v}^{face}(\xi_{qp_{face}}) |J_{el}(\xi_{qp_{face}})| w_{qp_{face}}, \quad (4.3)$$

where $\bar{\phi}_{el,v}(\xi_{qp}) = \phi_{el,v}(\xi_{qp})I_d$ and $\bar{\phi}_{el,v}^{face}(\xi_{qp_{face}}) = \phi_{el,v}^{face}(\xi_{qp_{face}})I_d$, the I_d is the identity matrix of order $d = 2$ in this case. Let us denote n_{el}^v (n_{el}^w) the number of vertices on element (note that $n_{el}^v = n_{el}^w$). The element matrix $K_{el} \in \mathbb{R}^{dn_{el}^v \times dn_{el}^v}$ and the element vector $F_{el} \in \mathbb{R}^{dn_{el}^v \times 1}$ are created by (4.1) and (4.2), respectively. The values of (4.3) are then added to the vector F_{el} at appropriate positions. This new vector will be denoted by $F_{e_{el}} \in \mathbb{R}^{dn_{el}^v \times 1}$ (named $F_{e_{el}}$ as extended F_{el}). Now we can decompose the matrix K_{el} and the vector $F_{e_{el}}$ into the following $d \times d$ ($d \times 1$, respectively) overlapping blocks. For $d = 2$ we have

$$K_{el} = \text{reindexed} \begin{pmatrix} K_{uu,el} & K_{uv,el} \\ K_{vu,el} & K_{vv,el} \end{pmatrix}, \quad F_{e_{el}} = \text{reindexed} \begin{pmatrix} F_{e_{u,el}} \\ F_{e_{v,el}} \end{pmatrix},$$

where u and v mean displacements in direction of the line coordinates, in the section 2 we denote them u_1 and u_2 . Since the structure of the assembling K_{el} and $F_{e_{el}}$ is quite similar we focus now only on the assembling of K_{el} in detail, because this assembling is more complicated. The blocks $K_{uu,el}$, $K_{uv,el}$, $K_{vu,el}$ and $K_{vv,el}$ can be written as

$$K_{uu,el} = \sum_{qp=1}^{n_{el}^{qp}} \sum_{v=1}^{n_{el}^v} \sum_{w=1}^{n_{el}^w} \tilde{\varepsilon}_{\xi,ij} \left(\begin{bmatrix} \phi_{el,v}(\xi_{qp}) \\ 0 \end{bmatrix} \right) C_{ijkl} \tilde{\varepsilon}_{\xi,kl} \left(\begin{bmatrix} \phi_{el,w}(\xi_{qp}) \\ 0 \end{bmatrix} \right) |J_{el}(\xi_{qp})| w_{qp}, \quad (4.4)$$

$$K_{uv,el} = \sum_{qp=1}^{n_{el}^{qp}} \sum_{v=1}^{n_{el}^v} \sum_{w=1}^{n_{el}^w} \tilde{\varepsilon}_{\xi,ij} \left(\begin{bmatrix} \phi_{el,v}(\xi_{qp}) \\ 0 \end{bmatrix} \right) C_{ijkl} \tilde{\varepsilon}_{\xi,kl} \left(\begin{bmatrix} 0 \\ \phi_{el,w}(\xi_{qp}) \end{bmatrix} \right) |J_{el}(\xi_{qp})| w_{qp}, \quad (4.5)$$

$$K_{vu,el} = \sum_{qp=1}^{n_{el}^{qp}} \sum_{v=1}^{n_{el}^v} \sum_{w=1}^{n_{el}^w} \tilde{\varepsilon}_{\xi,ij} \left(\begin{bmatrix} 0 \\ \phi_{el,v}(\xi_{qp}) \end{bmatrix} \right) C_{ijkl} \tilde{\varepsilon}_{\xi,kl} \left(\begin{bmatrix} \phi_{el,w}(\xi_{qp}) \\ 0 \end{bmatrix} \right) |J_{el}(\xi_{qp})| w_{qp}, \quad (4.6)$$

$$K_{vv,el} = \sum_{qp=1}^{n_{el}^{qp}} \sum_{v=1}^{n_{el}^v} \sum_{w=1}^{n_{el}^w} \tilde{\varepsilon}_{\xi,ij} \left(\begin{bmatrix} 0 \\ \phi_{el,v}(\xi_{qp}) \end{bmatrix} \right) C_{ijkl} \tilde{\varepsilon}_{\xi,kl} \left(\begin{bmatrix} 0 \\ \phi_{el,w}(\xi_{qp}) \end{bmatrix} \right) |J_{el}(\xi_{qp})| w_{qp}. \quad (4.7)$$

Note that for different combinations of i, j we have

$$\begin{aligned} \tilde{\varepsilon}_{\xi,00} \left(\begin{bmatrix} \phi_{el,v}(\xi_{qp}) \\ 0 \end{bmatrix} \right) &= \partial_{x_1} \phi_{el,v}, \\ \tilde{\varepsilon}_{\xi,01} \left(\begin{bmatrix} \phi_{el,v}(\xi_{qp}) \\ 0 \end{bmatrix} \right) &= \frac{1}{2} (\partial_{x_1} \phi_{el,v} + \partial_{x_2} 0), \\ \tilde{\varepsilon}_{\xi,10} \left(\begin{bmatrix} \phi_{el,v}(\xi_{qp}) \\ 0 \end{bmatrix} \right) &= \frac{1}{2} (\partial_{x_1} 0 + \partial_{x_2} \phi_{el,v}), \\ \tilde{\varepsilon}_{\xi,11} \left(\begin{bmatrix} \phi_{el,v}(\xi_{qp}) \\ 0 \end{bmatrix} \right) &= \partial_{x_2} 0. \end{aligned}$$

Let us look in detail on all non-zero combinations of

$$\tilde{\varepsilon}_{\xi,ij} \left(\begin{bmatrix} \phi_{el,v}(\xi_{qp}) \\ 0 \end{bmatrix} \right) \tilde{\varepsilon}_{\xi,kl} \left(\begin{bmatrix} \phi_{el,w}(\xi_{qp}) \\ 0 \end{bmatrix} \right) \quad (4.8)$$

from the equation (4.4). For following choices i, j, k, l we obtain

$$\begin{aligned} i, j, k, l = 0 &: \partial_{x_1} \phi_{el,v} \cdot \partial_{x_1} \phi_{el,w} =: A, \\ i, j, k = 0, l = 1 &: \partial_{x_1} \phi_{el,v} \cdot \frac{1}{2} \partial_{x_2} \phi_{el,w} =: B, \\ i, j, l = 0, k = 1 &: \partial_{x_1} \phi_{el,v} \cdot \frac{1}{2} \partial_{x_2} \phi_{el,w} =: C, \\ i, k, l = 0, j = 1 &: \frac{1}{2} \partial_{x_2} \phi_{el,v} \cdot \partial_{x_1} \phi_{el,w} =: D, \\ j, k, l = 0, i = 1 &: \frac{1}{2} \partial_{x_2} \phi_{el,v} \cdot \partial_{x_1} \phi_{el,w} =: E, \\ i, k = 0, j, l = 1 &: \frac{1}{2} \partial_{x_2} \phi_{el,v} \cdot \frac{1}{2} \partial_{x_2} \phi_{el,w} =: F, \\ i, l = 0, j, k = 1 &: \frac{1}{2} \partial_{x_2} \phi_{el,v} \cdot \frac{1}{2} \partial_{x_2} \phi_{el,w} =: G, \end{aligned}$$

$$j, k = 0, i, l = 1 : \frac{1}{2} \partial_{x_2} \phi_{el,v} \cdot \frac{1}{2} \partial_{x_2} \phi_{el,w} =: H,$$

$$j, l = 0, i, k = 1 : \frac{1}{2} \partial_{x_2} \phi_{el,v} \cdot \frac{1}{2} \partial_{x_2} \phi_{el,w} =: I.$$

As we can see some equations are same for different combinations, namely $B = C$, $D = E$ and $F = G = H = I$. So we have

$$A = 1 \cdot (\partial_{x_1} \phi_{el,v} \cdot \partial_{x_1} \phi_{el,w}) = \partial_{x_1} \phi_{el,v} \cdot \partial_{x_1} \phi_{el,w}, \quad (4.9)$$

$$2B = 2C = 2 \cdot \left(\partial_{x_1} \phi_{el,v} \cdot \frac{1}{2} \partial_{x_2} \phi_{el,w} \right) = \partial_{x_1} \phi_{el,v} \cdot \partial_{x_2} \phi_{el,w}, \quad (4.10)$$

$$2D = 2E = 2 \cdot \left(\frac{1}{2} \partial_{x_2} \phi_{el,v} \cdot \partial_{x_1} \phi_{el,w} \right) = \partial_{x_2} \phi_{el,v} \cdot \partial_{x_1} \phi_{el,w}, \quad (4.11)$$

$$4F = 4G = 4H = 4I = 4 \cdot \left(\frac{1}{2} \partial_{x_2} \phi_{el,v} \cdot \frac{1}{2} \partial_{x_2} \phi_{el,w} \right) = \partial_{x_2} \phi_{el,v} \cdot \partial_{x_2} \phi_{el,w}. \quad (4.12)$$

To obtain $K_{uu,el}$ we need all ($2^4 = 16$) combinations of $i, j, k, l \in \{0, 1\}$ to the sum (Einstein summation convention) in (4.4). The only 9 combinations (in the previous text denoted by A – I) of all combinations have non-zero values and thanks to the (4.9)–(4.12) we need only 4 representatives of these 9. The reader can make sure himself that A , B , D and F are the representatives with fixed indices $i = k = 0$ and varying indices j, l .

The terms in the (4.9) up to (4.12) can be written by using a nabla operator. Let us introduce the nabla operator

$$\tilde{\nabla}_\xi := (J_{el}(\xi))^{-1} \nabla_\xi,$$

where $\nabla_\xi = \left(\frac{\partial}{\partial \xi_1}, \dots, \frac{\partial}{\partial \xi_2} \right)$. By using it we can write

$$\partial_{x_1} \phi_{el,v} = \tilde{\nabla}_{\xi,0}(\phi_{el,v}),$$

$$\partial_{x_2} \phi_{el,v} = \tilde{\nabla}_{\xi,1}(\phi_{el,v})$$

and so

$$\tilde{\nabla}_{\xi,0}(\phi_{el,v}) \cdot \tilde{\nabla}_{\xi,0}(\phi_{el,w}),$$

$$\tilde{\nabla}_{\xi,0}(\phi_{el,v}) \cdot \tilde{\nabla}_{\xi,1}(\phi_{el,w}),$$

$$\tilde{\nabla}_{\xi,1}(\phi_{el,v}) \cdot \tilde{\nabla}_{\xi,0}(\phi_{el,w}),$$

$$\tilde{\nabla}_{\xi,1}(\phi_{el,v}) \cdot \tilde{\nabla}_{\xi,1}(\phi_{el,w}).$$

Now we look closer at the Einstein sum in (4.4), where 16 combinations of $i, j, k, l \in \{0, 1\}$ are summed, i.e., we sum 9 non-zero values A – I and additional 7 zero values multiplied by C_{ijkl} with appropriate combinations of i, \dots, l . The values of C_{ijkl} are the same for the combinations (B, C) , (D, E) and (F, G, H, I) thanks to the symmetries in C_{ijkl} . By taking

```

20:
21:     C_j=1, C_l=1;
22:     Kuu(i,j) += JxW[qp]*(eval_elasticity_tensor(C_i,C_j,C_k,C_l) *
23:         dphi[i][qp](C_j)*dphi[j][qp](C_l));
24:     }
25:     ...

```

Let us remind that C_{ijkl} is defined as

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

where

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)},$$

$$\mu = \frac{E}{2(1+\nu)}.$$

The coefficient E is Young's modulus and the ν is Poisson's ratio. In the next section we will choose these coefficients as $E = 1$ and $\nu = 0.3$. The Lamé coefficients λ and μ are denoted by `lambda1` and `lambda2` in `libMesh`.

5 Numerical experiments

For numerical experiments the Anselm supercomputer, built on the campus of VŠB-TU Ostrava, is used. The Anselm cluster [30] consists of 209 computational nodes (of which 180 are regular compute nodes), totaling 3344 compute cores with 15TB RAM and giving over 94 Tflop/s theoretical peak performance. Each node is a powerful x86-64 computer, equipped with 16 cores (two eight-core Intel Sandy Bridge processors), at least 64GB RAM, and 500GB hard drive.

We test the numerical (weak) and parallel (strong) scalability of our data generation implementation. The numerical scalability is the ability of a process to keep the number of iterations independent (invariable) on a growing amount of unknowns. The parallel scalability means that run time of a process is inversely proportional to the number of used processors.

For testing we choose following benchmark. We model a static linear elasticity problem for a homogeneous isotropic rectangular body $(-2, 2) \times (-1, 1)$. The left boundary is fixed and a vertical load $g = -1$ is applied at the right boundary, see Figure 5.1. We choose the Young's modulus as $E = 1$ and the Poisson's ratio as $\nu = 0.3$.

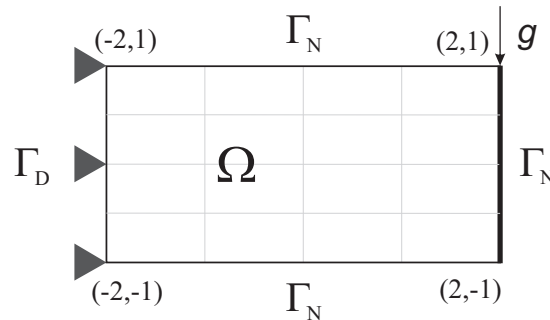


Figure 5.1: Geometry for the given example.

To demonstrate scalabilities we varied the number of subdomains (N). The number of elements in the whole body (n^{el}) is invariable for parallel (strong) scalability and variable for numerical (weak) scalability, where the number of elements of the subdomain is the almost same for all subdomains (for partitioning by METIS is not always exactly same).

The numerical (weak) scalability result is summarized in Table 5.1, 5.2 and 5.3, the parallel (strong) scalability result in Table 5.4 and 5.5. The numbers of iterations for a growing amount of subdomains is shown in Figure 5.2 and the time results are illustrated in Figure 5.3, 5.4 and 5.5. Note that the Num_dof's/pid0 and Num_elem/pid0 denotes the number of degrees of freedom for pid_mesh corresponding to process 0 and the number of elements for pid_mesh corresponding to process 0, respectively.

Table 5.1: Numerical (weak) scalability for ≈ 2500 elements of the subdomain.

Num_subs (N)	16	32	64	128	256	512	1024
Num_dof's	80802	160178	321602	640712	1283202	2562848	5126402
Num_dof's/pid0	5258	5204	5278	5208	5226	5208	5294
Num_elem (n^{el})	40000	79524	160000	319225	640000	1279161	2560000
CG iterations	32	36	41	44	48	49	51
Preproc. time [s]	1.0300	1.8781	3.8109	11.583	16.355	38.977	78.320
FLLOP time [s]	0.98137	0.61717	1.016	2.5041	0.88782	2.2805	2.1751
- QPSSolve time [s]	0.0573	0.1017	0.1349	0.2285	0.2131	0.6067	0.1279
Total time [s]	1.9940	2.4953	4.8269	14.0871	17.2428	41.2575	80.4951

Table 5.2: Numerical (weak) scalability for ≈ 10000 elements of the subdomain.

Num_subs (N)	16	32	64	128	256	512
Num_dof's	321602	640712	1283202	2562848	5126402	10242338
Num_dof's/pid0	20502	20434	20486	20416	20434	20462
Num_elem (n^{el})	160000	319225	640000	1279161	2560000	5116644
CG iterations	40	44	51	53	55	56
Preproc. time [s]	3.6178	7.3234	13.574	29.392	58.771	135.36
FLLOP time [s]	1.7109	1.5028	2.0642	1.7587	2.6025	2.2453
- QPSSolve time [s]	0.3073	0.2980	0.5192	0.5331	0.7822	0.4972
Total time [s]	5.3287	8.8262	15.6382	31.1507	61.3735	137.6053

Table 5.3: Numerical (weak) scalability for ≈ 22500 elements of the subdomain.

Num_subs (N)	16	32	64	128	256
Num_dof's	722402	1441602	2884802	5766408	11529602
Num_dof's/pid0	45750	45780	45640	45752	45728
Num_elem (n^{el})	360000	719104	1440000	2879809	5760000
CG iterations	45	52	56	61	63
Preproc. time [s]	7.4018	16.719	30.508	64.833	138.52
FLLOP time [s]	3.0786	2.5328	2.8002	2.9233	3.7791
- QPSSolve time [s]	2.0011	1.4514	1.5123	1.7170	2.0350
Total time [s]	10.4804	19.2518	33.3082	67.7563	142.2991

The maximum of the number of subdomains (cores) in Tables 5.1 - 5.3 is the maximum of the number of subdomains (cores), for which the implementation runs. For more subdomains it gets a limit of the store size, which is caused by high memory usage in `libMesh`, especially by the mesh generator. Total times are increasing, the growth is linear (considering to the number of subdomains).

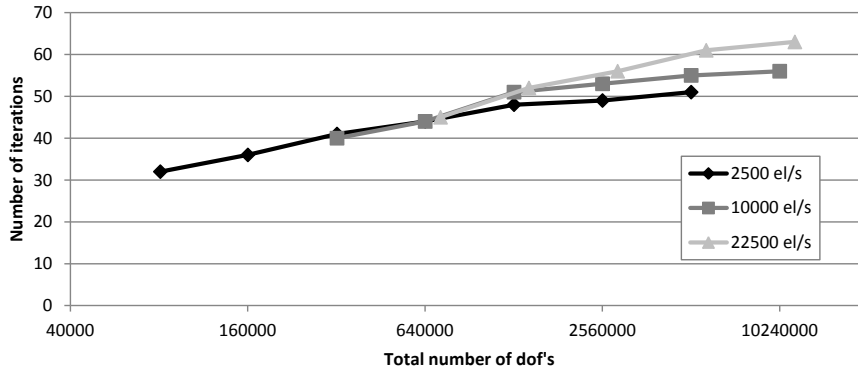


Figure 5.2: Number of iterations.

The preprocessing time means time of the `libMesh` part of code including all our assembling functions. How we can see in Figure 5.3 and 5.4 our implementation is not scalability. The time begins increase very much from 256 subdomains. This is caused by large amount of time, which is necessary for a building of mesh including creation, partitioning and communication of the mesh. However this is not surprising, because the default `SerialMesh` is used in `libMesh`. The time results would be better for using `ParallelMesh`, but this mesh is currently still in development. The parallel scalability is achieved for example for the time of the `assemble_elasticity` function, which assembles the stiffness matrix and the load vector, see Figure 5.5.

Table 5.4: Parallel (strong) scalability for 1000000 elements.

Num_subs (N)	16	32	64	128	256	512	1024
Num_dof's	2004002	2004002	2004002	2004002	2004002	2004002	2004002
Num_dof's/pid0	122600	63300	31770	15966	8104	4106	2098
Num_elem/pid0	60677	31245	15623	7797	3907	1952	977
Preproc. time [s]	23.491	21.213	21.416	22.131	23.604	49.881	36.488
- Ass_elast time [s]	0.9286	0.3758	0.1893	0.0938	0.0475	0.0243	0.0119
FLLOP time [s]	10.147	3.782	2.480	1.3262	1.169	3.3329	5.1302
Total time [s]	33.638	24.995	23.896	23.4572	24.773	53.2139	41.6182

Table 5.5: Parallel (strong) scalability for 4000000 elements.

Num_sub (N)	16	32	64	128	256	512	1024
Num_dof's	8008002	8008002	8008002	8008002	8008002	8008002	8008002
Num_dof's/pid0	502584	251360	127600	63398	31816	16066	8088
Num_elem/pid0	250210	124875	63211	31242	15622	7832	3898
Preproc. time [s]	93.489	85.834	84.701	85.297	86.765	100.96	113.99
- Ass_elast time [s]	3.2027	1.5033	0.7476	0.4056	0.2252	0.0915	0.0499
FLLOP time [s]	4.8515	17.478	10.342	4.1835	3.1084	2.8939	2.121
Total time [s]	98.3405	103.312	95.043	89.4805	89.8734	103.8539	116.111

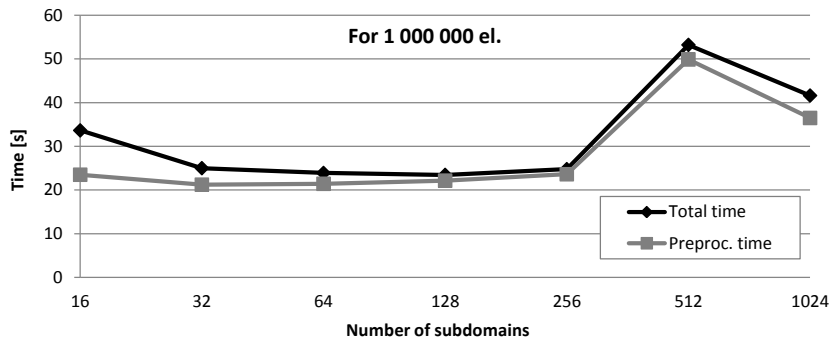


Figure 5.3: Total time and preprocessing time for 1000000 elements.

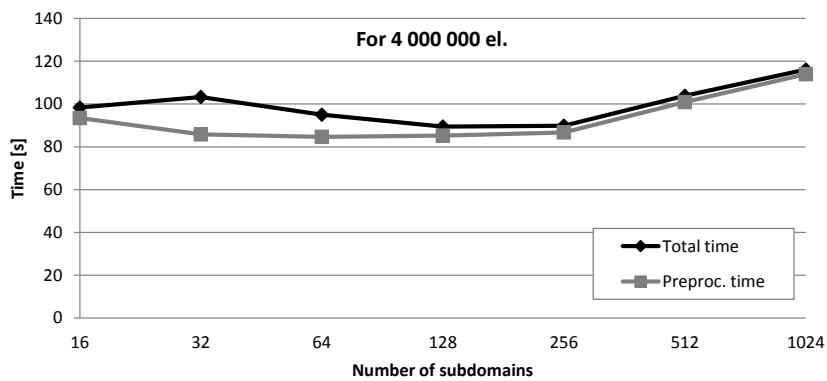


Figure 5.4: Total time and preprocessing time for 4000000 elements.

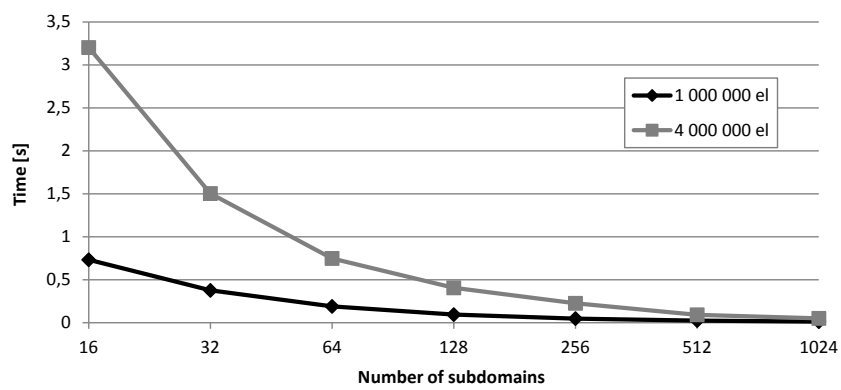


Figure 5.5: Time of the assemble_elasticity function.

6 Conclusion

The main aim of this thesis was to implement the input data generation in FETI methods in `libMesh` library on mesh imported from selected commercial software packages and make an interface to FLLOP library. The implementation was done for the static elasticity problem on the rectangular mesh generated by `libMesh`. The mesh wasn't imported from external software packages because of problems with supporting of mesh formats in `libMesh` and with correct reading of the input file in `libMesh`. The interface to the FLLOP library was made and so was showed that the FLLOP library can be connected to arbitrary FEM generic software, which is totally independent on the FLLOP library.

At the beginning of the thesis the dynamic elasticity problem, its weak formulation and its spatial and time discretization were described. There was also described the static elasticity problem in brief. The Total-FETI (FETI) and the FETI-DP methods were presented in the next section. Then a introduction to the `libMesh` library was made, some key data structures used in this library were presented and main functions of our implementation were described. In the last section numerical experiments were made.

The results of numerical experiments showed us that the mesh generator in `libMesh` is limited by store size, the library is not scalability for the default `SerialMesh` and the `ParallelMesh` is currently still in development and so is not applicable for massive parallel computations. The conclusion is that the `libMesh` library currently is not utilisable for more than hundreds of cores. However there has been a small number of the FEM generic softwares with the same functionality as `libMesh`, which support massive parallel computations. More information about massively parallel finite element generation is presented in [31], [32].

There are many things to finish in the data generic implementation. In future work I would like to focus on finishing and improvement of this implementation. I would also like to deal with FETI methods and FEM generic softwares in general.

Pavla Jirůtková

7 References

- [1] M. Merta, *Newmarkova metoda pro numerické řešení počátečních problémů 2. řádu*. Diploma thesis, VŠB - TU Ostrava, 2009.
- [2] R. Blaheta, *Matematické modelování a metoda konečných prvků*. Matematika pro inženýry 21. století (reg. č. CZ.1.07/2.2.00/07.0332), 2012; Chap. 6.
- [3] Introduction to Finite Element Methods, Department of Aerospace Engineering Sciences, University of Colorado at Boulder [online],
< <http://www.colorado.edu/engineering/CAS/courses.d/IFEM.d/> >
- [4] J. E. Flaherty, *Course Notes - Finite Element Analysis, Finite Element Approximation, Sec. 4.2*. [online],
< <http://www.cs.rpi.edu/~flaherje/pdf/fea4.pdf> >
- [5] Introduction to Finite Element Methods, Department of Aerospace Engineering Sciences, University of Colorado at Boulder, *Part II: Mathematical Formulation of Finite Elements, Chap. 16: The Isoparametric Representation*. [online],
< <http://www.colorado.edu/engineering/cas/courses.d/IFEM.d/IFEM.Ch16.d/> >
- [6] Wikipedia - the free encyclopedia, Gaussian quadrature [online],
< http://en.wikipedia.org/wiki/Gaussian_quadrature >
- [7] D.A. Dunavant, *High degree efficient symmetrical Gaussian quadrature rules for the triangle*. International Journal of Numerical Methods in Engineering, 21:1129-1148, 1985.
(Presented also in Table 6.3.1 in < <http://www.cs.rpi.edu/~flaherje/pdf/fea6.pdf> >.)
- [8] J. G. de Jalón, E. Bayo, *Kinematic and Dynamic Simulation of Multibody Systems: The Real-Time challenge*. Springer-Verlag, New-York, ISBN 0-387-94096-0, 1994; 440 pp, Sec. 7.1.5.
- [9] K.-J. Bathe, *Finite Element Procedures*. Cambridge, 2006; p. 780.
- [10] Z. Dostál, D. Horák, R. Kučera, *Total FETI - an easier implementable variant of the FETI method for numerical solution of elliptic PDE*. Communications in Numerical Methods in Engineering, 22, 2006.
- [11] C. Farhat, F-X. Roux, *An Unconventional Domain Decomposition Method for an Efficient Parallel Solution of Large-Scale Finite Element Systems*. SIAM Journal on Sci. and Stat. Computing 13, 1992; s. 379–396.
- [12] T. Kozubek, V. Vondrak, M. Mensik, D. Horak, Z. Dostal, V. Hapla, P. Kabelikova, M. Cermak, *Total FETI domain decomposition method and its massively parallel implementation*. Advances in Engineering Software, DOI: 10.1016/j.advengsoft.2013.04.001, 2013.

-
- [13] P. Jirůtková, *Paralelizace řešení eliptických okrajových úloh pomocí TFETI metody rozložení oblastí*. Bachelor thesis, VŠB - TU Ostrava, 2012.
- [14] Z. Dostál, T. Kozubek, A. Markopoulos, M. Mensik, *Cholesky decomposition and a generalized inverse of the stiffness matrix of a floating structure with known null space*. Applied Mathematics and Computation, 2011; **217**:6067-6077.
- [15] T. Brzobohatý, Z. Dostál, P. Kovář, T. Kozubek, A. Markopoulos, *Cholesky decomposition with fixing nodes to stable evaluation of a generalized inverse of the stiffness matrix of a floating structure*. IJNME, DOI: 10.1002/nme.3187.
- [16] Z. Dostál, *Optimal Quadratic Programming Algorithms: With Applications to Variational Inequalities (1st ed.)*. Springer Publishing Company, Incorporated, 2009.
- [17] J. Haslinger, T. Kozubek, R. Kučera, G. Peichl, *Projected Schur complement method for solving non-symmetric saddle-point systems arising from fictitious domain approach*. Numerical Linear Algebra with Applications 2007; 14(9):713-739.
- [18] Ch. Farhat, M. Lesoinne, P. LeTallec, K. Pierson, D. 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, 2001; 50:1523-1544.
- [19] A. Klawonn, O. Rheinbach, *Some computational results for robust FETI-DP methods applied to heterogeneous elasticity problems in 3D*. Springer Berlin Heidelberg, 2007; pp 389-396.
- [20] O. Rheinbach, *Parallel Iterative Substructuring in Structural Mechanics*. Archives of Computational Methods in Engineering, Springer Netherlands, 2009; pp 425-463.
- [21] M. Jarošová, *Efektivní implementace některých algoritmů kvadratického programování pro řešení rozsáhlých úloh*. Doctoral thesis, VŠB-TU Ostrava, 2010.
- [22] The libMesh library [online],
< <http://libmesh.sourceforge.net> >
- [23] The libMesh library, Download, Installation Instructions [online],
< <http://libmesh.sourceforge.net/installation.php> >
- [24] B. S. Kirk, J. W. Peterson, R. H. Stogner, G. F. Carey, *libMesh: A C++ Library for Parallel Adaptive Mesh Refinement/Coarsening Simulations*. Engineering with Computers, 22(3-4):237-254, 2006.
- [25] B. S. Kirk, J. W. Peterson, R. H. Stogner, *The libMesh Finite Element Library: a case for Object-Oriented High-Performance Computing*. PRACE Training, 2013.
< http://www.training.prace-ri.eu/uploads/tx_pracetmo/libmesh.pdf >

-
- [26] The libMesh library, Class Docs, Class Elem [online],
< http://libmesh.sourceforge.net/doxygen/classlibMesh_1_1Elem.php >.
- [27] The libMesh library, Class Docs, Class System [online],
< http://libmesh.sourceforge.net/doxygen/classlibMesh_1_1System.php >.
- [28] IT4Innovations, Supercomputing for Industry, FLLOP [online],
< <http://industry.it4i.cz/en/products/flop/> >.
- [29] The libMesh library, Examples, Linear Elastic Cantilever [online],
< http://libmesh.sourceforge.net/systems_of_equations_ex4.php >.
- [30] IT4Innovations, Hardware and Services, Anselm support and documentation [online],
< <http://www.it4i.cz/support/?lang=en> >
- [31] T. Heister, M. Kronbichler, W. Bangerth, *Massively Parallel Finite Element Programming*. Springer-Verlag Berlin Heidelberg, EuroMPI 2010, LNCS 6305, DOI: 10.1007/978-3-642-15646-5_13, 2010; pp. 122–131.
- [32] W. Bangerth, C. Burstedde, T. Heister, M. Kronbichler, *Algorithms and Data Structures for Massively Parallel Generic Adaptive Finite Element Codes*. ACM Transactions on Mathematical Software, V. 38, Iss. 2, DOI: 10.1145/2049673.2049678, 2011.