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

Optimal configuration of FETI solvers in HPC

Optimální konfigurace FETI řešičů v HPC

2017

Martin Beseda

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra aplikované matematiky

Zadání diplomové práce

Student:	**Bc. Martin Beseda**
Studijní program:	N2647 Informační a komunikační technologie
Studijní obor:	1103T031 Výpočetní matematika
Téma:	Optimální konfigurace FETI řešičů v HPC Optimal configuration of FETI solvers in HPC
Jazyk vypracování:	čeština

Zásady pro vypracování:

Cílem diplomové práce je nalezení optimálního nastavení Total FETI a Hybridního Total FETI metod z hlediska výkonu, tedy minimalizace výpočetního času pro různé velikosti a typy úlohy strukturální mechaniky.
Klíčovou součástí FETI řešičů dlouhodobě vyvíjených na Katedře aplikované matematiky jsou přímé řídké řešiče.

Prvotními úkoly studenta budou:
- seznámení se s různými implementacemi přímých řídkých řešičů,
- seznámení se s paralelními architekturami v HPC prostředí a studium vlivu negativních efektů, jako např. NUMA efektu, na výkon FETI řešičů,
- studium paralelní implementace FETI řešičů a jejich chování při paralelizaci pro sdílenou paměť (OpenMP) nebo distribuovanou paměť (MPI),
- seznámení se s metodami měření spotřeby elektrické energie v prostředí HPC,
- prvotní evaluace spotřeby energie pro různé části FETI řešičů, jež může v budoucnu vést k sestavení modelu pro spotřeby el. energie jako alternativy k modelu výpočetního času.

Poté, co se student seznámí s danými nástroji a architekturou, bude jeho primárním úkolem sestavit matematický model beroucí v úvahu
- různé kombinace hybridní paralelizace - MPI + OpenMP (kombinace počtu MPI procesů a OpenMP threadů na uzel)
- různé FETI metody (Total FETI a Hybridní Total FETI)
- různé předpodmiňovače,
- různé dekompozice.

Výsledný matematický model popisujícího chování FETI řešiče jako jednoho celku bude použit k nalezení jeho optimálního nastavení pro různé úlohy. Tento model bude kalibrován pro implementaci FETI řešičů v knihovně ESPRESO.

Seznam doporučené odborné literatury:

Manuály k jednotlivým řídkým řešičům a literatura dle 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: 01.09.2016

Datum odevzdání: 28.04.2017

[illegible]

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

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA
Fakulta elektrotechniky a informatiky
1D16
OSTRAVA

[illegible]

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

I hereby declare that this master's thesis was written by myself. I have quoted all the references I have drawn upon.

Ostrava, April 24, 2017 ....[illegible]............

I hereby agree to the publishing of the master's thesis as per s. 26, ss. 9 of the Study and Examination Regulations for Master's Degree Programmes at VŠB – Technical University of Ostrava.

Ostrava, April 24, 2017 ....Beseda........

I would like to thank to Ing. Lubomír Říha, PhD. and prof. Ing. Tomáš Kozubek, PhD. for supervising of my thesis and many useful suggestions about technical aspects.

Further I thank to Ing. Martina Litschmannová, PhD. and Ing. Jan Kracík, PhD. for introduction into the topic of regression analysis.

I also thank to Ing. Alexandros Markopoulos, PhD., Ing. Michal Běloch, Ing. Michal Béreš, Ing. Radim Sojka and Ing. Václav Ryška for many useful suggestions and a continual support.

And last but not least I thank to IT4Innovations National Supercomputing Center for the possibility of using Salomon cluster for my computations performed under the project OPEN-7-46.

Abstrakt

Hlavním tématem této diplomové práce byla predikce časové náročnosti metod Total FETI a Hybrid Total FETI implementovaných v knihovně ESPRESO. Druhým tématem byl průzkum energetické náročnosti kninhovny ESPRESO pro plánovaný model spotřeby.

Model časové náročnosti byl vytvořen pomocí zobecněné lineární regrese s použitím jazyku R pro implementaci softwaru potřebného ke zpracování naměřených dat. Energetická náročnost byla zkoumána pomocí nástrojů MERIC a RADAR implementovaných v rámci projektu READEX, který patří do programu Horizon2020.

Model, který popisuje časovou náročnost je užitečný pro odhadování optimálních nastavení i když pro větší hodnoty není predikce v současné verzi příliš přesná. MERIC a RADAR byly použity pro evaluaci energetické úspory pro několik hardwarových a aplikačních parametrů.

Model, který popisuje časovou náročnost bude implementován do knihovny ESPRESO, aby bylo možné automaticky odhadovat optimální nastavení pro minimální čas výpočtu bez předchozích testů. Podobně, model spotřeby bude sestaven za pomoci výsledků získaných pomocí nástrojů MERIC a RADAR a později implementován do knihovny ESPRESO také, čímž bude schopná odhadovat nejen optimální konfiguraci pro minimální čas výpočtu, ale také pro minimální spotřebu energie.

Klíčová slova: diplomová práce, výkon, run-time, model, hybridní paralelizace, energetická spotřeba, ESPRESO, READEX, Horizon2020, FETI

Abstract

The main objective of this thesis was the performance prediction of Total FETI and Hybrid Total FETI methods implemented in ESPRESO library. The secondary objective was the investigation of energy requirements of ESPRESO for a planned consumption model.

The performance model was created by generalized linear regression, using R-language for implementation of the software needed to process measured data. The energy consumption was investigated using MERIC and RADAR tools implemented under READEX project in Horizon2020 programme.

The model describing performance is useful for estimations of optimal settings, although the fit is not very precise for larger values. MERIC and RADAR were used to evaluate energy savings for multiple hardware and application parameters.

The performance model will be implemented into the ESPRESO library, so it will be able to estimate optimal settings for minimal run-time without demanding prior tests. Similarly, the consumption model will be assembled using results obtained by MERIC and RADAR and later implemented into the ESPRESO too, making it able to estimate not only optimal settings for minimal run-time, but for the minimal energy consumption too.

Key Words: master thesis, performance, run-time, model, hybrid parallelization, energy consumption, ESPRESO, READEX, Horizon2020, FETI

Contents

List of symbols and abbreviations

AIC	–	Akaike Information Criterion
CG	–	Conjugate Gradients
CV	–	Cross-Validation
DOF	–	Degrees Of Freedom
FETI	–	Finite Element Tearing and Interconnect Method
g.o.f.	–	Goodness of fit
HTFETI	–	Hybrid Total FETI Method
LpO	–	Leave-p-Out validation
MAPE	–	Mean absolute percentage error
MLE	–	Maximum Likelihood Estimation
OLS	–	Ordinary Least Squares method
PDE	–	Partial Differential Equation
p.d.f.	–	Probability density function
PPCG	–	Projected Preconditioned Conjugate Gradients
PPG	–	Parallel Problem Generator
QP	–	Quadratic programming
RHS	–	Right-Hand Side (of an equation)
RSS	–	Residual Sum of Squares
sMAPE	–	Symmetric Mean Absolute Percentage Error
TFETI	–	Total FETI method

List of Figures

List of Tables

List of Listings

List of Algorithms

1 Introduction

1.1 Motivation

The motivation behind this thesis was, that there exist several implementations of different FETI methods, to be introduced in the next section, but it is very difficult for users of these libraries to use them efficiently. This thesis describes a performance model for one particular implementation of Total FETI and Hybrid Total FETI in the ESPRESO library.

This performance model allows any inexperienced user to set-up a FETI solver in sense of (1) domain decomposition and (2) combination of OpenMP and MPI parallelization to minimize the solver runtime because it takes into account the parallel scalability of the solver.

The model is composed from several partial models each of whom estimates the run-time of a separate region. This allows not only the easier customization of the model for future versions of ESPRESO, but also the detailed predictions about how single regions behave for different application settings. This can help a new user understand, if the preprocessing phase will be generally more time-demanding than the CG solver for some settings, or to estimate, which region is the cause of an unexpected delay etc.

Another hot issue nowadays is the energy consumption in supercomputing. Considering ESPRESO is meant to be an exascale parallel solver which is able to effectively use "tens or hundreds of thousands of cores", as it is written on its website[1], its energy requirements could become really high while solving large problems.

This led us to the idea of the consumption model, which would estimate the energy requirements of significant regions of ESPRESO in a similar way to the performance model. While this consumption model is out of the scope of this thesis, the progress has been made in the energy measurement. We have developed tools MERIC and RADAR for this purpose. The former is used to measure important properties, like the run-time, an arithmetical intensity or the energy consumption of significant regions in applications and the latter is used to evaluate those data and to generate report containing information about optimal settings.

This information could be used as a training data for the consumption model, as we are now able to measure both the run-time and the energy requirements on several different types of hardware, thanks to those tools.

Both the performance and the consumption model could be also implemented to ESPRESO in the future, so it would be able to estimate differences among different settings, like problem decomposition, different parallelization or different preconditioners and choose the estimated optimal settings automatically.

This thesis introduces the core computational methods of ESPRESO - TFETI and HTFETI in the first section, where the description focuses on the parts recognized as the significant regions which are later modeled. In Section 2 and 3 there are briefly explained statistical "tools"

[1]`http://espreso.it4i.cz/`

and methods used for assembling the performance model and some hardware effects related to the performance, respectively. The model problem used for measurements is described in the Section 4, together with details about performed measurements. Finally in Section 5 there is the performance model described in detail. Finally, Section 6 briefly introduces the energy measurement tools mentioned above.

1.2 FETI Methods

The FETI (Finite Element Tearing and Interconnecting) method was introduced by Farhat and Roux in [10]. It is the method used to solve linear problem described by elliptic PDEs in parallel. It decomposes the spatial domain into non-overlapping sub-domains, "glued" together by Lagrange multipliers. The primal problem is then transferred, or rather reduced, into the smaller and better conditioned constrained QP problem. This problem is then solved iteratively, typically using some form of CG method.

The FETI method was modified (described in [9]), when it was observed, that the mathematical treatment of floating sub-domains together with the Projected Conjugate Gradients method (described by the pseudo-code 1 and in [30] in detail) is equivalent to the assembling and solution of a coarse problem, which accelerates convergence and guarantees, that the FETI algorithm performance is independent of a number of sub-domains.

For practical purposes there was required some method for automatic identification of kernels of stiffness matrices of subdomains, because these kernels are used during the elimination of primal variables and in definition of the coarse grid projectors. The algorithm for identification was described by Farhat and Gérardin in [8] as the combination of Cholesky factorization and the singular value decomposition.

The kernel identification problem partially persisted in real-world problems even with this improvement, because of the rounding errors. This led to the improved version of FETI called FETI-DP described in [11]. The difference is, that FETI-DP makes all the stiffness matrices of sub-domain invertible by enforcing the continuity of displacements at the corners on primal level. This is advantageous for some applications, because FETI-DP is efficiently preconditioned and so it scales better, than FETI. The main drawback is, that the coarse grid defined by the corners is less efficient without additional preconditioning than the one defined by the rigid body motions, as described in [7]. The FETI-DP algorithm is also much more difficult to implement, because it requires a special treatment of corners.

The new algorithm, which addresses this two drawbacks is called Total FETI (TFETI).

1.2.1 Total FETI (TFETI)

TFETI was introduced by Dostal et al. in [6] as the new variant of the FETI method trying to eliminate the disadvantages of Total-DP. Its basic idea is the simplified inversion of stiffness matrices of subdomains with both "gluing" the subdomains and enforcing Dirichlet boundary conditions using Lagrange multipliers.

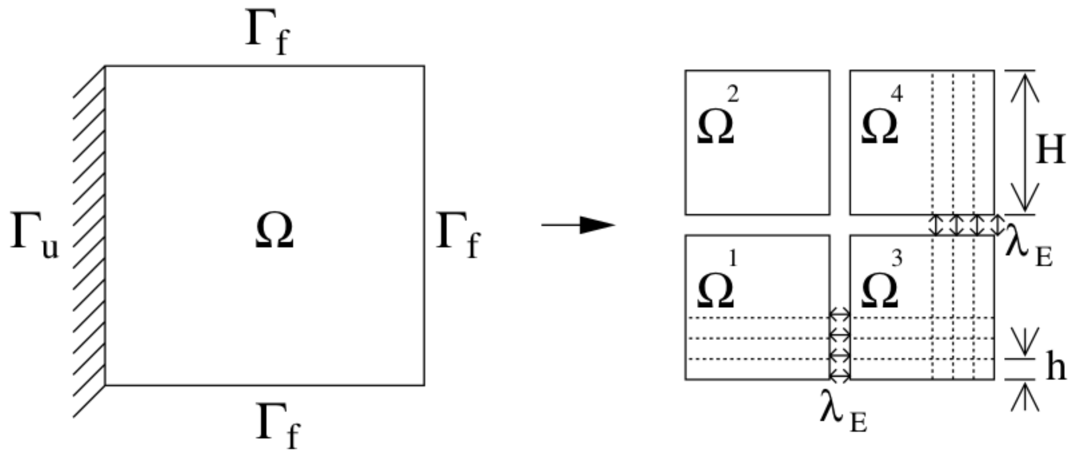


Figure 1: 2D domain decomposition and discretization in TFETI (source [6])

Let us consider an elastic body comprised by the domain $\Omega \subset \mathbb{R}^2 \vee \Omega \subset \mathbb{R}^3$. Let us further assume, that the boundary Γ of Ω is divided into to disjoint parts Γ_u and Γ_f, as can be seen in fig. 1, such that $\Gamma = \overline{\Gamma}_u \cup \overline{\Gamma}_f$. Dirichlet boundary conditions are prescribed on Γ_u and Neumann boundary conditions are prescribed on Γ_f. Let us finally presume, that the problem will be well-posed.

According to the description of the FETI domain decomposition described in [9], we will decompose Ω into n subdomains $\Omega^1, \ldots, \Omega^n$ as can be seen in fig. 1. The local stiffness matrix, sub-domain force (i.e. the local load vector), displacement vectors, the "gluing" matrix and the vector of constraints corresponding with individual sub-domains will be denoted as K_i, f_i, u_i, B_i and c_i, respectively.

Considering the objects above, we will get the optimization problem

$$\begin{aligned} \min_{u} \quad & \frac{1}{2}u^T K u - u^T f \\ \text{s.t.} \quad & Bu = c, \end{aligned} \tag{1}$$

where

$$u = \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix}, K = \begin{bmatrix} K_1 & & \\ & \ddots & \\ & & K_n \end{bmatrix}, f = \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}, B = [B_1, \ldots, B_n] \tag{2}$$

Let us assume in this section, that $c = o$. The "gluing" matrix $B \in \mathbb{R}^{m \times n}$ together with the constraint vector $c \in \mathbb{R}^m$ enforces both the continuity on the artificial boundary among sub-domains and the displacement on the boundary Γ_u with prescribed Dirichlet boundary conditions.

The prescribed imposed Dirichlet condition $u_j = u_D(i)$ on the boundary Γ_u is enforced by the row b_j in the B matrix, where all the elements equal zero except the one element at the position i. In the constraints vector $c_j = u_D(i)$. And because of our earlier assumption, in this section $u_D(i) = 0$.

The continuity on the artificial boundary between two sub-domains in the individual node is also enforced with the row b_j, where -1 and 1 are located in the corresponding positions to the node index and $c_j = 0$ as described in [30].

In the original FETI method it is assumed, that the Dirichlet conditions on sub-domains are inherited from the original problem, so *the defect of the stiffness matrices* K_i (i.e. the number of columns of the null-space) possibly vary from 0 at the boundary sub-domains K_i matrices to the maximum at the inner floating sub-domains K_i matrices. In 3D elasticity problems the maximum value is 6, corresponding to the number of independent rigid body motions (both translational and rotational motions in axes X, Y and Z).

TFETI copes with this situation in a different way. All the sub-domain stiffness matrices are treated like if there were no prescribed displacements and those displacements are instead enforced by matrix B, as it is described above. The improvement over FETI is, that all the stiffness matrices now have the same defect, which is known (e.g. 3 for 2D and 6 for 3D elasticity problems).

The remaining procedure is basically identical with the original FETI. The Lagrange function corresponding to the optimization problem (1) is

$$\mathcal{L}(u, \lambda) = \frac{1}{2} u^T K u - f^t u + \lambda^T B u \tag{3}$$

and it is known from [2], that the Lagrangian is equivalent to the saddle point problem

$$\text{Find } \left(\overline{u}, \overline{\lambda}\right), \text{ so that} \mathcal{L}\left(\overline{u}, \overline{\lambda}\right) = \sup_{\lambda} \inf_{u} \mathcal{L}(u, \lambda). \tag{4}$$

Now, when we fix λ, the Lagrangian $\mathcal{L}(\cdot, \lambda)$ will be convex in the first variable and the minimizer u will satisfy

$$Ku - f + B^T \lambda = 0. \tag{5}$$

The Eq. (5) has solution if and only if it satisfies

$$R^T \left(f - B^T \lambda\right) = 0. \tag{6}$$

Kernels R_i corresponding to the local stiffness matrices K_i are known, e.g. considering 2D elasticity problem and the sub-domain Ω^i discretized to m_i nodes with the coordinates $(x_k, y_k), k = 1, \ldots, m_i$, R_i will look like this

$$R_i = \begin{bmatrix} (R_i^1)^T \\ \vdots \\ (R_i^m)^T \end{bmatrix} \tag{7}$$

$$R_i^k = \begin{bmatrix} 1 & 0 & -y_k \\ 0 & 1 & x_k \end{bmatrix} \tag{8}$$

Using matrices R_i there can be assembled the block-diagonal basis R of the kernel of K like

$$R = \begin{bmatrix} R_1 & & \\ & \ddots & \\ & & R_n \end{bmatrix} \tag{9}$$

Let us denote a generalized inversion of the stiffness matrix $K^\dagger$ (described by Eq. (10)) and let us assume, that λ satisfies the Eq. (6).

$$KK^\dagger K = K \tag{10}$$

If u solves the Eq. (5), then there is a vector α satisfying

$$u = K^\dagger \left(f - B^T\lambda\right) + R\alpha \tag{11}$$

Substituting the Eq. (11) into the Eq. (4) will produce a new optimization problem

$$\begin{aligned} \min_{\lambda} \quad & \frac{1}{2}\lambda^T BK^\dagger B^T\lambda - \lambda^T BK^\dagger f \\ \text{s.t.} \quad & R^T\left(f - B^T\lambda\right) = 0, \end{aligned} \tag{12}$$

Let us denote

$$F = BK^\dagger B^T \tag{13}$$

$$G = R^T B^T \tag{14}$$

$$d = BK^\dagger f \tag{15}$$

$$e = R^T f, \tag{16}$$

so we can rewrite the problem (12) and homogenize it as

$$\begin{aligned} \min_{\lambda} \quad & \frac{1}{2}\lambda^T F\lambda - \lambda^T d \\ \text{s.t.} \quad & G\lambda = 0. \end{aligned} \tag{17}$$

The problem (17) has the same saddle point structure as (4), however, (17) is smaller size problem and F is better conditioned than K, so this new problem is much easier to solve numerically.

Let us introduce a new projector

$$P = I - G^T \left(GG^T\right)^{-1} G, \tag{18}$$

I being the identity matrix. Using the projector P we can rewrite the problem (17) to the equivalent problem (described in [6])

$$\begin{aligned} \min_{\lambda} \quad & \frac{1}{2}\lambda^T PFP\lambda - \lambda^T Pd \\ \text{s.t.} \quad & G\lambda = 0, \end{aligned} \tag{19}$$

which can be finally solved by *Projected Preconditioned Conjugate Gradients method (PPCG)*, described in detail in [27], [30]. The PPCG algorithm is given by the pseudo-code

Algorithm 1 PPCG algorithm (source [30])

```
Require: $\lambda^0_{Ker} = 0$
         $r^0 = d$
1: for k = 1, 2, ... until convergence do
2:     Project: $w^{k-1} = Pr^{k-1}$;
3:     Precondition: $z^{k-1} = \bar{F}^{-1}w^{k-1}$;
4:     Re-project: $y^{k-1} = Pz^{k-1}$;
5:     $\beta^k = (y^{k-1})^{\mathrm{T}} w^{k-1} / (y^{k-2})^{\mathrm{T}} w^{k-2}$;    $(\beta^1 = 0)$
6:     $p^k = y^{k-1} + \beta^k p^{k-1}$;    $(p^1 = y^0)$
7:     $\alpha^k = (y^{k-1})^{\mathrm{T}} w^{k-1} / (p^k)^{\mathrm{T}} Fp^k$;
8:     $\lambda^k_{Ker} = \lambda^{k-1}_{Ker} + \alpha^k p^k$;
9:     $r^k_{Ker} = r^{k-1} - \alpha^k Fp^k$;
10: end for
```

where $\bar{F}^{-1}$ is a preconditioner, described in detail in the Sec. 1.2.3.

1.2.2 Hybrid Total FETI (HTFETI)

In the TFETI method there exist limitations because of the size of the coarse problem, or rather its memory requirements. In HTFETI the key idea is the aggregation of a small number of neighboring sub-domains into so called *clusters* using Lagrange multipliers, which results into a smaller coarse problem. Using Lagrange multipliers here means, that the TFETI method is used twice - both on cluster and subdomain levels. This approach is advantageous because it allows to extend the parallelization of the local problem, because it reduces the memory requirements. On the other hand, the negative effect is the worsened convergence rate compared to the TFETI method. This problem is reduced by the transformation of the basis as stated below.

The model problem for HTFETI is the same as the one for the TFETI, i.e. the elastic body with prescribed Dirichlet and Neumann boundary conditions. The problem is described in detail in Sec. 1.2.1 and by the Eq. (1).

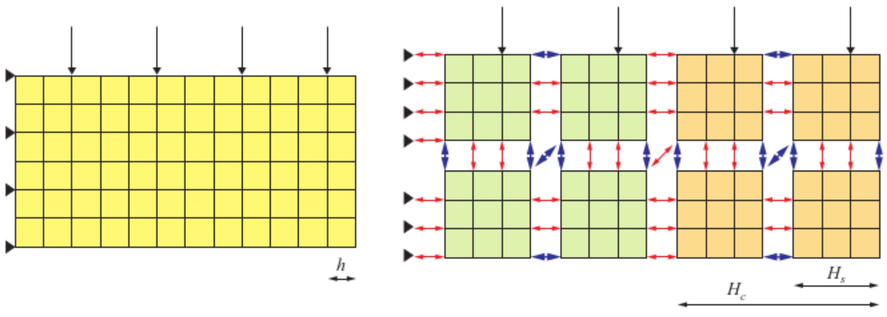


Figure 2: HTFETI decomposition of a problem (source [17])

First of all, the body of the problem is teared from the part of the boundary with the Dirichlet boundary condition as it is done in the TFETI.

To decompose the problem let domain Ω be decomposed into n_c non-overlapping clusters with diameter H_c and let each cluster be decomposed into n_s sub-domains with diameter H_s, as can be seen in Fig. 2. So, the total number of sub-domains is given by the formula

$$n = n_c \cdot n_s. \tag{20}$$

Then we have to introduce so called "gluing" conditions to enforce the continuity of displacements and the imposed Dirichlet boundary conditions. Those properties are enforced by the matrix $B \in \mathbb{R}^{m \times n}$ and the vector c given by formulas

$$B = \begin{bmatrix} B_g^T \\ B_D^T \end{bmatrix} \tag{21}$$

$$c = \begin{bmatrix} o^T \\ c_D^T \end{bmatrix}, \tag{22}$$

m being the number of Lagrange multipliers, which is significantly smaller than the number of degrees of freedom. The matrix B is constructed from values -1, 0 and 1, similarly to TFETI. The values of the solution u associated with more than one domain coincide, when $B_g u = o$ and the values of the solution u on the boundary with the Dirichlet condition satisfy $B_D u = c_D$.

The problem has the same structure as in the TFETI (see the Eq. (1)), but in HTFETI there is used a slightly modified form of this problem:

$$\begin{aligned} \min_{u} \quad & \frac{1}{2}u^T K u - u^T f \\ \text{s.t.} \quad & B_0 u = c_0, \\ & B_1 u = c_1 \end{aligned} \tag{23}$$

The first part $B_0 u = c_0$ of the equality constraints consisting of m_0 equalities enforces the continuity in the sub-domain corner nodes of each cluster and the second part $B_1 u = c_1$ consisting of m_1 equalities enforces the continuity among the rest of the sub-domain interfaces and the prescribed Dirichlet condition.

The problem (23) can be described with respect to the Karush-Kuhn-Tucker conditions as

$$\left[\begin{array}{cc|c} K & B_0^T & B_0^T \\ B_0 & O & O \\ \hline B_1 & O & O \end{array}\right] \left[\begin{array}{c} u \\ \lambda_0 \\ \hline \lambda_1 \end{array}\right] = \left[\begin{array}{c} f \\ c_0 \\ \hline c_1 \end{array}\right]. \tag{24}$$

This equation can be rewritten further as

$$\left[\begin{array}{c|c} \tilde{K} & \tilde{B}^T \\ \tilde{B} & O \end{array}\right] \begin{bmatrix} \tilde{u} \\ \lambda \end{bmatrix} = \begin{bmatrix} \tilde{f} \\ \tilde{c} \end{bmatrix}, \tag{25}$$

where $\tilde{K}, \tilde{B}, \tilde{u}, \tilde{f}$ and $\tilde{c}$ respect the block structure highlighted in the Eq. (24).

The first equation of the system (25) has solution if and only if

$$\tilde{R}\left(\tilde{f} - \tilde{B}^T \tilde{\lambda}\right) = o \tag{26}$$

$$\tilde{R} = \begin{bmatrix} R_c^1 & & \\ & \ddots & \\ & & R_c^{n_c} \\ O & O & O \end{bmatrix} \tag{27}$$

$$R_c^i = \begin{bmatrix} R^{1,i} \\ \vdots \\ R^{n_s,i} \end{bmatrix}, \tag{28}$$

where columns of the matrix $\tilde{R}$ span the null space of $\tilde{K}$ and $R^{p,i}, p = 1, \ldots, n_s$ means p-th sub-domain of the i-th cluster.

Kernel bases of the sub-domains are known beforehand, so the blocks $R^{p,i}$ can be assembled directly from the segments R_k defined for the k-th mesh node in 2D problem by the Eq. (8).

To eliminate the primal variables $\tilde{u}$ from the singular system described by the first equation in the problem (25), we will use the generalized inverse matrix $\tilde{K}^\dagger$ satisfying $\tilde{K}\tilde{K}^\dagger\tilde{K} = \tilde{K}$. If $\tilde{u}$ is a solution to the first equation in the problem (25), then there exists a vector $\tilde{\alpha}$ such that

$$\tilde{u} = \tilde{K}^\dagger \left(\tilde{f} - \tilde{B}^T\tilde{\lambda}\right) + \tilde{R}\tilde{\alpha}. \tag{29}$$

After substitution of (29) into the second equation of (25) and the application of (26) we get the system

$$\begin{bmatrix} \tilde{B}\tilde{K}^\dagger\tilde{B}^T & -\tilde{B}\tilde{R} \\ -\tilde{R}^T\tilde{B}^T & O \end{bmatrix} \begin{bmatrix} \tilde{\lambda} \\ \tilde{\alpha} \end{bmatrix} = \begin{bmatrix} \tilde{B}\tilde{K}^\dagger\tilde{f} - \tilde{c} \\ -\tilde{R}^T\tilde{f} \end{bmatrix} \tag{30}$$

Let us denote

$$\tilde{F} = \tilde{B}\tilde{K}^\dagger\tilde{B}^T \tag{31}$$

$$\tilde{G} = -\tilde{R}^T\tilde{B}^T \tag{32}$$

$$\tilde{g} = \tilde{B}\tilde{K}^\dagger\tilde{f} - \tilde{c} \tag{33}$$

$$\tilde{e} = -\tilde{R}^T\tilde{f}, \tag{34}$$

so we can rewrite the system (30) as

$$\begin{bmatrix} \tilde{F} & \tilde{G}^T \\ \tilde{G} & O \end{bmatrix} \begin{bmatrix} \tilde{\lambda} \\ \tilde{\alpha} \end{bmatrix} = \begin{bmatrix} \tilde{g} \\ \tilde{e} \end{bmatrix}. \tag{35}$$

After that we introduce the projector

$$\tilde{P} = I - \tilde{G}^T \left(\tilde{G}\tilde{G}^T\right)^{-1} \tilde{G} \tag{36}$$

onto the kernel of $\tilde{G}$. After the application of $\tilde{P}$ to the Eq. (35) we get

$$\begin{aligned} &\tilde{P}\tilde{F}\tilde{\lambda} = \tilde{P}\tilde{g} \\ \text{s.t.} \quad &\tilde{G}\tilde{\lambda} = \tilde{e} \end{aligned} \tag{37}$$

and after homogenization of the constraints, considering $\tilde{\lambda}_{Ker} \in Ker\tilde{G}$ we get

$$\tilde{P}\tilde{F}\tilde{\lambda}_{Ker} = \tilde{P}\left(\tilde{g} - \tilde{F}\tilde{\lambda}_{Im}\right) \tag{38}$$

$$\tilde{\lambda}_{Im} = \tilde{G}^T \left(\tilde{G}\tilde{G}^T\right)^{-1} \tilde{e}, \tag{39}$$

where $\tilde{\lambda} = \tilde{\lambda}_{Ker} + \tilde{\lambda}_{Im}$. We know, that $\tilde{P}\tilde{F}$ is a SPD matrix (proof in [17]). Because of that the problem (39) can be efficiently solved by the PPCG method.

In every single PPCG iteration we compute $\tilde{x} = \tilde{K}^{\dagger}\tilde{b}$, $\tilde{b}$ being a given vector. We compute it by solving the system

$$\tilde{K}\tilde{x} = \tilde{b} \Longleftrightarrow \begin{bmatrix} K & B_0^T \\ B_0 & O \end{bmatrix} \begin{bmatrix} x_0 \\ \mu_0 \end{bmatrix} = \begin{bmatrix} b_0 \\ d_0 \end{bmatrix} \tag{40}$$

using the TFETI method again. By a substitution of the variable

$$x_0 = K^{\dagger}\left(b_0 - B_0^T\mu_0\right) + R_0\beta_0 \tag{41}$$

into the second equation in the problem (40) we get the system

$$\begin{bmatrix} F_0 & G_0^T \\ G_0 & O \end{bmatrix} \begin{bmatrix} \mu_0 \\ \beta_0 \end{bmatrix} = \begin{bmatrix} g_0 \\ e_0 \end{bmatrix}, \tag{42}$$

where

$$F_0 = B_0K^{\dagger}B_0^T \tag{43}$$

$$G_0 = -R_0^TB_0^T \tag{44}$$

$$g_0 = B_0K^{\dagger}b_0 - d_0 \tag{45}$$

$$e_0 = -R_0^Tb_0 \tag{46}$$

$$R_0 = \begin{bmatrix} R_1 & & \\ & \ddots & \\ & & R_n \end{bmatrix}, \tag{47}$$

n is described by the Eq. (20) and $R^j, j = 1, \ldots, n$ being the same as $R^{p,i}$ (described above), with $j = (i-1)\cdot n_s + p$. To obtain the vector $\begin{bmatrix} x^T \\ \mu^T \end{bmatrix}$, both systems (40) and (42) are subsequently solved in following steps

$$\beta_0 = S_{\alpha}^{\dagger}\left(G_0F_0^{-1}g_0 - e_0\right) \tag{48}$$

$$\mu_0 = F_0^{-1}\left(g_0 - G_0^T\beta_0\right) \tag{49}$$

$$x_0 = K^{\dagger}\left(b_0 - B_0^T\mu_0\right) + R_0\beta_0, \tag{50}$$

wher a singular Schur complement

$$S_{\alpha} = G_0^TF_0^{-1}G_0^T \tag{51}$$

appears, as described in [28].

To obtain the full-rank matrix $\overline{G}_0 = -\overline{R}_0^T B_0^T$ we have to introduce the matrix $\overline{R}_0$, which is obtained from the matrix R_0 by deleting the block columns corresponding to the last sub-domains of all clusters. The matrix F_0 is SPD on $\mathbb{R}^{m_0}$ as described by Lemma 2 in [17].

Now let $\overline{e} = -\overline{R}_0^T b_0$. Using the fact, that F_0 is SPD and replacing G_0 and e_0 by $\overline{G}_0$ and $\overline{e}_0$ in (42) gives us the well-defined saddle point problem, which can be solved again by PPCG (described by the pseudo- code 1), or by some direct solver, because of its small size.

Now the size of the coarse problem is reduced in comparison with TFETI as mentioned above, but the convergence worsened, which can be improved by the transformation of the basis described in detail in [17].

1.2.3 Preconditioners

To improve the convergence of the solver there are several preconditioners available in ESPRESO. Lumped preconditioner and Dirichlet preconditioner are being included in the models in this thesis, so they are briefly described below. More detailed information can be found in [27].

The local stiffness matrix K_i can be divided into four blocks

$$K_i = \begin{bmatrix} K_{i,oo} & K_{i,ob} \\ K_{i,bo} & K_{i,bb} \end{bmatrix}, \tag{52}$$

o being the index of ordinary (i.e. internal) DOFs and b being the index of boundary DOFs.

Dirichlet preconditioner

The Dirichlet preconditioner is given by formula

$$F^{D^{-1}} = \sum_{i=1}^{n} B_i \begin{bmatrix} 0 & 0 \\ 0 & S_{i,bb} \end{bmatrix} (B_i)^T, \tag{53}$$

$S_{i,bb}$ being the Schur complement of the block $K_{i,oo}$, considering the sub-domain K_i. It is described by the formula

$$S_{i,bb} = K_{i,bb} - (K_{i,ob})^T (K_{i,oo})^{-1} K_{i,ob}. \tag{54}$$

The Dirichlet preconditioner is is the most numerically efficient preconditioner for FETI, but is is also the most computationally demanding one.

Lumped preconditioner

The Lumped preconditioner is given by the formula

$$F^{L^{-1}} = \sum_{i=1}^{n} B_i \begin{bmatrix} 0 & 0 \\ 0 & K_{i,bb} \end{bmatrix} (B_i)^T \tag{55}$$

and it is less computationally expensive than the Dirichlet preconditioner, but it is also less numerically efficient.

Scaling matrix

Redundant constraints are allowed, so the matrix B_i may not have a full column rank. In that case, the scaling matrix A_i has to be introduced to be able to use preconditioners described above.

A_i is the symmetric, diagonal matrix and its size equals the number of dual variables. Its diagonal entries are given by the weight vector w. The value of w_i equals $\frac{1}{2}$, when the associated node was split into two duplicate nodes. Generally, the value of $w_i = \frac{1}{m}$ for every associated DOF i, m being the number of sub-domains associated with the node.

Using the scaling matrix A_i, the modified Dirichlet preconditioner $F_{mod}^{D^{-1}}$ and the modified Lumped preconditioner $F_{mod}^{L^{-1}}$ are given by formulas

$$F_{mod}^{D^{-1}} = \sum_{i=1}^{n} A_i B_i \begin{bmatrix} 0 & 0 \\ 0 & S_{i,bb} \end{bmatrix} (B_i)^T A_i \tag{56}$$

$$F_{mod}^{L^{-1}} = \sum_{i=1}^{n} A_i B_i \begin{bmatrix} 0 & 0 \\ 0 & K_{i,bb} \end{bmatrix} (B_i)^T A_i. \tag{57}$$

The Projected Preconditioned Conjugate Gradient method used with those preconditioners is described in [27].

2 Description of statistic tools used for models

2.1 Generalized linear model (GLM)

GLM is the generalization of the *general linear model*, introduced in [23].

The general linear model is described by the Eq. (58), y being the vector of values of the response variable, β being the vector of coefficients (so called *regression parameters*, x being the vector of predictor variables and ϵ being the *residual* (i.e. the error term). The model is linear with respect to coefficients β_i, so the predictor variables $x_1, \ldots, x_n$ can be of higher powers, multiplied with each other etc. and the model is still considered linear.

$$y_i = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n + \epsilon_i \tag{58}$$

In general linear models we assume, that errors $\epsilon_1, \ldots, \epsilon_n$ are independent, holding the property described by the Eq. (59).

$$\epsilon_i \sim \mathcal{N}\left(0, \sigma^2\right) \tag{59}$$

While a really useful tool, the general linear model has its limitations. The first one of them is the case, when the response variable is somehow restricted, because the linear model is unbounded. The second one is the case, when the variance of the response variable depends on the mean. GLM addresses both of these cases.

A generalized linear model consists of several parts. The first of them, a *random component* is a distribution from the exponential family described by the conditional expected value μ_i (i.e. the mean) of the response variable y, given x of the chosen distribution. A random component is described by the Eq. (60).

$$\mathrm{E}\left(y|x\right) = \mu_i \tag{60}$$

The second one is a linear predictor η, described by the Eq. (61).

$$\eta_i = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n \tag{61}$$

The third one is the *link function* g. It describes, how the mean $\mu_i = \mathrm{E}\left(y_i\right)$ depends on the linear predictor. It is described by the Eq. (62), y being the vector of values of the response variable.

$$g\left(\mu_i\right) = \eta_i \tag{62}$$

The last of them is the *variance function* V. It explains, how the variance *var* depends on the mean μ_i. It is described by the Eq. (63), ϕ being the constant parameter.

$$\mathrm{Var}\left(y_i\right) = \phi V\left(\mu_i\right) \tag{63}$$

Finally, the general linear model given by the Eq. (59) is a special case of GLM. Its properties are given by equations (64) and (65).

$$g\left(\mu_i\right) = \mu_i \tag{64}$$

$$V\left(\mu_i\right) = 1 \tag{65}$$

2.2 Fisher's scoring algorithm

The numeric method for solving maximum likelihood equations. It was first presented in [20].

Here we can see a brief sketch of its derivation.

At first, let's define the *likelihood function* $\mathcal{L}$ as the function of the parameters of a model, while the observations are fixed. Likelihood function is described by the equation (66), where x are the observations and θ is a vector of parameter values.

$$\mathcal{L}\left(\theta|x\right) = P\left(x|\theta\right) \tag{66}$$

When X is a random variable following continuous probability distribution, then the equation (66) can be more specific with $f\left(x|\theta\right)$ being the p.d.f. of the value $x \in X$ for the parameter values θ. This case is described by equation (67).

$$\mathcal{L}\left(\theta|x\right) = f\left(x|\theta\right) \tag{67}$$

When there are multiple independent and identically distributed observations $x_1, \ldots, x_n$, the likelihood function is defined by the Eq. (69), N being the number of observations.

$$\mathcal{L}\left(\theta|x_1, \ldots, x_n\right) = f\left(x_1, \ldots, x_n|\theta\right) \tag{68}$$

$$= \prod_{i=1}^{N} f\left(x_i|\theta\right) \tag{69}$$

The natural logarithm of the likelihood function is called a *log-likelihood* function. It is sometimes more convenient to use it, because it allows us to use sum instead of product in the definition of function, as can be seen from Eq. (70).

$$\ln \mathcal{L}\left(\theta|x_1, \ldots, x_n\right) = \sum_{i=1}^{N} \ln f\left(x_i|\theta\right) \tag{70}$$

Another important term to define is a *score function* $\mathcal{V}$, which was introduced in [4]. It describes the sensitivity of the likelihood function $\mathcal{L}(\theta|x)$ to its parameter θ. Technically, it is a gradient of the log-likelihood function with respect to θ, as it is described by the Eq. (71).

$$\mathcal{V}(\theta|x) = \frac{\partial \ln \mathcal{L}(\theta|x)}{\partial \theta} \tag{71}$$

Last term we need to define to explain the Fisher's scoring algorithm is the *Maximum likelihood estimate method (MLE).* It is defined as the finding of θ which maximizes the average likelihood function or its natural logarithm. We will call such vector of parameters the *maximum likelihood estimator θ_{mle}.*

θ_{mle} of the average likelihood function equals θ_{mle} of the *average log likelihood function $\overline{\ell}$* (defined by Eq. (73)), because the natural logarithm is a monotonically increasing function.

$$\begin{aligned} \overline{\ell}(\theta|x_1,\dots,x_n) &= \frac{\ln \mathcal{L}(\theta|x_1,\dots,x_n)}{N} && (72) \\ &= \frac{\sum_{i=1}^{N} \ln f(x_i|\theta)}{N} && (73) \end{aligned}$$

For some models, the maximum does not exist and so does not θ_{mle}, on the contrary, there can be multiple estimators θ_{mle} which maximize the (log-)likelihood function. And for some other models θ_{mle} can not be found as an explicit function of the observed data $x_1,\dots,x_n$, so we must approximate it numerically. The maximum likelihood estimator $\hat{\theta}_{mle}$, i.e. the whole Maximum likelihood method is defined by the Eq. (74).

$$\{\theta_{mle}\} \subseteq \left\{ \underset{\theta\in\mathbb{R}^n}{\arg\max}\ \overline{\ell}(\theta|x_1,\dots,x_n) \right\} \tag{74}$$

Now when we have MLE method defined, we can proceed with numerical approximation of $\hat{\theta}_{mle}$.

To start off we can write the score function $\mathcal{V}$ down as the first-order Taylor polynomial about θ_0 in Eq. (76), where θ_0 is also the starting point of the algorithm.

$$\begin{aligned} \mathcal{V}(\theta) &\approx \mathcal{V}(\theta_0) + \nabla_\theta \Big|_{\theta=\theta_0} \mathcal{V}(\theta|x_1,\dots,x_n) && (75) \\ &\approx \mathcal{V}(\theta_0) + \nabla^2_\theta \Big|_{\theta=\theta_0} \sum_{i=1}^{N} \ln f(x_i|\theta)\cdot(\theta-\theta_0) && (76) \end{aligned}$$

Let's substitute $\theta = \theta_{mle}$ in Eq. (77).

$$\mathcal{V}(\theta_{mle}) \approx \mathcal{V}(\theta_0) + \nabla^2_\theta \Big|_{\theta=\theta_0} \sum_{i=1}^{N} \ln f(x_i|\theta)\cdot(\theta_{mle}-\theta_0) \tag{77}$$

$\mathcal{V}$ is the derivative of the log-likelihood function, hence $\mathcal{V}(\theta_{mle}) = 0$, under the assumption $\mathcal{L}(\theta_{mle}|x_1, \dots, x_n)$.

Moreover, we can factor out -1 in Eq. (77), so we get an *observed information matrix* $\mathcal{J}(\theta)$ at θ_0, as we can see in Eq. (78).

$$\underbrace{\mathcal{V}(\theta_{mle})}_{0} \approx \mathcal{V}(\theta_0) - \underbrace{\left(-\nabla_\theta^2 \Big|_{\theta=\theta_0} \sum_{i=1}^{N} \ln f(x_i|\theta)\right)}_{\mathcal{J}(\theta_0)} \cdot (\theta_{mle} - \theta_0) \tag{78}$$

After several steps we get the approximate value for θ_0 in Eq. (82).

$$0 \approx \mathcal{V}(\theta_0) - \mathcal{J}(\theta_0) \cdot (\theta_{mle} - \theta_0) \tag{79}$$

$$\mathcal{V}(\theta_0) \approx \mathcal{J}(\theta_0) \cdot (\theta_{mle} - \theta_0) \tag{80}$$

$$\mathcal{J}^{-1}(\theta_0) \cdot \mathcal{V}(\theta_0) \approx \theta_{mle} - \theta_0 \tag{81}$$

$$\theta_{mle} \approx \theta_0 + \mathcal{J}^{-1}(\theta_0) \cdot \mathcal{V}(\theta_0) \tag{82}$$

The Eq. (82) can be generalized to "n+1 formula" described in Eq. (83). This equation is the formula for *Newton-Raphson* method.

$$\theta_{n+1} = \theta_n + \mathcal{J}^{-1}(\theta_0) \cdot \mathcal{V}(\theta_0) \tag{83}$$

The observed information $\mathcal{J}(\theta)$ converges in probability to its expected value because of the law of large numbers, hence $\mathcal{J}(\theta)$ can be replaced with it. The expected value of the observed information is called Fisher information $\mathcal{I}(\theta)$. And the method using it instead of the observed information is called Fisher's scoring. This substitution is described by the Eq. (85).

$$\theta_{n+1} = \theta_n + \mathrm{E}(\mathcal{J}(\theta_0))^{-1} \cdot \mathcal{V}(\theta_0) \tag{84}$$

$$\theta_{n+1} = \theta_n + \mathcal{I}^{-1}(\theta_0) \cdot \mathcal{V}(\theta_0) \tag{85}$$

2.3 Evaluation of a goodness-of-fit and a predictive capability

Akaike information criterion (AIC)

AIC is a coefficient used for comparison of quality of statistical models. It was first described in [1].

AIC does not test models in the sense of testing a null hypothesis, so it does not tell us anything about the absolute g.o.f. or a quality of a model in general. Instead, it estimates the quality of each model relative to the others.

$$AIC = 2k - 2\ln(\mathcal{L}^*) \tag{86}$$

It is described by the Eq. (86), k being the number of parameters in the model and $\mathcal{L}^*$ being the maximum value of the likelihood function, i.e. $\mathcal{L}^* = P\left(x|\theta^*\right)$, where x are the observed data and θ^* is the vector of parameter values which maximize the likelihood function $\mathcal{L}$.

As can be seen from the Eq. (86), AIC decreases with increasing g.o.f. described by $\mathcal{L}^{ast}$ and simultaneously increases with increasing number of parameters k. This property penalizes models with large number of parameters, which is useful, because they tend to over-fit.

So, the best one of compared statistical models according to AIC is the one with the smallest coefficient.

Null and residual deviance

Deviance is a quantity describing the g.o.f. of a statistical model (M).

It generalizes the idea of using the *residual sum of squares* (RSS) in the *Ordinary least squares* (OLS) method described by Eq. (87) and (88), where y_i is the vector of observed values, x_i is the vector of values of predictor variables and θ is the vector of parameters, based on which we are minimizing the RSS. Deviance is equal to RSS for linear models.

$$RSS\left(\theta\right) = \sum_{i=1}^{n}\left(y_i - x_i^T\theta\right) \tag{87}$$

$$OLS\left(\theta\right) = \underset{\theta\in\mathbb{R}^n}{\arg\min}\, RSS\left(\theta\right) \tag{88}$$

Deviance is used in cases, where fitting is performed by Maximum likelihood estimation (see Eq. (74)) and it is described by Eq. (89), where y is a vector of observations, $\hat{\theta}_M$ is a vector of fitted parameters of model M and $\hat{\theta}_S$ is a vector of fitted parameters of a *saturated model.*

By saturated model we mean the statistical model with a parameter for every observation, so it fits perfectly, but it has no predictive ability.

$$D\left(y\right) = -2\left(\ln p\left(y\middle|\hat{\theta}_M\right) - \ln p\left(y\middle|\hat{\theta}_S\right)\right) \tag{89}$$

There are another two important terms for purposes of this thesis - *null deviance* and *residual deviance.* Null deviance is a deviance of the model constituted just from the *intercept* (i.e. the constant) = so called *null model.* Residual deviance is a deviance of the proposed model, i.e. the model we are examining for the g.o.f.

The proposed model will have more terms than just one (unlike the null model), which implies bigger flexibility, i.e. better fit of the proposed model. Deviance decreases with increasing g.o.f., same as RSS, so, if the proposed model is well-fitting, the residual deviance will be much smaller, than the null deviance.

Symmetric mean absolute percentage error (sMAPE)

sMAPE is an accuracy measure providing percentage errors. Similarly to deviance - the lower value means a better fit. It is based on a Mean absolute percentage error (MAPE), which is given by the Eq. (90), N being the number of observations, y being the vector of observed values and $\hat{y}$ being the vector of predicted values.

$$MAPE = \frac{100}{N} \sum_{i=1}^{N} \frac{|y_i - \hat{y}_i|}{|y_i|} \; [\%] \tag{90}$$

The main disadvantage of MAPE is, that it is asymmetric, i.e. the errors are not treated equally. When the predicted value is larger than the observed one, MAPE punishes the error more severely, than the other case - lower predicted value, than the observation.

Because of this problem there were several types of symmetric MAPE developed. The latest one is was introduced in [32] and it is described by the Eq. (91).

$$SMAPE = \frac{\sum_{i=1}^{N} |\hat{y}_i - y_i|}{\sum_{i=1}^{N} (y_i + \hat{y}_i)} \tag{91}$$

This version solves the problem of symmetry of errors punishment well, but for all version of MAPE persists another one - the case, when the observed or predicted value equals zero. In that case the error reported by MAPE increases rapidly, so it could skew the result significantly.

Because of this sMAPE should be used mostly in combination with other accuracy measures.

Root mean squared error (RMSE)

RMSE is a measure of differences between values predicted by a statistical model and observed values (so, RMSE is decreasing with better fit). It represents a sample standard deviations of these differences. It serves as a measure of prediction accuracy and so it can be used to compare models while using the same data set for training (different data sets can not be compared, because RMSE is scale-dependent). Its distinctive property is, that RMSE punishes large errors more aggressive, than MAPE or sMAPE.

RMSE is described by the Eq. (92), where N is the number of observations, y is the vector of observations and $\hat{y}$ is the vector of predicted values.

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2} \tag{92}$$

Cross-validation (CV)

Cross-validation is the method of evaluating the prediction capability of the model. Its basic principle is pretty simple. A data-set is divided into two groups - a *training set* and a *test set.* The first of them is used for training the model and the latter is used to evaluate the prediction capability of the model. The g.o.f. on the test set, i.e. the the prediction capability of the model can be described with variety of measures. For purposes of this thesis we will use MAPE and RMSE, both of which are described in Sec. 2.3.

There are several strategies, how to divide the full data-set and each of them has its advantages and disadvantages.

The first strategy is called *leave-p-out validation (LpO).* It decomposes the full data-set into two subsets such that the test set contains p observations and the rest of them belongs to the training set. LpO is, so called, *exhaustive cross-validation*, i.e. the validation process is repeated on all ways to decompose the full data-set. LpO requires both the training and the testing phase repeat $\binom{N}{p}$ times, N being the number of observation in the full set.

The advantage of this strategy is, that it examines the behavior of the model in detail. The downside is, that it is really computationally expensive, to the point, where it can become actually infeasible.

The least computationally demanding version of LpO strategy is a *leave-one-out validation (LoO).* It is the special case, when $p = 1$, so the number of validation steps is $\binom{N}{1}$. When N is large, than this could make a vast difference - we can see the example for $N = 100$, $p_1 = 20$ and $p_2 = 1$ from equations (93) and (94).

$$\binom{100}{20} \approx 5.4 \cdot 10^{20} \tag{93}$$

$$\binom{100}{1} = 100 \tag{94}$$

Another two strategies are non-exhaustive, i.e. unlike LpO they do not check all combinations possible. Instead, they try just some smaller amount of steps with intention to decrease the computational time consumption. As such, they are approximations of LpO.

The first one of non-exhaustive cross-validation strategies is the *k-fold cross validation.* It decomposes the full data-set into k equal-sized subsets. One of these subsets is used as a test set and the remaining ones are used for the training as can be seen in the Fig. 3. When $k = N$, then k-fold validation "degenerates" to the LoO validation.

The validation process itself is performed k times, so that every subset is used for testing just once. Finally, those k results are averaged to get the single estimation.

Model validation using, for example, the data-set with 100 observations and $k = 10$ will perform only 10 steps, so it is very easy to compute and still all the observations from the full data-set are used for the training and the test, which is its main advantage.

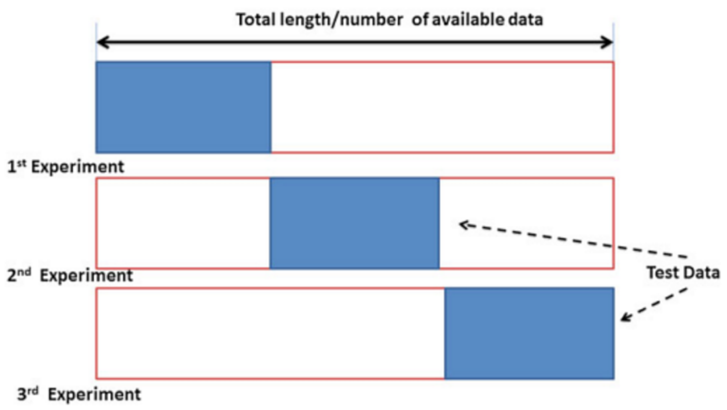


Figure 3: 3-fold cross validation (source [26])

One of the disadvantages is, that the subsets can be very different from each other. We can compensate this difficulty with selecting subsets, so that all of them will have similar mean value of the response variable. This strategy is called a *stratified k-fold validation.*

Another two disadvantages of the k-fold cross validation are, that we can not choose the size of test sets independently and the number of validation steps is fixed, so it can still become problem when we are working with very large data-sets.

The problem with number of validation steps is addressed by another cross-validation strategy called *Repeated random sub-sampling validation.* The point of this strategy is, that the full data-set is decomposed into the training and test set randomly in every step, so that every observation can be used multiple times, or contrarily, not even once, as can be seen in fig. 4. Its simplest form, i.e. the case with just one step is called the *holdout method.* The downside of this case is, that its evaluations can have a high variance, so that we could not reliably decide about the model predictive capability.

The main advantage of repeated random sub-sampling is, that the number of steps is not bounded, so we can perform an arbitrary number of validation steps. With large number of steps the result converges to the result of leave-p-out validation. On the contrary, its disadvantage is, that observations are used different number of times or not at all, so, when we do not perform enough validation steps, the final result could be skewed.

Finally, if response values of the observations differ dramatically, we can use the stratified form of this strategy as well, i.e. we can select such subsets, that have approximately the same mean of the response variable values.

Wald test

Wald t. is used to test for a real value of a parameter. In GLM, it is used to find out whether some term in a statistical model is significant (i.e. we test parameters for the zero value) or it can

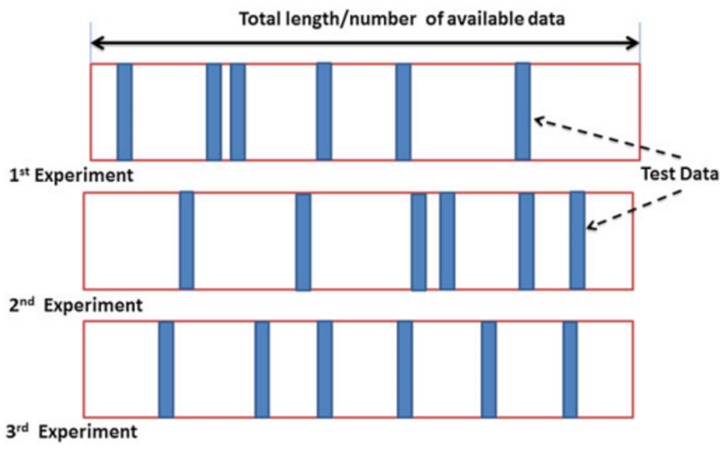


Figure 4: Random sub-sampling validation (source [26])

be omitted without changing predicted values substantially. Its null and alternative hypothesis are described by equations (95) and (96), respectively, where β_i is the i-th coefficient of the GLM described by the Eq. (62) and μ_0 is the value β_i coefficient is tested for.

$$H_0 : \beta_i = \mu_0 \tag{95}$$

$$H_A : \beta_i \neq \mu_0 \tag{96}$$

For significance testing we are usually interested in $\mu_0 = 0$, so that when we do not reject H_0 hypothesis, then we can try to omit the i-th term from the model.

The test statistic is described by the Eq. (97), where $se\left(\beta_i\right)$ is the *standard error* of the i-th term.

$$t = \frac{\beta_i}{se\left(\beta_i\right)} \tag{97}$$

The vector of standard errors is given by the Eq. (99), where X, described by the Eq. (98), is the random vector whose elements $X_1, \ldots, X_n$ are random scalars.

$$X = \begin{bmatrix} X_1 \\ \vdots \\ X_n \end{bmatrix} \tag{98}$$

$$se(\beta_i) = \sqrt{\Sigma_{ii}} \tag{99}$$

Σ described by the Eq. (100) is the *matrix of covariance*, where $\text{Cov}\left(X_i, X_j\right)$ is the *covariance* (described by the Eq. (101)).

$$\Sigma = \begin{bmatrix} \mathrm{Cov}\left(X_1, X_1\right) & \mathrm{Cov}\left(X_1, X_2\right) & \cdots & \mathrm{Cov}\left(X_1, X_n\right) \\ \mathrm{Cov}\left(X_2, X_1\right) & \mathrm{Cov}\left(X_2, X_2\right) & \cdots & \mathrm{Cov}\left(X_2, X_n\right) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{Cov}\left(X_n, X_1\right) & \mathrm{Cov}\left(X_n, X_2\right) & \cdots & \mathrm{Cov}\left(X_n, X_n\right) \end{bmatrix} \tag{100}$$

$$\mathrm{Cov}\left(X_i, X_j\right) = \mathrm{E}\left[\left(X_i - \mathrm{E}\left(X_i\right)\right)\left(X_j - \mathrm{E}\left(X_j\right)\right)\right] \tag{101}$$

The p-value is finally obtained from a χ^2-distribution, normal distribution, or, for the purposes of this thesis, t-distribution.

3 Hardware effects related to performance modeling

Processors with architecture Haswell-EP used in Salomon supercomputer (see description in Sec. 4.2) have 12-core dies. It is designed as it is depicted in Fig. 5, i.e. there are two bi-directional rings in one die. Eight cores with respective slices of L3 cache, QPI interconnect, a memory controller and the PCIe controller are connected to the first ring.

The remaining four cores with their respective L3 cache slices and the second memory controller are connected to the second bi-directional ring. The rings are then connected together by two queues. Threads from every core can access all DDR4 memory units. It is clear that accessing the local memory, in this case memory on the local ring, is faster, than accessing the remote one (using the memory controller on remote ring). This, so called, *Non-Uniform Memory Access (NUMA)* effect can significantly affect the behavior of parallel applications.

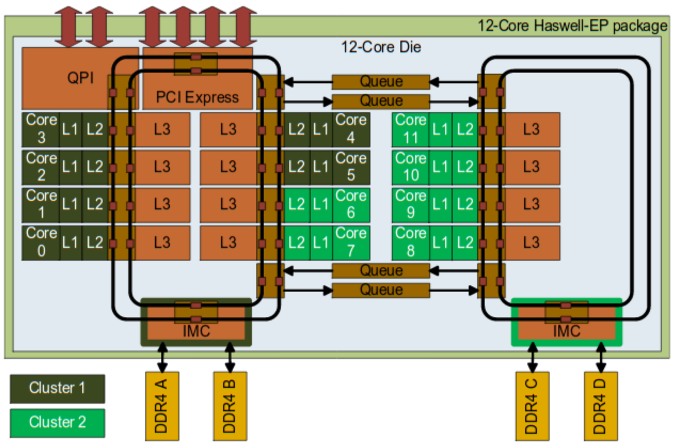


Figure 5: Haswell-EP socket with 12 cores (source [21]

This leads us to two types of a possible delay caused by accessing remote memory units. Both types of delay are usually called *NUMA effect* and the first type is caused by reading from RAM memory connected closely to the other ring, e.g. accessing DDR4 D memory by thread bounded to the Core 3, considering the Fig. 5. The delay is caused by data transfer through some of the queues between the two bi-directional rings.

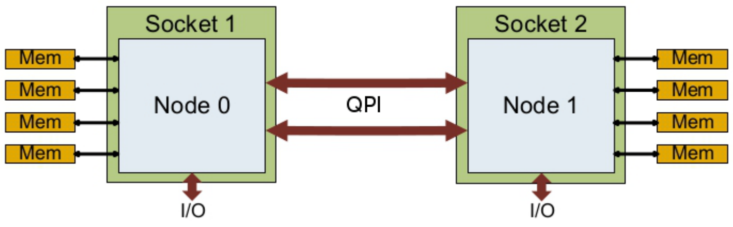


Figure 6: Dual socket Xeon E5 v3 system - one NUMA node per socket (source [21]

The second and the more significant type of NUMA effect is caused by accessing memory attached to another socket (i.e.processor) in a multi CPU/socket system. As can be seen from the Fig. 6, data have to be transferred through the QPI link which causes much more significant delay than the first case, as described in [21].

NUMA effect become apparent in the course of modeling TFETI and HTFETI methods in this thesis. MPI processes - threads decomposition represented by *nprocs* parameter was significantly less effective with 1 process and 24 threads than in other cases, so the observations lied on the hyperbolic curve and the terms containing the number of processes were expressed like $\frac{1}{nprocs}$. This effect can be seen in Fig. 7, where the run-time for 1 process is almost four times as high as the one for 2 processes and even 2 and 3 processes are significantly slower, than the other settings.

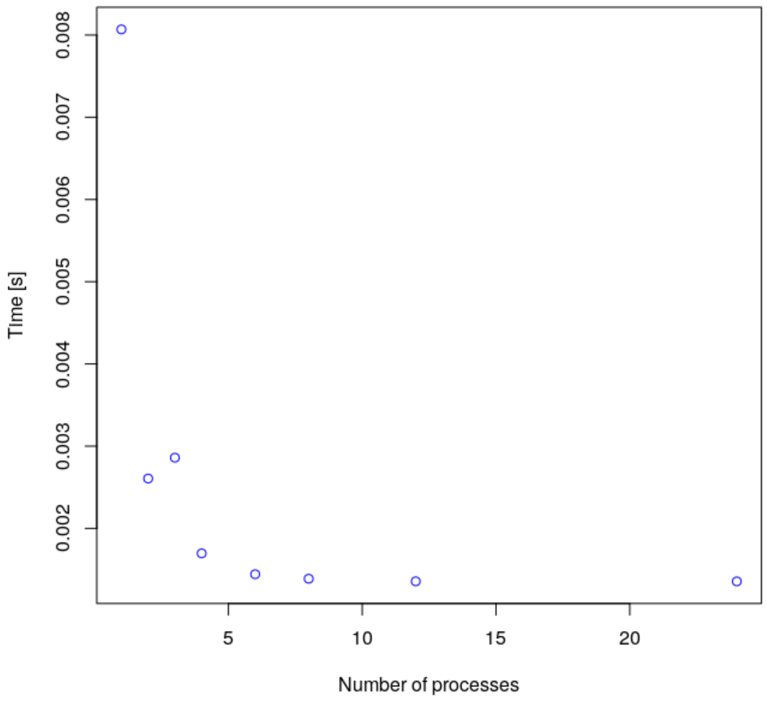


Figure 7: NUMA effect in action of K in TFETI

4 Benchmark problems

ESPRESO supports both loading mesh from file as well as generating its own mesh using benchmark generator, see Fig. 9. All measurements in this thesis were performed using linear elasticity benchmark problem generated by the *Parallel Problem Generator* (described below in this section).

The type of a problem affects the sparsity pattern of the stiffness matrices K. It has an effect on a number of rigid body motions per sub-domain, as described in the Sec. 1.2.1, which determine the size of the coarse problem $\boldsymbol{G^T G}$. For 3D linear elasticity there are 6 RBMs per sub-domain.
Both TFETI and HTFETI methods uses PARDISO [24] as the main direct solver. PARDISO solver was chosen by ESPRESO developers because it is thread safe and also generally one of the fastest one as it was also observed in [12]). Therefore this solver was also used for all performance measurements in this thesis. The effect of following settings was examined:

- Different domain decompositions (always hierarchical 2-level decomposition):
  - HTFETI - decomposition of problem into clusters and decomposition of clusters into sub-domains,
  - TFETI - decomposition into domain groups mapped to one MPI process (corresponds to one cluster in HTFETI) and decomposition into domains in one group - again, it is a hierarchical decomposition, but TFETI algorithm considers it as a 1-level.
- Different sub-domains sizes,
- Different preconditioners (no preconditioner, Lumped prec. and Dirichlet prec.) - details can be found Section 1.2.3.

Additionally in case of HTFETI an effect of assembling the $\boldsymbol{F_0}$ and $\boldsymbol{S_\alpha}$ matrices (their models are presented in Sections 5.1.7 and 5.1.13, respectively) is evaluated.

4.1 Parallel Problem Generator (PPG)

PPG is part of the ESPRESO library and it is designed to enable fast evaluation of the solver for very large problems of linear elasticity (described in [28]). It generates a benchmark problems in form of the steel cube with following properties:

- Volume $\boldsymbol{V = 9000\,mm^3}$
- Elastic modulus $\boldsymbol{E = 2.1 \cdot 10^5\,MPa}$
- Poisson's ratio $\boldsymbol{\mu = 0.3}$
- Density $\boldsymbol{\rho = 7850 kg\,m^{-3}}$

- Gravity constant $\boldsymbol{g = 9.81 m\, s^{-1}}$

The cube is loaded with its own weight in the direction of $\boldsymbol{x_1}$ and it is fixed on the plane $\boldsymbol{x_1 = 0}$ as can be seen in Fig. 8.

The discretization process consists of two steps. The first step it is generating cubical meshes, one per each cluster. The second step is the decomposition of cluster meshes into cubical sub- domains using geometric decomposition. All the discretization is done using 8-node brick elements. Both steps of discretization can be seen in the Fig. 8.

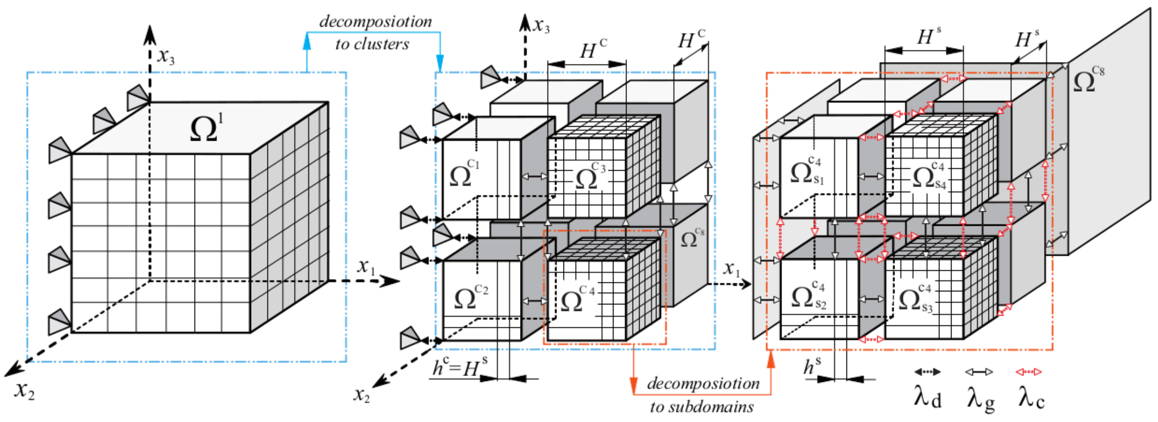


Figure 8: Decomposition of the elasticity problem (steel cube) generated by PPG (source [28])

The matrix assembler then creates following objects according to the mesh data:

- Stiffness matrix $\boldsymbol{K_i}$
- RHS $\boldsymbol{f_i}$
- "Gluing" matrix $\boldsymbol{B_i}$
- Matrix of HTFETI corners $\boldsymbol{B_{0,i}}$
- Set of fixed nodes used for regularization of $\boldsymbol{K_i}$ in TFETI (described in [3])

$\boldsymbol{K_i}$ and $\boldsymbol{f_i}$ are created in parallel, using multiple Cilk++ threads. The matrix $\boldsymbol{B_i}$ is assembled from 3 parts:

- Dirichlet boundary conditions
- "Gluing" among sub-domains (i.e. inside cluster)
- "Gluing" among clusters

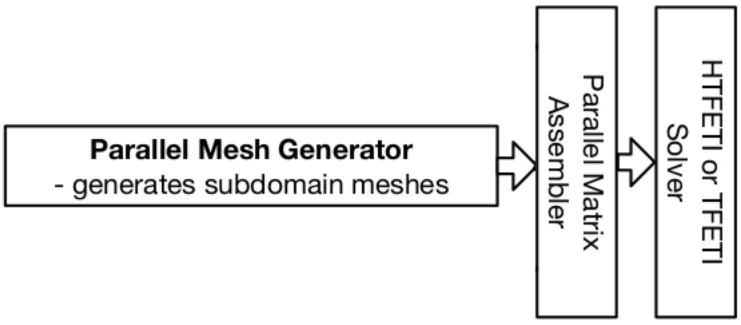


Figure 9: ESPRESO mesh-data input (source [28])

Dirichlet boundary conditions and "gluing" among sub-domains are independent among clusters and row indices are globally renumbered after they are assembled on all clusters. Renumbering is performed by single call of `MPI_Scan()`[22] function.

Assembling the third part, which arranges "gluing" among clusters employs the nearest-neighbor strategy - it performs an exchange of global indices of surface DOF among the neighboring clusters. It is performed in parallel among clusters and its most time consuming operations, like binary search in an array for every surface DOF, can be further parallelized by threads.

4.2 Measurement details

Measurements were performed on Salomon cluster (see [2]) using nodes which hold following properties:

- Operating system CentOS 6.x Linux
- 2 processors per node with 12 cores each (Intel Xeon E5-2680v3 at 2.5 GHz (Haswell architecture))
- 128GB RAM per node, i.e. 5.3 GB per core, DDR4@2133 MHz

Measurements were performed for different number of nodes ***nnodes***, the number of MPI processes per node ***nprocs***, the number of domains per MPI process ***ndoms*** and the number of DOF per domain ***nDOF***. All combinations of the settings presented in the listing bellow were executed and measured:

$$\boldsymbol{nnodes} \coloneqq \{\mathbf{1, 2, 4, 8, 16}\}$$

$$\boldsymbol{nprocs} \coloneqq \{\mathbf{1, 2, 3, 4, 6, 8, 12, 24}\}$$

[2]`https://docs.it4i.cz/salomon/introduction/`

$$\boldsymbol{ndoms} := \{1152, 576, 384, 288\}$$

$$\boldsymbol{nDOF} := \{375, 1029, 2187, 3993, 6591, 10125, 14739\}$$

ESPRESO is configured using a set of parameters in the following format:

```
mpirun -n N ./espreso -p problem.txt 0  A A A  B B B  C C C,
```

where `A A A` is the number of clusters or MPI processes ($\boldsymbol{N = A * A * A}$), i.e. $\boldsymbol{nnodes \cdot nprocs}$, `B B B` is the number of domains per cluster/MPI process and `C C C` is the size of one domain, all of them in 3D. The decomposition is done into cubical subdomains as can be seen in the Fig. 1, 2 and 8.

$$\boldsymbol{nDOF = 3\,(x+1)\,(y+1)\,(z+1)} \tag{102}$$

Clusters					
nprocs	**1 node**	**2 nodes**	**4 nodes**	**8 nodes**	**16 nodes**
1	$\mathbf{1\times1\times1}$	$\mathbf{2\times1\times1}$	$\mathbf{2\times2\times1}$	$\mathbf{2\times2\times2}$	$\mathbf{2\times2\times4}$
2	$\mathbf{1\times1\times2}$	$\mathbf{2\times1\times2}$	$\mathbf{2\times2\times2}$	$\mathbf{2\times2\times4}$	$\mathbf{4\times2\times4}$
3	$\mathbf{1\times1\times3}$	$\mathbf{2\times1\times3}$	$\mathbf{2\times2\times3}$	$\mathbf{2\times3\times4}$	$\mathbf{4\times3\times4}$
4	$\mathbf{1\times2\times2}$	$\mathbf{2\times2\times2}$	$\mathbf{2\times2\times4}$	$\mathbf{4\times2\times4}$	$\mathbf{4\times4\times4}$
6	$\mathbf{1\times2\times3}$	$\mathbf{2\times2\times3}$	$\mathbf{2\times3\times4}$	$\mathbf{4\times3\times4}$	$\mathbf{4\times6\times4}$
8	$\mathbf{2\times2\times2}$	$\mathbf{4\times2\times2}$	$\mathbf{4\times4\times2}$	$\mathbf{4\times4\times4}$	$\mathbf{4\times8\times4}$
12	$\mathbf{2\times2\times3}$	$\mathbf{4\times2\times3}$	$\mathbf{4\times4\times3}$	$\mathbf{4\times6\times4}$	$\mathbf{6\times8\times4}$
24	$\mathbf{2\times3\times4}$	$\mathbf{4\times3\times4}$	$\mathbf{4\times6\times4}$	$\mathbf{8\times6\times4}$	$\mathbf{8\times6\times8}$

Table 1: Arrangement of clusters for different decompositions

For the purposes of this thesis it is sufficient to keep domains cubical (i.e. sizes of domains in all dimension are identical) and maintain the decomposition into subdomains as regular as possible (i.e. decompose problem in each dimension into the same number of domains).

The final set of values which define the decompostion into clusters and domains is listed in Tab. 1 and 2, respectively.

$$\boldsymbol{nDOF = 3\,(x+1)^3} \tag{103}$$

	Domains			
nprocs	**1152 nDOF**	**576 nDOF**	**384 nDOF**	**288 nDOF**
1	$\mathbf{8 \times 12 \times 12}$	$\mathbf{8 \times 12 \times 6}$	$\mathbf{8 \times 6 \times 8}$	$\mathbf{6 \times 8 \times 6}$
2	$\mathbf{8 \times 12 \times 6}$	$\mathbf{6 \times 8 \times 6}$	$\mathbf{8 \times 6 \times 4}$	$\mathbf{6 \times 4 \times 6}$
3	$\mathbf{8 \times 12 \times 4}$	$\mathbf{8 \times 6 \times 4}$	$\mathbf{4 \times 8 \times 4}$	$\mathbf{4 \times 6 \times 4}$
4	$\mathbf{8 \times 6 \times 6}$	$\mathbf{6 \times 4 \times 6}$	$\mathbf{4 \times 6 \times 4}$	$\mathbf{4 \times 6 \times 3}$
6	$\mathbf{8 \times 6 \times 4}$	$\mathbf{4 \times 6 \times 4}$	$\mathbf{4 \times 4 \times 4}$	$\mathbf{4 \times 3 \times 4}$
8	$\mathbf{4 \times 6 \times 6}$	$\mathbf{4 \times 6 \times 3}$	$\mathbf{4 \times 3 \times 4}$	$\mathbf{3 \times 4 \times 3}$
12	$\mathbf{4 \times 6 \times 4}$	$\mathbf{4 \times 3 \times 4}$	$\mathbf{4 \times 2 \times 4}$	$\mathbf{3 \times 4 \times 2}$
24	$\mathbf{4 \times 3 \times 4}$	$\mathbf{3 \times 4 \times 2}$	$\mathbf{2 \times 4 \times 2}$	$\mathbf{2 \times 3 \times 2}$

Table 2: Arrangement of domains for different decomposition

Number of DOF for 3D linear elasticity problem is computed according to Eq. (102), where $\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{z}$ is number of DOF per dimensions. In our case where cubical domains are used and size in all dimensions is the same the $\boldsymbol{nDOF}$ can be calculated by Eq. (103).

5 Performance models of the FETI solvers

The model estimates (i) the preprocessing time and (ii) the single iteration time. To predict the total time model therefore needs the estimated number of iterations. Since the number of iterations is extremely difficult to estimate for real-world problems, it needs to by considered as another model input.

A simple workflow can be as follows:

1. User guess a number of iterations and uses model to get initial settings from model
2. Run the solver with these settings without preconditioner, with Lumped preconditioner and with Dirichlet preconditioner to get numbers of CG iterations for these three cases
3. Using number of iteration obtained in Step 2 run the model again to get more precise settings

Please note that the model does not take into account the numerical properties of the FETI solver for different decompositions. This is beyond the scope of this master thesis.

Model input:

- Total size of a problem
- FETI method
- Estimated number of CG iterations

The purpose of this model is to predict the computation time for the specific problem. Its size is given by the total number of DOF. The computational methods are Total FETI or Hybrid Total FETI, details are in Sec. 1.2.1 and 1.2.2, respectively.

Model output (for one configuration):

- Estimated optimal decomposition and parallelization settings of the solver
- Anticipated time of computation

Thus the model can be used to compare computational times among different configurations and find the optimal decomposition and preconditioner for the problem of given size.

Details of the model

Model consists of several sub-models described in details in the Sec. 5.1, which describe the most significant (i.e. time-consuming) regions of ESPRESO.

This strategy is advantageous compared to the modeling of the whole ESPRESO solver as one piece, because one can see the behavior of ESPRESO in more details. Moreover, it gives us the advantage if one want to update the model for newer versions of ESPRESO. It will be enough to simply re- model the updated region. This will be much simpler formula, than the "all-consisting" one. Finally, it gives us another advantage, when we are trying to train the model. It is much easier to train several simple models, than one complex model, where the numerical optimization can be almost infeasible.

All the observations (i.e. the run-time measurements) are positive values, so the positivity condition was forced by using generalized linear model with the natural logarithm as the link function (see Sec. 2.1 for description).

Model training was performed by the Fisher scoring algorithm, i.e. the numerical approximation of the Maximum likelihood estimation method. Both methods are described in the the Sec. 2.2.

Every sub-model was evaluated by criteria

- Goodness-of-fit on the full data-set used for the model training
- Significance of single terms in the formula
- Prediction capability

The g.o.f. on the training data were evaluated by comparing null and residual deviances (see Sec. 2.3) and different version of the same model were compared among each other by Akaike information criterion (described in Sec. 2.3).

Every model was simplified until all its terms were significant enough using Wald test (described in Sec. 2.3) with the level of significance $\boldsymbol{\alpha = 0.05}$. Those different versions were compared among each other by AIC, as mentioned above.

And finally, every sub-model is tested by 15 iterations of Random sub- sampling cross-validation using RMSE and sMAPE (both described in Sec. 2.3) as a measure of a prediction capability. The example of R-code performing cross-validation can be seen in Lst. 3.

The full model was then evaluated for the optima-predictive capability by comparison with the measured data (see Tab. 21) and the overall fit using RMSE and sMAPE.

All tools used for the model evaluation as listed above are described in details in the Sec. 2.3.

R syntax

The modeling itself was performed in R language (see [25]), which is a specialized programming language for statistical computations. Because the R-syntax is capable of containing the information about polynomials and term interaction in a relatively compact format this syntax is used to describe the models in this thesis.

The input variables such as the number of nodes, number of MPI processes (i.e. number of clusters) per node, number of domains per cluster and number of DOF per node were denoted as `nnodes`, `nprocs`, `ndoms` and `nDOF`, respectively.

Fitting a model was accomplished with the function `glm()`, which performs the Fisher scoring algorithm (see Sec. 2.2) to fit the generalized linear model, described in Sec. 2.1. The example can be seen in Lst. 1.

```
fit1 <- glm(formula=time ~ (poly(I(1/nprocs), 2) + poly(ndoms, 2) + poly(nDOF, 2))^3,
            data=dataFact,
            family=gaussian(link="log"))
```

Listing 1: Fitting the function using generalized linear regression in R

Formula is described by a special syntax whose brief overview follows. Several examples can be seen in Tab. 3.

$\sim$ denotes "a dependency". If used in GLM, then it describes the response dependent on a predictor - i.e. we can substitute it with $=$ in a mathematical notation.

$\boldsymbol{(x+y)\,\hat{}\,n}$ describes main effects and all interaction up to the n-th order of the listed variables.

$\boldsymbol{I(\cdots)}$ describes the environment with numeric operations, e.g. $\boldsymbol{I(x\hat{}2) = x^2}$, $\boldsymbol{I(log(x)) = \ln x}$ etc.

$\boldsymbol{poly(x,n)}$ describes an orthogonal polynomial of the variable $\boldsymbol{x}$ up to the n-th order.

$\boldsymbol{poly(x,n,raw=T)}$ describes an ordinary polynomial of the variable $\boldsymbol{x}$ up to the n-th order.

An orthogonal polynomial is internally computed as a model matrix of an ordinary polynomial and then its columns are adjusted to be orthogonal to all the previous ones. The first advantage of the orthogonality is, that coefficients remain the same while adding new ones, i.e. $\boldsymbol{\beta_0}$ and $\boldsymbol{\beta_1}$ in the polynomial $\boldsymbol{\beta_0 + \beta_1 x}$ will not change their values after adding $\boldsymbol{\beta_2 x^2}$ etc.

We can show it on a simple example, given by the Lst. 2.

```
m <- lm(dist ~ poly(speed, 1), data = cars, x=T)
m1 <- lm(dist ~ poly(speed, 2), data = cars, x=T)
```

Listing 2: The effect of adding a new term into the orthogonal polynomial in R

The coefficients of the model $\boldsymbol{m}$, obtained by a `summary()` function look like this:

```
                   Estimate
(Intercept)          42.980
poly(speed, 1)      145.552
```

```
                    Estimate
(Intercept)           42.980
poly(speed, 2)1      145.552
poly(speed, 2)2       22.996
```

And we can see, that after adding another term into the polynomial of the ***speed*** variable (see model $\boldsymbol{m1}$ in listing 2), the first two coefficients did not change at all.

The second advantage of the orthogonal polynomials is, that while the previous terms keep their coefficients, the new terms add only new "effects" to the model. So if the new term is not significantly contributing to the model, it can be easily detected with Wald test (described in Sec. 2.3).

When using ordinary polynomials, coefficients of the previous terms change with every new term added to the polynomial. Therefore the contribution of every single new term can not be detected so easily. This could result in all the terms in the ordinary polynomial being detected as non-significant by Wald test, despite the fact, that only its last term does not contribute to the model.

R formula	**Meaning**
`y ~ x + z`	$\boldsymbol{y = \beta_0 + \beta_1 x + \beta_2 z}$
`x + z + x:z`	$\boldsymbol{\beta_0 + \beta_1 x + \beta_2 z + \beta_3 xz}$
`(x + y + z)^3`	$\boldsymbol{\beta_0 + \beta_1 x + \beta_2 y + \beta_3 z + \beta_4 xy + \beta_5 xz + \beta_6 yz + \beta_7 xyz}$
`poly(x, 2, raw=T)`	$\boldsymbol{\beta_0 + \beta_1 x + \beta_2 x^2}$
`I(1/x)`	$\boldsymbol{\frac{1}{x}}$

Table 3: Examples of R formula syntax

The main advantage of the ordinary polynomials over the orthogonal ones is their easy interpretation, as can be seen in the Tab. 3. So, the orthogonal polynomials were mostly used in this thesis during the process of creating models and when the model was well-fitting and simple enough, they were replaced with ordinary ones, to make formulas easy to understand and easy to write down, if needed.

5.1 Partial models

5.1.1 Modeling strategy (Action of K in HTFETI)

This model describes the time-consumption of the action of the $\tilde{\boldsymbol{F}}$-operator, given by the Eq. (31), in the HTFETI method.

The first version of the model will be the Eq. (104), containing polynomial of every predictor variable and all their interactions, so we do not omit anything. The variable ***nprocs*** is described as a fraction $\frac{1}{nprocs}$, because data dependent on it lies on a hyperbola as described in the Sec. 3.

$$\texttt{timeActKH ~ (poly(I(1/nprocs), 2) + poly(ndoms, 2) + poly(nDOF, 2))^3} \quad (104)$$

The summary of the model (see Tab. 4) shows, that only some of the main effects significantly contribute to the model. The proposed model fits much better then the null model, as can be seen from listed deviances (described in Sec. 2.3). Also we can see, that Fisher scoring algorithm, described in Sec. 2.2, found some optima, because there is the finite number of its iterations.

	t value	Pr(>|t|)	
(Intercept)	-85.758	<2e-16	***
poly(I(1/nprocs), 2)1	1.244	0.21546	
poly(I(1/nprocs), 2)2	-0.188	0.85098	
poly(ndoms, 2)1	21.908	<2e-16	***
poly(ndoms, 2)2	-3.227	0.00155	**
poly(nDOF, 2)1	47.715	<2e-16	***
poly(nDOF, 2)2	-19.592	<2e-16	***
poly(I(1/nprocs), 2)1:poly(ndoms, 2)1	0.036	0.97125	
⋮	⋮	⋮	
poly(I(1/nprocs), 2)1:poly(ndoms, 2)1:poly(nDOF, 2)1	-0.045	0.96445	
⋮	⋮	⋮	
Null deviance	9.766522 (167DOF)		
Residual deviance	0.030749 (141DOF)		
AIC	-913.02		
Fisher Scoring it.	6		

Table 4: Action of $\boldsymbol{K}$ - HTFETI (1st model)

So, in the next step we will remove all the interactions and the quadratic term of the ***nprocs*** polynomial, because, as we can see in the summary, it is marked as non-significant too. The linear term of this polynomial has also high p-value, but it is not convenient to remove all the main effects of some predictor variable just because of Wald test. It is because after removing some terms the remaining ones can be already detected as significant ones. And even if the remaining main effects would not be detected as significant by Wald test, the model has usually better predictive capability with the main effects included for all predictor variables.

The new model is described by the Eq. (105).

$$\texttt{time ~ I(1/nprocs) + poly(ndoms, 3) + poly(nDOF, 3)} \quad (105)$$

As we can see in the Tab. 5, all the terms in the model are now detected as significant by Wald test. Akaike information criterion AIC (described in Sec. 2.3) is much lower than previous -913.02, which denotes the significant improvement of the model. Moreover the residual deviance is significantly lower the the null deviance, which means means great improvement over the null model, i.e. the well-fitting model on the data-set.

So, we can change the model to use ordinary polynomials and perform cross-validation, as described in Sec. 2.3 to evaluate the predictive capability of the model. The estimated coefficients are listed in Tab. 5.

	Estimate	Std. Error	t value	Pr(>|t|)	
(Intercept)	-6.693e+00	1.168e-01	-57.284	<2e-16	***
I(1/nprocs)	5.772e-02	1.081e-02	5.340	3.14e-07	***
poly(ndoms, 3, raw = T)1	6.047e-03	6.210e-04	9.738	<2e-16	***
poly(ndoms, 3, raw = T)2	-5.647e-06	1.046e-06	-5.398	2.39e-07	***
poly(ndoms, 3, raw = T)3	2.126e-09	5.123e-10	4.150	5.38e-05	***
poly(nDOF, 3, raw = T)1	7.290e-04	1.524e-05	47.847	<2e-16	***
poly(nDOF, 3, raw = T)2	-5.256e-08	1.759e-09	-29.874	<2e-16	***
poly(nDOF, 3, raw = T)3	1.473e-12	6.193e-14	23.785	<2e-16	***
Null deviance	9.7665223 (167 DOF)				
Residual deviance	0.0056578 (160 DOF)				
AIC	-1235.4				
Fisher Scoring it.	5				

Table 5: Action of $\boldsymbol{K}$ - HTFETI (final model)

As we can see, the coefficients differ dramatically from the previous ones, but all the terms significantly contribute to the model. Predicted values did not change, as we can see from the null deviance, residual deviance and AIC being the same. We can see the visualized fit on the all data-set used for training in Fig. 10. There were performed 15 iterations of repeated random sub-sampling cross-validation (described in Sec. 2.3). The results are listed in the Tab. 6 and visualized in Fig. 11. As we can see, expected values of RMSE and MAPE are pretty low and their standard deviations are low too. Therefore the predictive capability of this model is good and it does not tend to worsen significantly for some training sets.

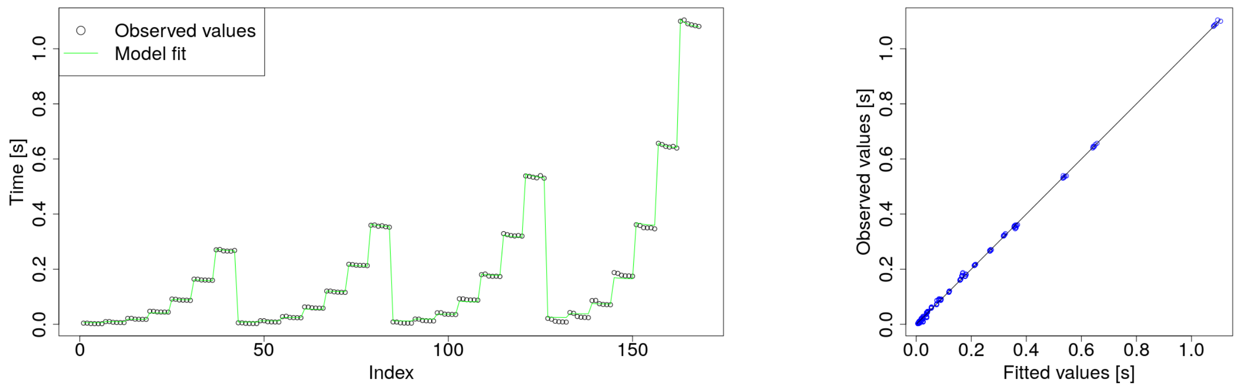
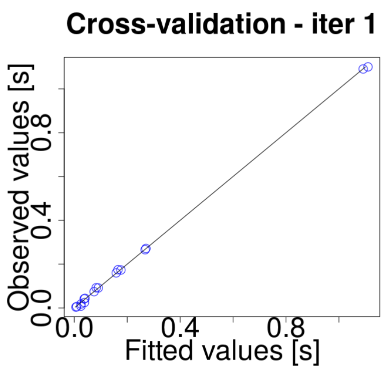
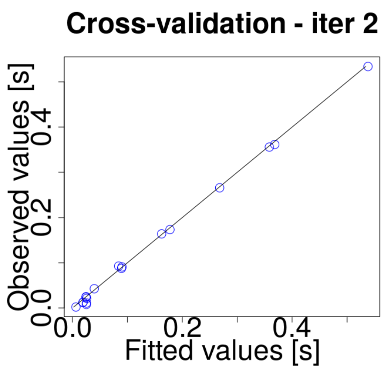


Figure 10: Action of $\boldsymbol{K}$ in HTFETI





Iter	**RMSE**	**MAPE**
1	0.005235	0.141316
2	0.007103	0.138670
3	0.005464	0.220107
4	0.007080	0.184871
5	0.006369	0.220926
6	0.007130	0.234072
7	0.007505	0.222066
8	0.005203	0.179255
9	0.007091	0.277452
10	0.005718	0.165361
11	0.005801	0.160772
12	0.004936	0.154611
13	0.005661	0.142787
14	0.005435	0.102461
15	0.005426	0.091041
	mean(RMSE)	0.006077
	sd(RMSE)	0.000874
	mean(MAPE)	0.175718
	sd(MAPE)	0.051386

Figure 11 & Table 6: Action of $\boldsymbol{K}$ in HTFETI - cross-validation

5.1.2 Action of $\boldsymbol{K}$ in TFETI

This model describes the time-consumption of the action of the stiffness matrix $\boldsymbol{K}$ in the TFETI method, i.e. the Gaussian Elimination applied on the factorized $\boldsymbol{K}$ matrix occuring during the application of the $\boldsymbol{F}$-operator, which is described by the Eq. (13). The model of the Cholesky factorization itself is described in the Sec. 5.1.8 and the Total-FETI method is covered in the Sec. 1.2.1.

The model is described by the equation

```
time ActTT ~ poly(I(1/nprocs), 3, raw=T) + poly(ndoms, 3, raw=T)
           + poly(nDOF, 3, raw=T) + I(1/nprocs):poly(nDOF, 2, raw=T),
```
(106)

its properties can be seen in the Tab. 7 and the fit is visualized in the Fig. 12.

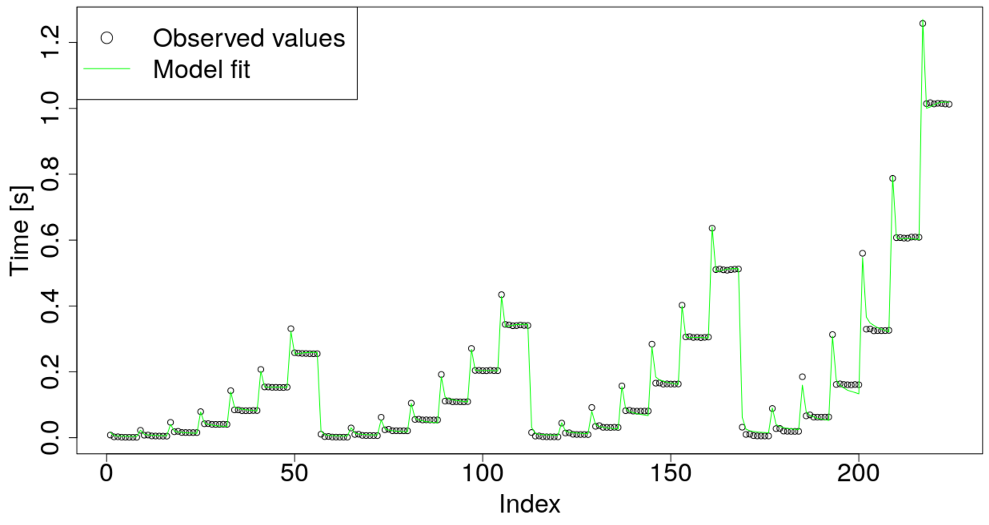


Figure 12: Action of $\boldsymbol{K}$ in TFETI

	Estimate	Std. Error	t value	Pr(>|t|)	
(Intercept)	-7.278e+00	1.359e-01	-53.535	<2e-16	***
poly(I(1/nprocs), 3, raw = T)1	1.416e+00	9.398e-02	15.071	<2e-16	***
poly(I(1/nprocs), 3, raw = T)2	-5.927e-01	2.083e-01	-2.845	0.004869	**
poly(I(1/nprocs), 3, raw = T)3	7.341e-01	1.405e-01	5.226	4.14e-07	***
poly(ndoms, 3, raw = T)1	6.133e-03	7.002e-04	8.758	6.36e-16	***
poly(ndoms, 3, raw = T)2	-5.840e-06	1.180e-06	-4.950	1.51e-06	***
poly(ndoms, 3, raw = T)3	2.217e-09	5.778e-10	3.837	0.000165	***
poly(nDOF, 3, raw = T)1	8.543e-04	1.849e-05	46.199	<2e-16	***
poly(nDOF, 3, raw = T)2	-6.220e-08	1.989e-09	-31.273	<2e-16	***
poly(nDOF, 3, raw = T)3	1.721e-12	6.786e-14	25.359	<2e-16	***
I(1/nprocs):poly(nDOF, 2, raw = T)1	-2.035e-04	1.099e-05	-18.511	<2e-16	***
I(1/nprocs):poly(nDOF, 2, raw = T)2	7.663e-09	5.128e-10	14.943	<2e-16	***
Null deviance	12.390915 (223 DOF)				

Residual deviance	0.012698 (212 DOF)
AIC	-1528.6
Fisher Scoring it.	5
Cross-validation	
mean(RMSE)	0.008685
sd(RMSE)	0.002105
mean(MAPE)	0.218370
sd(MAPE)	0.040581

Table 7: Action of $\boldsymbol{K}$ in TFETI

5.1.3 Preconditioning with Dirichlet p.

This model describes preconditioning with the Dirichler preconditioner in PPCG algorithm described on line 3 of the Alg. 1, or rather its time- consumption. The Dirichlet preconditioner itself is described in the Sec. 1.2.3.

The model is described by the equation

```
timePrecDir ~ I((1/nprocs)^2) + poly(nDOF, 3, raw=T) + poly(ndoms, 3, raw=T),
```
(107)

its properties, like parameter estimates, g.o.f. criteria etc. can be seen in the Tab. 8 and the g.o.f. is visualized in Fig. 13.

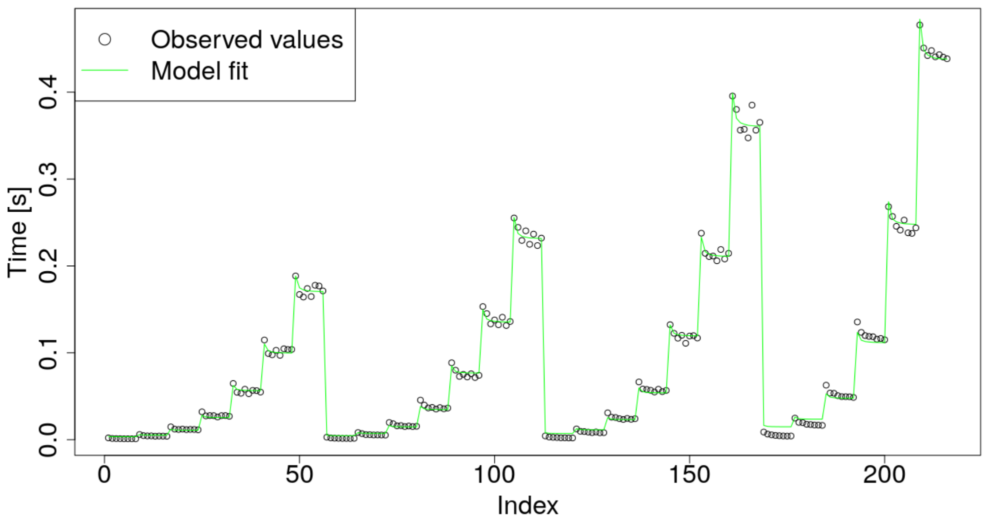


Figure 13: Dirichlet preconditioning

	Estimate	Std. Error	t value	Pr(>|t|)	
(Intercept)	-7.356e+00	1.285e-01	-57.243	<2e-16	***
I((1/nprocs)^2)	9.985e-02	6.565e-03	15.208	<2e-16	***
poly(nDOF, 3, raw = T)1	7.851e-04	1.680e-05	46.746	<2e-16	***
poly(nDOF, 3, raw = T)2	-5.897e-08	1.930e-09	-30.545	<2e-16	***
poly(nDOF, 3, raw = T)3	1.700e-12	6.811e-14	24.956	<2e-16	***
poly(ndoms, 3, raw = T)1	6.223e-03	6.799e-04	9.153	<2e-16	***
poly(ndoms, 3, raw = T)2	-5.513e-06	1.144e-06	-4.820	2.77e-06	***

poly(ndoms, 3, raw = T)3	1.968e-09	5.597e-10	3.516	0.000538	***
Null deviance	2.8865447 (215 DOF)				
Residual deviance	0.0049819 (208 DOF)				
AIC	-1675.3				
Fisher Scoring it.	5				
Cross-validation					
mean(RMSE)	0.004955				
sd(RMSE)	0.000791				
mean(MAPE)	0.216527				
sd(MAPE)	0.040192				

Table 8: Dirichlet preconditioning

5.1.4 Preconditioning with Lumped p.

This model describes the time consumption of the preconditioning with the Lumped preconditioner described by the line 3 in the pseudo-code 1. The preconditioner itself is described in the Sec. 1.2.3.

The model is given by the equation

```
timePrecLum ~ poly(I(1/nprocs), 3, raw=T) + poly(ndoms, 3, raw=T) + poly(nDOF, 3, raw=T)
            + I(1/nprocs):nDOF
```
(108)

Its properties are listed in the Tab. 9 and its g.o.f. is visualized in the Fig. 14.

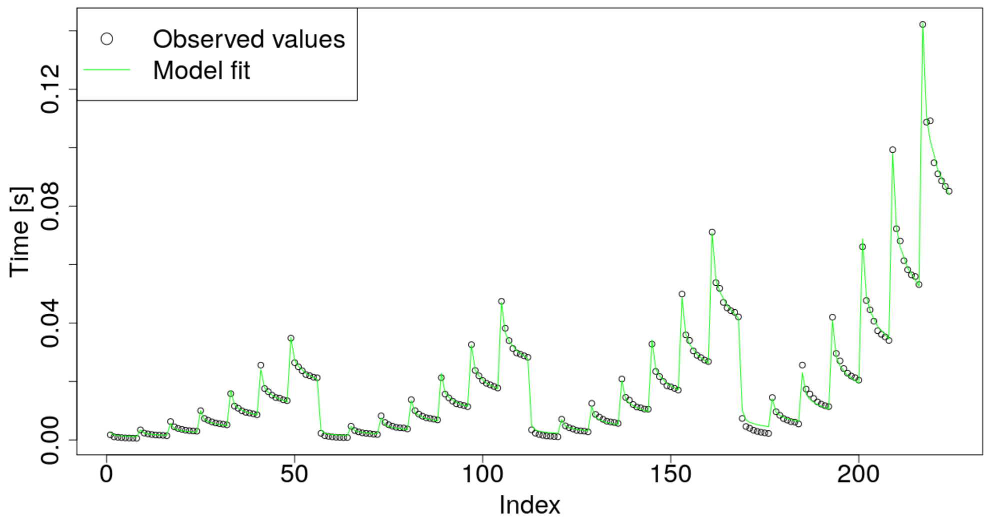


Figure 14: Lumped preconditioning

	Estimate	Std. Error	t value	Pr(>|t|)	
(Intercept)	-8.471e+00	1.667e-01	-50.808	<2e-16	***
poly(I(1/nprocs), 3, raw = T)1	1.235e+00	1.064e-01	11.615	<2e-16	***
poly(I(1/nprocs), 3, raw = T)2	-8.448e-01	2.749e-01	-3.074	0.00239	**
poly(I(1/nprocs), 3, raw = T)3	4.779e-01	1.822e-01	2.623	0.00933	**
poly(ndoms, 3, raw = T)1	6.566e-03	9.241e-04	7.105	1.78e-11	***

poly(ndoms, 3, raw = T)2	-6.465e-06	1.556e-06	-4.155	4.71e-05	***
poly(ndoms, 3, raw = T)3	2.507e-09	7.618e-10	3.291	0.00117	**
poly(nDOF, 3, raw = T)1	5.713e-04	1.291e-05	44.263	<2e-16	***
poly(nDOF, 3, raw = T)2	-4.371e-08	1.608e-09	-27.190	<2e-16	***
poly(nDOF, 3, raw = T)3	1.316e-12	5.959e-14	22.089	<2e-16	***
I(1/nprocs):nDOF	-2.038e-05	1.757e-06	-11.601	<2e-16	***
Null deviance	0.11913698 (223 DOF)				
Residual deviance	0.00022559 (213 DOF)				
AIC	-2433.4				
Fisher Scoring it.	4				
Cross-validation					
mean(RMSE)	0.001120				
sd(RMSE)	0.000368				
mean(MAPE)	0.111473				
sd(MAPE)	0.029503				

Table 9: Lumped preconditioning

5.1.5 Assembling of the Dirichlet preconditioner

This model describes the time consumed by assembling the Dirichlet preconditioner, given by the Eq. (53). The Dirichlet p. is described in detail in the sec. 1.2.3.

The model is given by the equation

```
timeAsmDir ~ poly(I(1/nprocs), 2, raw=T) + poly(nDOF, 4, raw=T) + poly(ndoms, 3, raw=T),   (109)
```

its properties are listed in the Tab. 10 and the g.o.f. is visualized in the Fig. 15.

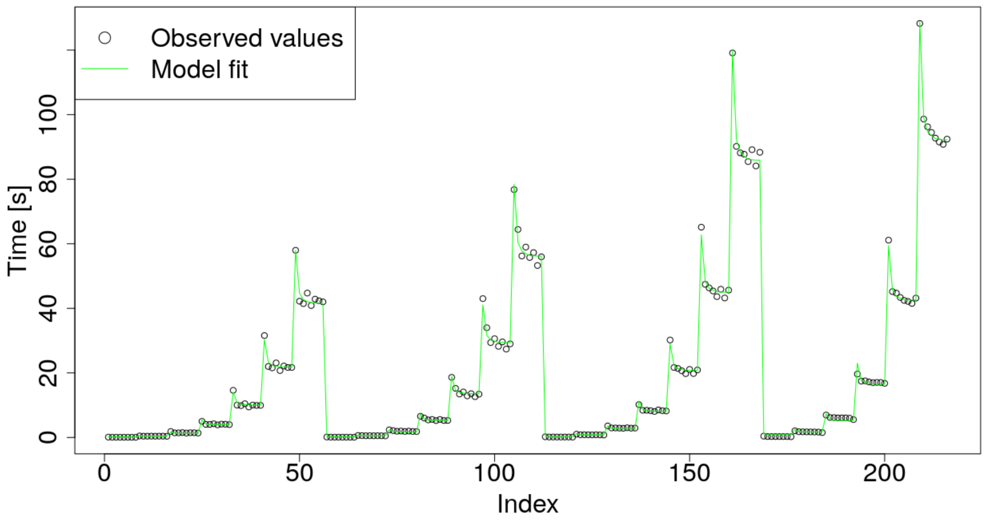


Figure 15: Assemble Dirichlet prec.

	Estimate	Std. Error	t value	Pr(>|t|)	
(Intercept)	-3.546e+00	1.911e-01	-18.557	<2e-16	***

poly(I(1/nprocs), 2, raw = T)1	-4.781e-02	2.399e-02	-1.993	0.0476	*
poly(I(1/nprocs), 2, raw = T)2	3.797e-01	2.144e-02	17.704	<2e-16	***
poly(nDOF, 4, raw = T)1	1.394e-03	1.058e-04	13.173	<2e-16	***
poly(nDOF, 4, raw = T)2	-1.690e-07	2.243e-08	-7.538	1.49e-12	***
poly(nDOF, 4, raw = T)3	1.084e-11	1.914e-12	5.666	4.87e-08	***
poly(nDOF, 4, raw = T)4	-2.686e-16	5.607e-17	-4.791	3.17e-06	***
poly(ndoms, 3, raw = T)1	6.483e-03	5.039e-04	12.866	<2e-16	***
poly(ndoms, 3, raw = T)2	-6.136e-06	8.481e-07	-7.235	9.00e-12	***
poly(ndoms, 3, raw = T)3	2.311e-09	4.152e-10	5.566	8.05e-08	***
Null deviance	162589.78 (215 DOF)				
Residual deviance	160.46 (206 DOF)				
AIC	570.77				
Fisher Scoring it.	6				
Cross-validation					
mean(RMSE)	0.956285				
sd(RMSE)	0.258553				
mean(MAPE)	0.187039				
sd(MAPE)	0.041617				

Table 10: Assembling of the Dirichlet preconditioner

5.1.6 Assembling of the stiffness matrix $\boldsymbol{K}$

This model described the time consumption of the assembling of the distributed global stiffness matrix $\boldsymbol{K}$. The model is given by the equation

```
timeAsmK ~ poly(I(1/nprocs), 2, raw = T) + poly(nDOF, 3, raw = T) + poly(ndoms, 3, raw = T)
         + I(1/nprocs):poly(ndoms, 2, raw = T),
```
(110)

whose properties are listed in the Tab. 11 and its g.o.f. is visualized in the Fig. 16.

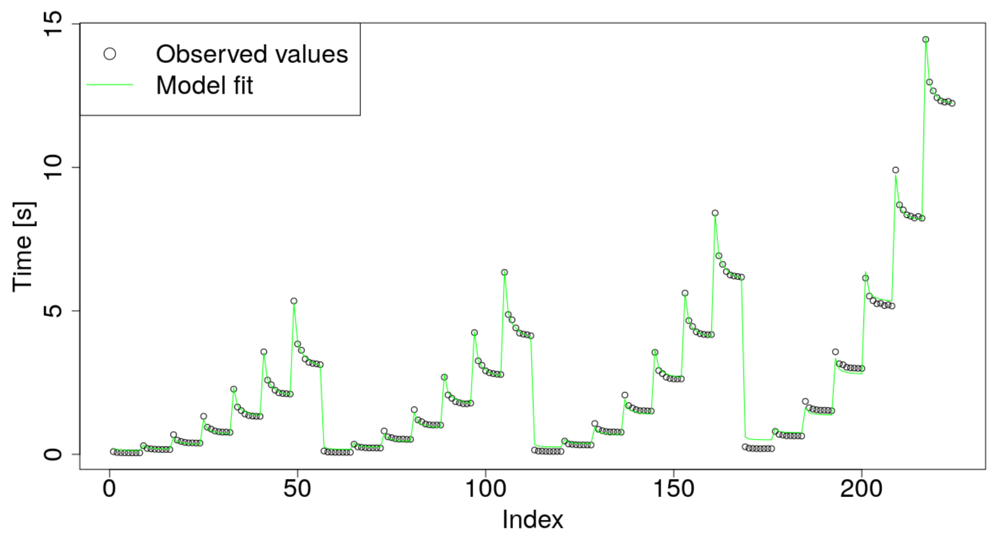


Figure 16: Assemble stiffness matrix $\boldsymbol{K}$

	Estimate	Std. Error	t value	P(>|t|)	
(Intercept)	-3.570e+00	1.339e-01	-26.662	<2e-16	***
poly(I(1/nprocs), 2, raw = T)1	7.868e-01	6.659e-02	11.815	<2e-16	***
poly(I(1/nprocs), 2, raw = T)2	1.199e-01	2.355e-02	5.093	7.75e-07	***
poly(nDOF, 3, raw = T)1	6.790e-04	1.340e-05	50.681	<2e-16	***
poly(nDOF, 3, raw = T)2	-5.341e-08	1.609e-09	-33.197	<2e-16	***
poly(nDOF, 3, raw = T)3	1.570e-12	5.819e-14	26.979	<2e-16 ***	***
poly(ndoms, 3, raw = T)1	5.545e-03	7.238e-04	7.661	6.40e-13	***
poly(ndoms, 3, raw = T)2	-4.723e-06	1.213e-06	-3.893	0.000132	***
poly(ndoms, 3, raw = T)3	1.648e-09	5.936e-10	2.776	0.005985	**
I(1/nprocs):poly(ndoms, 2, raw = T)1	-1.402e-03	1.887e-04	-7.429	2.60e-12	***
I(1/nprocs):poly(ndoms, 2, raw = T)2	6.675e-07	1.198e-07	5.570	7.65e-08	***
Null deviance	1916.0464 (223 DOF)				
Residual deviance	2.6869 (213 DOF)				
AIC	-331.13				
Fisher Scoring it.	6				
Cross-validation					
mean(RMSE)	0.116945				
sd(RMSE)	0.016700				
mean(MAPE)	0.177021				
sd(MAPE)	0.032645				

Table 11: Assembling stiffness matrix $\boldsymbol{K}$

5.1.7 Assembling of the $\boldsymbol{F}_0$ matrix

This model described the time consumed by the assembling of $\boldsymbol{F}_0$ operator matrix in the HTFETI method, described by the Eq. (43). The HTFETI method is described in detail in the sec. 1.2.2.

The model can be described by the equation

```
timeAsmF0 ~ poly(I(1/nprocs), 3) + poly(ndoms, 3) + poly(nDOF, 3),
```
(111)

its properties like g.o.f. and the predictive capability can be seen in the Tab. 12 and the g.o.f. is visualized in the fig.

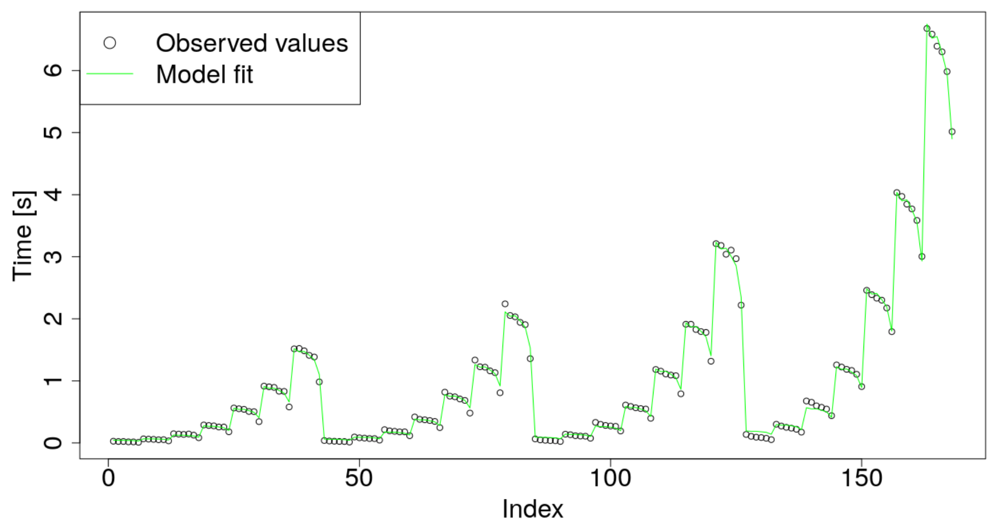


Figure 17: Assembling of $\boldsymbol{F_0}$ in TFETI

	Estimate	Std. Error	t value	Pr(>|t|)	
(Intercept)	-0.80214	0.01443	-55.606	<2e-16	***
poly(I(1/nprocs), 3)1	1.11721	0.03001	37.223	<2e-16	***
poly(I(1/nprocs), 3)2	-0.72635	0.03042	-23.874	<2e-16	***
poly(I(1/nprocs), 3)3	0.39464	0.02976	13.261	<2e-16	***
poly(ndoms, 3)1	7.09411	0.04153	170.825	<2e-16	***
poly(ndoms, 3)2	-1.35300	0.05033	-26.883	<2e-16	***
poly(ndoms, 3)3	0.25416	0.05769	4.406	1.94e-05	***
poly(nDOF, 3)1	15.31205	0.16739	91.474	<2e-16	***
poly(nDOF, 3)2	-4.79144	0.12472	-38.418	<2e-16	***
poly(nDOF, 3)3	1.53099	0.07024	21.795	<2e-16	***
Null deviance	313.08700 (167 DOF)				
Residual deviance	0.38069 (158 DOF)				
AIC	-524.31				
Fisher Scoring it.	5				
Cross-validation					
mean(RMSE)	0.054819				
sd(RMSE)	0.010340				
mean(MAPE)	0.162463				
sd(MAPE)	0.041088				

Table 12: Assembling of the $\boldsymbol{F_0}$ matrix

5.1.8 Factorization of the stiffness matrix $\boldsymbol{K}$

This model describes the Cholesky factorization of the stiffness matrices $\boldsymbol{K}$.

It is given by the equation

```
timeFactK ~ poly(I(1/nprocs), 2, raw = T) + poly(nDOF, 3, raw = T) + poly(ndoms, 3, raw = T)
          + I(1/nprocs):poly(nDOF, 3, raw = T),
```
(112)

its properties are shown in the Tab. 13 and its g.o.f. is visualized in the Fig. 18.

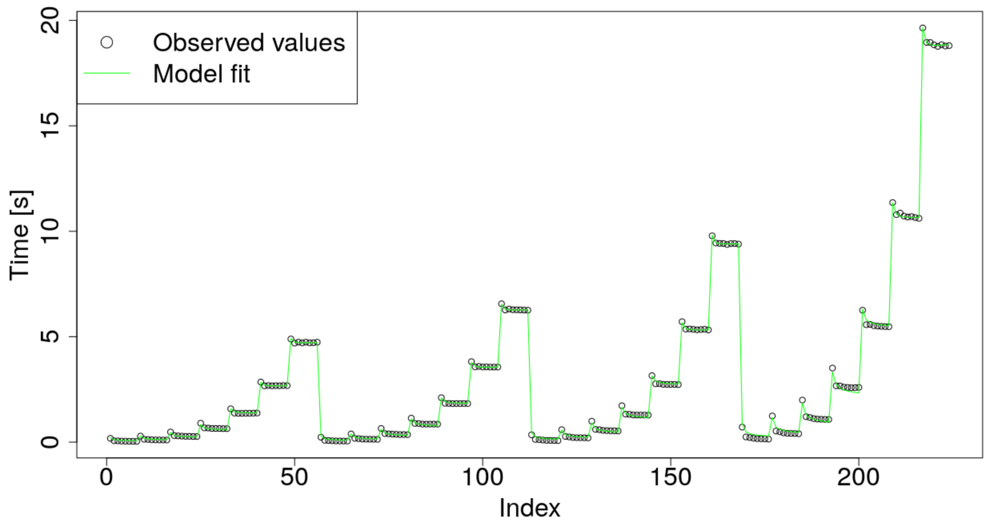


Figure 18: Factorization of $\boldsymbol{K}$

	Estimate	Std. Error	t value	Pr(>|t|)	
(Intercept)	-4.296e+00	7.941e-02	-54.107	<2e-16	***
poly(I(1/nprocs), 2, raw = T)1	1.200e+00	6.946e-02	17.275	<2e-16	***
poly(I(1/nprocs), 2, raw = T)2	6.975e-02	1.158e-02	6.022	7.52e-09	***
poly(nDOF, 3, raw = T)1	8.070e-04	1.568e-05	51.476	<2e-16	***
poly(nDOF, 3, raw = T)2	-5.716e-08	1.791e-09	-31.915	<2e-16	***
poly(nDOF, 3, raw = T)3	1.574e-12	6.257e-14	25.161	<2e-16	***
poly(ndoms, 3, raw = T)1	6.090e-03	3.812e-04	15.976	<2e-16	***
poly(ndoms, 3, raw = T)2	-5.717e-06	6.422e-07	-8.903	2.48e-16	***
poly(ndoms, 3, raw = T)3	2.151e-09	3.145e-10	6.840	8.33e-11	***
poly(nDOF, 3, raw = T)1:I(1/nprocs)	-3.316e-04	2.670e-05	-12.417	<2e-16	***
poly(nDOF, 3, raw = T)2:I(1/nprocs)	3.009e-08	3.173e-09	9.482	<2e-16	***
poly(nDOF, 3, raw = T)3:I(1/nprocs)	-8.989e-13	1.137e-13	-7.902	1.47e-13	***
Null deviance	3891.8051 (223 DOF)				
Residual deviance	1.1254 (212 DOF)				
AIC	-524.07				
Fisher Scoring it.	6				
Cross-validation					
mean(RMSE)	0.076611				
sd(RMSE)	0.014860				

mean(MAPE)	0.142689
sd(MAPE)	0.017324

Table 13: Factorization of the stiffness matrix $\boldsymbol{K}$

5.1.9 Assembling and inverting the $\boldsymbol{GG^T}$ matrix in HTFETI

This model describes the time consumption of the assembling of $\boldsymbol{GG^T}$ and its inversion for the HTFETI method, used in the problem described by the Eq. (37). The assembling itsef is described in [28]. HTFETI method itself is described in the Sec. 1.2.2.

The model is given by the equation

```
timeAsmGGTH ~ poly(I(1/nprocs), 3, raw=T) + poly(nnodes, 3, raw=T) + poly(nDOF, 3, raw=T)
            + poly(ndoms, 3, raw=T) + poly(I(1/nprocs), 2, raw=T):ndoms + ndoms:nDOF        (113)
            + I(1/nprocs^2):nnodes,
```

its properties are then listed in the Tab. 14 nad its g.o.f. is visualized in the Fig. 19.

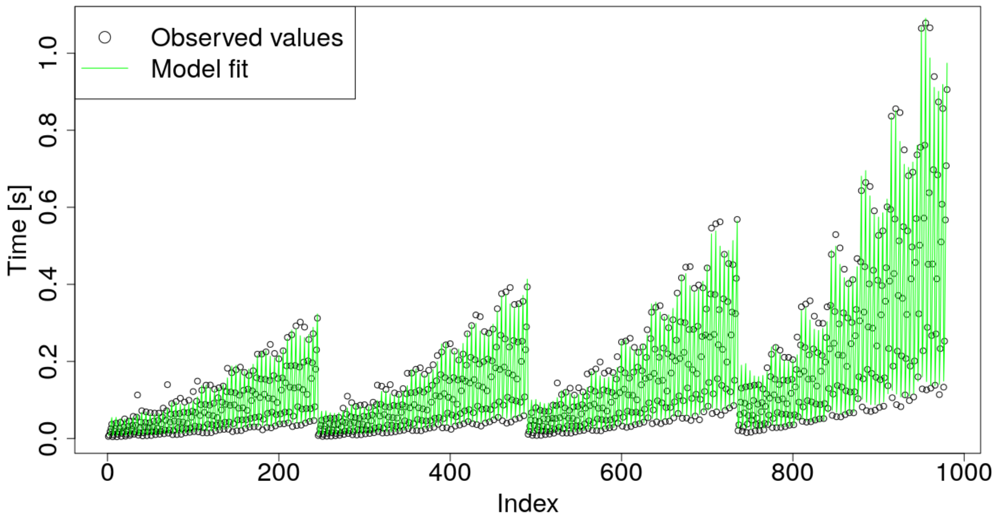


Figure 19: Assembling and inverting the $\boldsymbol{GG^T}$ in HTFETI

	Estimate	Std. Error	t value	Pr(>|t|)	
(Intercept)	-6.531e+00	1.602e-01	-40.768	<2e-16	***
poly(I(1/nprocs), 3, raw = T)1	-5.279e+00	2.676e-01	-19.724	<2e-16	***
poly(I(1/nprocs), 3, raw = T)2	1.849e+01	9.526e-01	19.410	<2e-16	***
poly(I(1/nprocs), 3, raw = T)3	-1.992e+01	1.107e+00	-17.997	<2e-16	***
poly(nnodes, 3, raw = T)1	7.653e-01	1.503e-02	50.917	<2e-16	***
poly(nnodes, 3, raw = T)2	-8.395e-02	2.091e-03	-40.144	2e-16	***
poly(nnodes, 3, raw = T)3	2.883e-03	7.933e-05	36.335	<2e-16	***
poly(nDOF, 3, raw = T)1	3.777e-04	8.102e-06	46.612	<2e-16	***
poly(nDOF, 3, raw = T)2	-3.263e-08	1.105e-09	-29.537	2e-16	***
poly(nDOF, 3, raw = T)3	1.023e-12	4.364e-14	23.433	<2e-16	***
poly(ndoms, 3, raw = T)1	5.966e-03	8.788e-04	6.789	1.97e-11	***

poly(ndoms, 3, raw = T)2	-6.710e-06	1.483e-06	-4.524	6.84e-06	***
poly(ndoms, 3, raw = T)3	2.759e-09	7.275e-10	3.793	0.000158	***
poly(I(1/nprocs), 2, raw = T)1:ndoms	1.801e-03	1.992e-04	9.041	<2e-16	***
poly(I(1/nprocs), 2, raw = T)2:ndoms	-2.250e-03	3.624e-04	-6.208	7.98e-10	***
ndoms:nDOF	6.535e-09	1.672e-09	3.909	9.94e-05	***
I(1/nprocs^2):nnodes	2.584e-02	5.679e-03	4.550	6.06e-06	***
Null deviance	24.88119 (979 DOF)				
Residual deviance	0.26493 (963 DOF)				
AIC	-5234.4				
Fisher Scoring it.	5				
Cross-validation					
mean(RMSE)	0.018052				
sd(RMSE)	0.001778				
mean(MAPE)	0.114571				
sd(MAPE)	0.007239				

Table 14: Assembling $\boldsymbol{GG^T}$ in HTFETI

5.1.10 Assembling and inverting of the $\boldsymbol{GG^T}$ matrix in TFETI

This model describes the time consumption of the assembling of the coarse problem matrix $\boldsymbol{GG^T}$ and its inversion in the TFETI method, used in projector, i.e. in the problem given by the Eq. (19). The assembling of $\boldsymbol{GG^T}$ is described in detail in [28]. TFETI metod itself is described in the Sec. 1.2.1.

There was an enormous NUMA effect (described in the Sec. 3) observed in this region during the measurements, when using 1 process, with 24 threads and 2 processes, each with 12 threads. To decrease the impact of the NUMA effect on the accuracy of the model, we have divided the model into three separate models - for 1 process per node, 2 processes per node and for other number of processes per node.

The model for 1 process with 24 threads per node is described by the equation

```
timeAsmGGTT ~ nnodes + poly(I(log(2*nnodes)), 3, raw=T) + poly(ndoms, 2, raw=T)
            + poly(nDOF, 3, raw=T) + I(log(2*nnodes)^3):poly(nDOF, 2, raw=T),
```
(114)

with properties described in the Tab. 15 and its fit visualized in the Fig. 20.

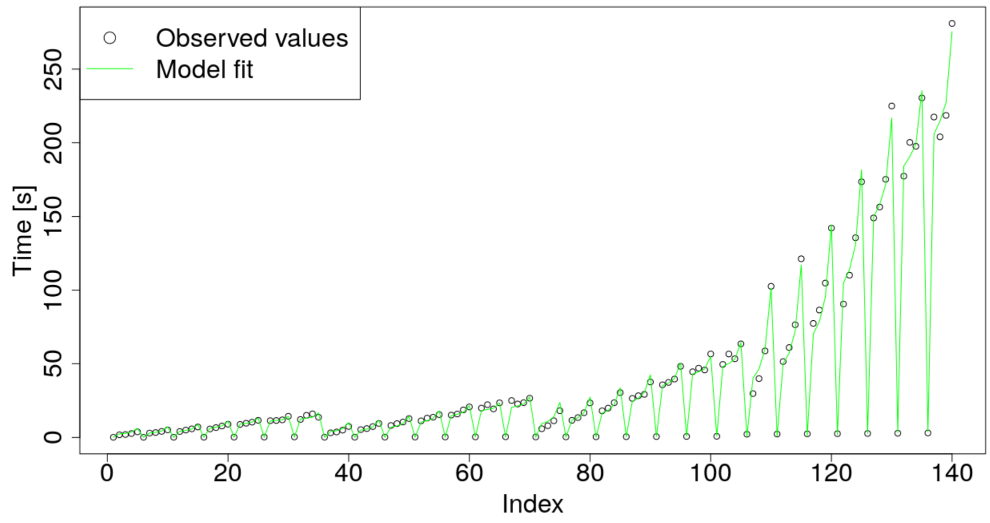


Figure 20: Assembling of the $\boldsymbol{GG^T}$ matrix in TFETI - 1 process per node

	Estimate	**Std. Error**	**t value**	**Pr(>|t|)**	
(Intercept)	-1.696e+01	2.180e+00	-7.776	2.15e-12	***
poly(I(log(2 * nnodes)), 3, raw = T)1	3.218e+01	4.454e+00	7.225	4.00e-11	***
poly(I(log(2 * nnodes)), 3, raw = T)2	-1.636e+01	2.270e+00	-7.206	4.40e-11	***
poly(I(log(2 * nnodes)), 3, raw = T)3	3.925e+00	5.472e-01	7.173	5.24e-11	***
poly(ndoms, 2, raw = T)1	6.866e-03	4.166e-04	16.481	<2e-16	***
poly(ndoms, 2, raw = T)2	-2.514e-06	2.511e-07	-10.011	<2e-16	***
poly(nDOF, 3, raw = T)1	3.876e-04	1.423e-05	27.233	<2e-16	***
poly(nDOF, 3, raw = T)2	-2.965e-08	1.772e-09 -16.734	-16.734	<2e-16	***
poly(nDOF, 3, raw = T)3	7.959e-13	6.820e-14	11.670	<2e-16	***
nnodes	-3.845e+00	5.491e-01	-7.003	1.27e-10	***
I(log(2 * nnodes)^3) :poly(nDOF, 2, raw = T)1	-3.287e-06	2.606e-07	-12.612	<2e-16	***
I(log(2 * nnodes)^3) :poly(nDOF, 2, raw = T)2	1.423e-10	1.453e-11	9.796	<2e-16	***
Null deviance	498544.8 (139 DOF)				
Residual deviance	1632.1 (128 DOF)				
AIC	767.14				
Fisher Scoring it.	4				
Cross-validation					
mean(RMSE)	5.110199				
sd(RMSE)	2.328642				
mean(MAPE)	0.169155				
sd(MAPE)	0.035732				

Table 15: Assembling of the $\boldsymbol{GG^T}$ matrix in TFETI - 1 process per node

The model for 2 processes per node, each with 12 threads, is described by the equation

```
timeAsmGGTT ~ poly(nnodes, 3, raw=T) + poly(ndoms, 2, raw=T) + poly(nDOF, 3, raw=T)
                  + poly(nnodes, 2, raw=T):poly(nDOF, 2, raw=T),
```
(115)

whose properties are listed in the Tab. 16 and its fit can be seen in the figure 21.

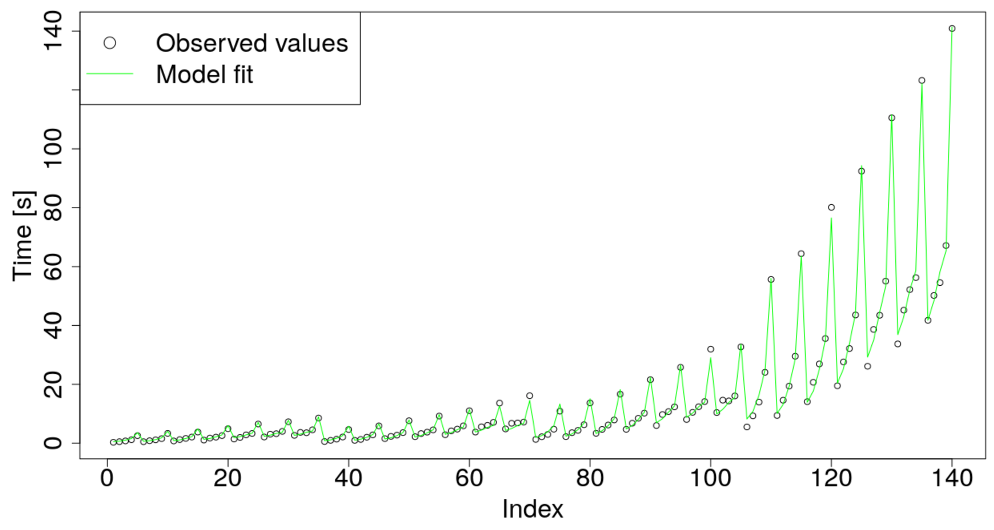


Figure 21: Assembling of the $\boldsymbol{GG^T}$ matrix in TFETI - 2 processes per node

	Estimate	**Std. Error**	**t value**	**Pr(>|t|)**	
(Intercept)	-2.843e+00	1.401e-01	-20.286	<2e-16	***
poly(nnodes, 3, raw = T)1	3.766e-01	2.405e-02	15.662	<2e-16	***
poly(nnodes, 3, raw = T)2	-3.297e-02	2.847e-03	-11.582	<2e-16	***
poly(nnodes, 3, raw = T)3	1.154e-03	1.063e-04	10.857	<2e-16	***
poly(ndoms, 2, raw = T)1	6.668e-03	3.429e-04	19.449	<2e-16	***
poly(ndoms, 2, raw = T)2	-2.416e-06	2.070e-07	-11.672	<2e-16	***
poly(nDOF, 3, raw = T)1	3.742e-04	1.807e-05	20.708	<2e-16	***
poly(nDOF, 3, raw = T)2	-2.632e-08	1.638e-09	-16.069	<2e-16	***
poly(nDOF, 3, raw = T)3	6.578e-13	5.561e-14	11.829	<2e-16	***
poly(nnodes, 2, raw = T)1 :poly(nDOF, 2, raw = T)1	-2.407e-05	4.059e-06	-5.931	<2.68e-08	***
poly(nnodes, 2, raw = T)2 :poly(nDOF, 2, raw = T)1	9.008e-07	2.038e-07	4.420	<2.09e-05	***
poly(nnodes, 2, raw = T)1 :poly(nDOF, 2, raw = T)2	9.117e-10	2.276e-10	4.007	<0.000104	***
poly(nnodes, 2, raw = T)2 :poly(nDOF, 2, raw = T)2	-3.219e-11	1.152e-11	-2.794	<0.006007	**
Null deviance	79149.74 (139 DOF)				
Residual deviance	189.49 (127 DOF)				
AIC	467.68				
Fisher Scoring it.	4				
Cross-validation					
mean(RMSE)	1.562232				
sd(RMSE)	0.426681				
mean(MAPE)	0.107546				
sd(MAPE)	0.029196				

Table 16: Assembling of the $\boldsymbol{GG^T}$ matrix in TFETI - 2 processes per node

And finally, the model for other numbers of processes than 1 or 2 per node is given by the formula

```
timeAsmGGTT ~ poly(nnodes, 3, raw = T) + poly(ndoms, 2, raw = T) + poly(nDOF, 3, raw = T)
            + poly(I(1/nprocs), 3, raw = T) + nnodes:nDOF + nDOF:I(1/nprocs)              (116)
            + nnodes:I(1/nprocs),
```

whose properties are listed in the Tab. 17 and its fit is visualized in the fig.

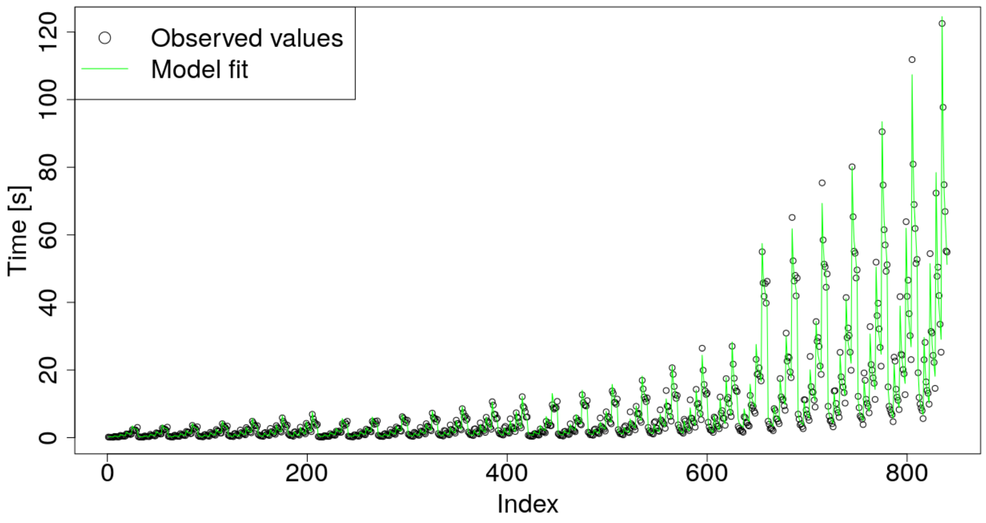


Figure 22: Assembling of the $\boldsymbol{GG^T}$ matrix in TFETI - more than 2 processes per node

	Estimate	Std. Error	t value	Pr(>|t|)	
(Intercept)	-4.110e+00	1.118e-01	-36.744	<2e-16	***
poly(nnodes, 3, raw = T)1	5.939e-01	2.103e-02	28.246	<2e-16	***
poly(nnodes, 3, raw = T)2	-5.291e-02	2.957e-03	-17.897	<2e-16	***
poly(nnodes, 3, raw = T)3	1.749e-03	1.127e-04	15.514	<2e-16	***
poly(ndoms, 2, raw = T)1	6.596e-03	2.772e-04	23.798	<2e-16	***
poly(ndoms, 2, raw = T)2	-2.328e-06	1.673e-07	-13.917	<2e-16	***
poly(nDOF, 3, raw = T)1	1.214e-04	6.600e-06	18.395	<2e-16	***
poly(nDOF, 3, raw = T)2	-7.765e-09	9.794e-10	-7.928	7.21e-15	***
poly(nDOF, 3, raw = T)3	2.194e-13	4.123e-14	5.323	1.32e-07	***
poly(I(1/nprocs), 3, raw = T)1	3.522e+00	4.687e-01	7.513	1.50e-13	***
poly(I(1/nprocs), 3, raw = T)2	-1.231e+01	2.687e+00	-4.581	5.35e-06	***
poly(I(1/nprocs), 3, raw = T)3	2.477e+01	4.591e+00	5.396	8.90e-08	***
nnodes:nDOF	-2.360e-06	1.210e-07	-19.508	<2e-16	***
nDOF:I(1/nprocs)	1.161e-04	5.397e-06	21.511	<2e-16	***
nnodes:I(1/nprocs)	-4.442e-02	6.003e-03	-7.399	3.37e-13	***
Null deviance	192081.2 (839 DOF)				
Residual deviance	1649.3 (825 DOF)				
AIC	2982.6				
Fisher Scoring it.	5				
Cross-validation					
mean(RMSE)	1.530878				

sd(RMSE)	0.256043
mean(MAPE)	0.145268
sd(MAPE)	0.013867

Table 17: Assembling of the $\boldsymbol{GG^T}$ matrix in TFETI - more than 2 processes per node

5.1.11 Action of $\boldsymbol{GG^T}$ in HTFETI

The action of $\boldsymbol{GG^T}$ in HTFETI is the process occuring during the application of the projector described by the Eq. (36). HTFETI itself is described in the Sec. 1.2.2.

The model is given by the equation

```
timeActGGTH ~ I(nDOF^3) + I(ndoms^2) + I(nDOF^2) + (poly(I(1/nprocs), 2, raw = T)
            + poly(nnodes, 2, raw = T))^2 + I(1/nprocs):nDOF + (poly(nnodes, 2, raw = T)   (117)
            + ndoms)^2 + nnodes:nDOF + nDOF:ndoms + nDOF,
```

whose properties are shown in the Tab. 18 and its fit is visualized in the fig. 23.

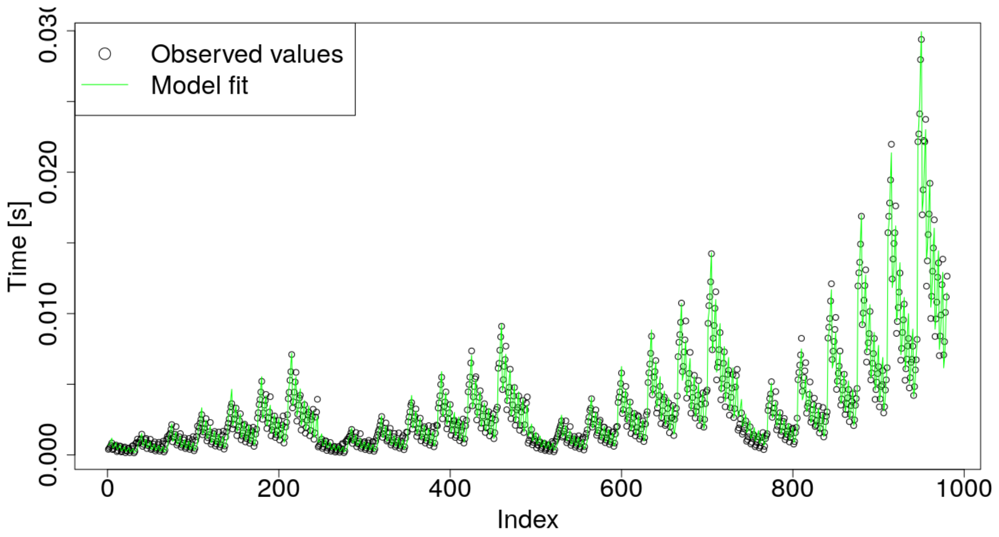


Figure 23: Action of $\boldsymbol{GG^T}$ in HTFETI

	Estimate	Std. Error	t value	Pr(>|t|)	
(Intercept)	-1.012e+01	6.134e-02	-164.953	<2e-16	***
I(nDOF^3)	1.281e-12	5.194e-14	24.668	<2e-16	***
I(ndoms^2)	-1.274e-06	6.123e-08	-20.815	<2e-16	***
I(nDOF^2)	-3.934e-08	1.339e-09	-29.368	<2e-16	***
poly(I(1/nprocs), 2, raw = T)1	5.772e+00	2.404e-01	24.011	<2e-16	***
poly(I(1/nprocs), 2, raw = T)2	-4.688e+00	3.683e-01	-12.729	<2e-16	***
poly(nnodes, 2, raw = T)1	1.769e-01	1.185e-02	14.935	<2e-16	***
poly(nnodes, 2, raw = T)2	-6.391e-03	6.289e-04	-10.161	<2e-16	***
ndoms	3.455e-03	1.023e-04	33.768	<2e-16	***
nDOF	4.435e-04	1.027e-05	43.198	<2e-16	***
poly(I(1/nprocs), 2, raw = T)1 :poly(nnodes, 2, raw = T)1	-4.101e-01	7.045e-02	-5.822	7.94e-09	***

poly(I(1/nprocs), 2, raw = T)2 :poly(nnodes, 2, raw = T)1	4.684e-01	1.113e-01	4.207	2.83e-05	***
poly(I(1/nprocs), 2, raw = T)1 :poly(nnodes, 2, raw = T)2	1.599e-02	3.780e-03	4.230	2.56e-05	***
poly(I(1/nprocs), 2, raw = T)2 :poly(nnodes, 2, raw = T)2	5.996e-03	-1.782e-02	-2.973	0.003027	**
I(1/nprocs):nDOF	-1.761e-05	4.004e-06	-4.398	1.21e-05	***
poly(nnodes, 2, raw = T)1:ndoms	-3.581e-05	7.173e-06	-4.993	7.06e-07	***
poly(nnodes, 2, raw = T)2:ndoms	1.290e-06	3.857e-07	3.345	0.000853	***
nDOF:nnodes	-3.586e-07	1.070e-07	-3.350	0.000840	***
nDOF:ndoms	1.845e-08	2.008e-09	9.187	<2e-16	***
Null deviance	0.01408544 (979 DOF)				
Residual deviance	0.00016512 (961 DOF)				
AIC	-12463				
Fisher Scoring it.	4				
Cross-validation					
mean(RMSE)	0.000416				
sd(RMSE)	0.000042				
mean(MAPE)	0.127067				
sd(MAPE)	0.013751				

Table 18: Action of $\boldsymbol{GG^T}$ in HTFETI

5.1.12 Action of GG^T in TFETI

The action of the coarse problem matrix $\boldsymbol{GG^T}$ is occuring during the application of the projector given by the Eq. (18). The TFETI method itself is described in the Sec. 1.2.1.

The model is given by the formula

```
timeActGGTT ~ poly(nnodes, 3, raw=T) + poly(I(1/nprocs), 3, raw=T) + poly(ndoms, 3, raw=T)
            + poly(nDOF, 2, raw=T) + poly(nnodes, 3, raw=T):nDOF + nnodes:ndoms,
```
(118)

its properties are listed in the Tab. 19 and its fit is visualized in the figure

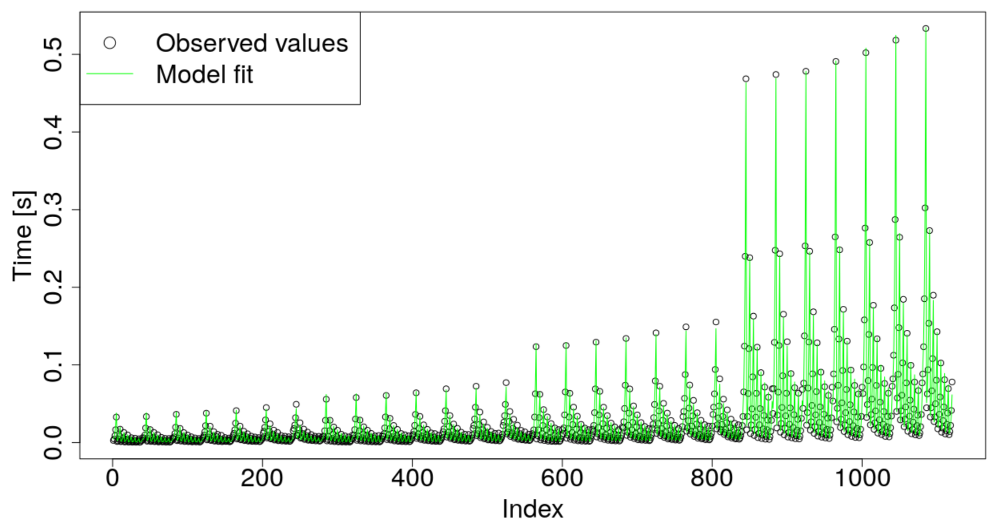


Figure 24: Action of $\boldsymbol{GG^T}$ in TFETI

	Estimate	Std. Error	t value	Pr(>|t|)	
(Intercept)	-1.045e+01	2.082e-01	-50.185	<2e-16	***
poly(nnodes, 3, raw = T)1	5.687e-01	1.642e-02	34.644	<2e-16	***
poly(nnodes, 3, raw = T)2	-5.081e-02	2.301e-03	-22.086	<2e-16	***
poly(nnodes, 3, raw = T)3	1.590e-03	8.749e-05	18.170	<2e-16	***
poly(I(1/nprocs), 3, raw = T)1	5.084e+00	1.090e-01	46.663	<2e-16	***
poly(I(1/nprocs), 3, raw = T)2	-4.007e+00	2.544e-01	-15.753	<2e-16	***
poly(I(1/nprocs), 3, raw = T)3	1.291e+00	1.586e-01	8.143	1.04e-15	***
poly(ndoms, 3, raw = T)1	9.603e-03	1.140e-03	8.423	<2e-16	***
poly(ndoms, 3, raw = T)2	-8.807e-06	1.897e-06	-4.642	3.86e-06	***
poly(ndoms, 3, raw = T)3	3.251e-09	9.222e-10	3.525	0.000441	***
poly(nDOF, 2, raw = T)1	8.568e-05	3.284e-06	26.092	<2e-16	***
poly(nDOF, 2, raw = T)2	-5.968e-10	6.159e-11	-9.689	<2e-16	***
poly(nnodes, 3, raw = T)1:nDOF	-1.821e-05	1.713e-06	-10.628	<2e-16	***
poly(nnodes, 3, raw = T)2:nDOF	1.820e-06	2.450e-07	7.429	2.18e-13	***
poly(nnodes, 3, raw = T)3:nDOF	-5.893e-08	9.394e-09	-6.273	5.07e-10	***
nnodes:ndoms	1.940e-05	1.347e-06	14.395	<2e-16	***
Null deviance	3.3758607 (1119 DOF)				
Residual deviance	0.0072112 (1104 DOF)				
AIC	-10175				
Fisher Scoring it.	5				
Cross-validation					
mean(RMSE)	0.002877				
sd(RMSE)	0.000350				
mean(MAPE)	0.137909				
sd(MAPE)	0.007489				

Table 19: Action of $\boldsymbol{GG^T}$ in TFETI

5.1.13 Assembling of the $\boldsymbol{S}_\alpha$ matrix

The assembling of the Schur complement matrix $\boldsymbol{S}_\alpha$ in the HTFETI method is described by the Eq. (51). The HTFETI method itself is described in the Sec. 1.2.2.

The model of the assembling $\boldsymbol{S}_\alpha$ is described by the equation

```
timeAsmS ~ nDOF + poly(I(1/nprocs), 3, raw=T) + poly(ndoms, 2, raw=T)
          + poly(nDOF, 3):poly(I(1/nprocs), 3),
```
(119)

its properties are listed in the Tab. 20 and its fit is visualized in the Fig. 25.

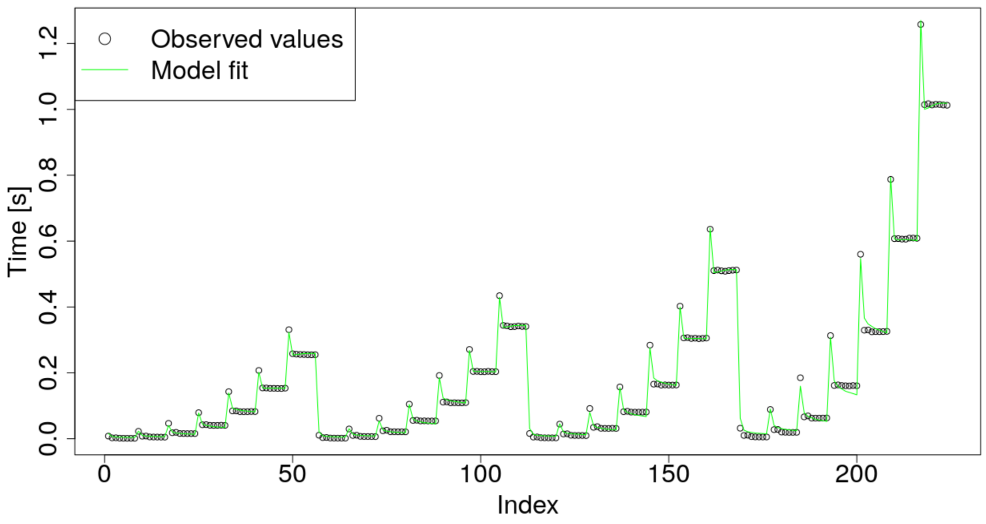


Figure 25: Assembling of the $\boldsymbol{S}_\alpha$ matrix

	Estimate	**Std. Error**	**t value**	**Pr(>|t|)**	
(Intercept)	-9.871e+00	1.826e-01	-54.060	<2e-16	***
poly(I(1/nprocs), 2, raw = T)1	1.550e+01	7.171e-01	21.616	<2e-16	***
poly(I(1/nprocs), 2, raw = T)2	-1.236e+01	9.286e-01	-13.315	<2e-16	***
poly(ndoms, 2, raw = T)1	8.727e-03	3.487e-04	25.025	<2e-16	***
poly(ndoms, 2, raw = T)2	-3.389e-06	2.075e-07	-16.331	<2e-16	***
nDOF	-3.812e-04	3.085e-05	-12.359	<2e-16	***
poly(nDOF, 3, raw = T)1 :poly(I(1/nprocs), 3, raw = T)1	4.930e-03	5.412e-04	9.110	<2e-16	***
poly(nDOF, 3, raw = T)2 :poly(I(1/nprocs), 3, raw = T)1	-3.230e-07	9.645e-08	-3.349	0.000986	***
poly(nDOF, 3, raw = T)3 :poly(I(1/nprocs), 3, raw = T)1	2.042e-11	4.339e-12	4.705	5.02e-06	***
poly(nDOF, 3, raw = T)1 :poly(I(1/nprocs), 3, raw = T)2	-1.757e-02	2.701e-03	-6.506	7.33e-10	***
poly(nDOF, 3, raw = T)2 :poly(I(1/nprocs), 3, raw = T)2	1.646e-06	5.030e-07	3.273	0.001275	**
poly(nDOF, 3, raw = T)3 :poly(I(1/nprocs), 3, raw = T)2	-1.033e-10	2.266e-11	-4.558	9.47e-06	***

poly(nDOF, 3, raw = T)1 :poly(I(1/nprocs), 3, raw = T)3	1.841e-02	3.344e-03	5.506	1.24e-07	***
poly(nDOF, 3, raw = T)2 :poly(I(1/nprocs), 3, raw = T)3	-1.997e-06	6.237e-07	-3.202	0.001615	**
poly(nDOF, 3, raw = T)3 :poly(I(1/nprocs), 3, raw = T)3	1.248e-10	2.810e-11	4.441	1.55e-05	***
Null deviance	14.675879 (195 DOF)				
Residual deviance	0.018029 (181 DOF)				
AIC	-1233.4				
Fisher Scoring it.	8				
Cross-validation					
mean(RMSE)	0.023546				
sd(RMSE)	0.020141				
mean(MAPE)	0.416362				
sd(MAPE)	0.080325				

Table 20: Assembling of the $\boldsymbol{S_\alpha}$ matrix

5.2 Final models and their verification

The final model for the TFETI method without Dirichlet preconditioner (described in the Sec. 1.2.3) is given by the following formula

```
time ~ timeAsmK + timeFactK + timeAsmGGTT
                                                              (120)
     + iterNum*(timeActKT + timeActGGTT + timeActPrec),
```

where `iterNum` is the number of CG iterations (estimated as described in the Sec. 5) and `timeActPrec` is the `timePrecLum` if Lumped preconditioner (described in the Sec. 1.2.3) is used or 0 if there is no preconditioner.

When using Dirichlet preconditioner which requires extra preprocessing time the formula must be modified to

```
time ~ timeAsmK + timeFactK + timeAsmGGTT + timeAsmDir
                                                              (121)
     + iterNum*(timeActKT + timeActGGTT + timePrecDir).
```

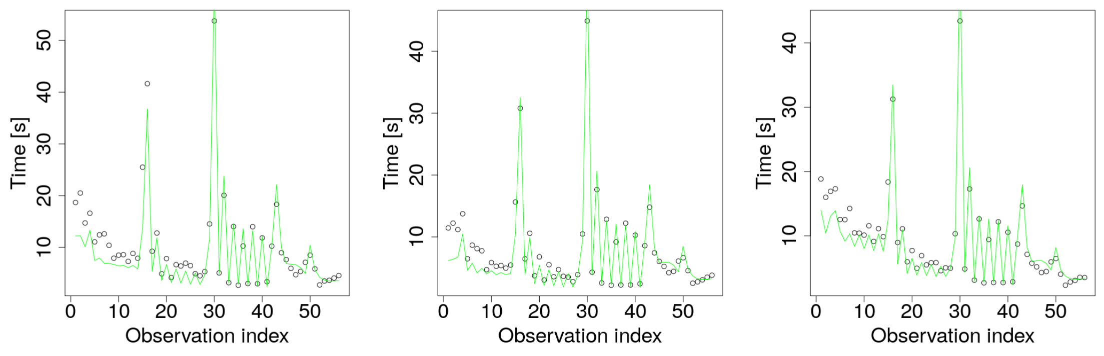


Figure 26: Total FETI model fit with: no preconditioner (left), Lumped precondition (middle) and Dirichlet preconditioner (right).

For HTFETI method both formulas remain very similar, only `timeAsmGGTT`, `timeActGGTT` and `timeActKT` will be replaced with `timeAsmGGTH`, `timeActGGTH` and `timeActKH`. In the HTFETI method there are also matrices $\boldsymbol{F_0}$ and $\boldsymbol{S_\alpha}$ used, who are described in the Sec. 5.1.7 and 5.1.13, respectively.

So, the model for HTFETI without preconditioner or using Lumped preconditioner will be modified as

```
time ~ timeAsmF0 + timeAsmS + timeAsmK + timeFactK + timeAsmGGTH
     + iterNum*(timeActKH + timeActGGTH + timeActPrec)
```
(122)

and the model for HTFETI with Dirichlet preconditioner as

```
time ~ timeAsmF0 + timeAsmS + timeAsmK + timeFactK + timeAsmGGTH
     + iterNum*(timeActKH + timeActGGTH + timeActPrec + timePrecDir)
```
(123)

The results of the final tests are presented in the Tab. 21, the visualized fit of TFETI model is shown in the Fig. 26 and the visualized fit of HTFETI model is in the Fig. 27.

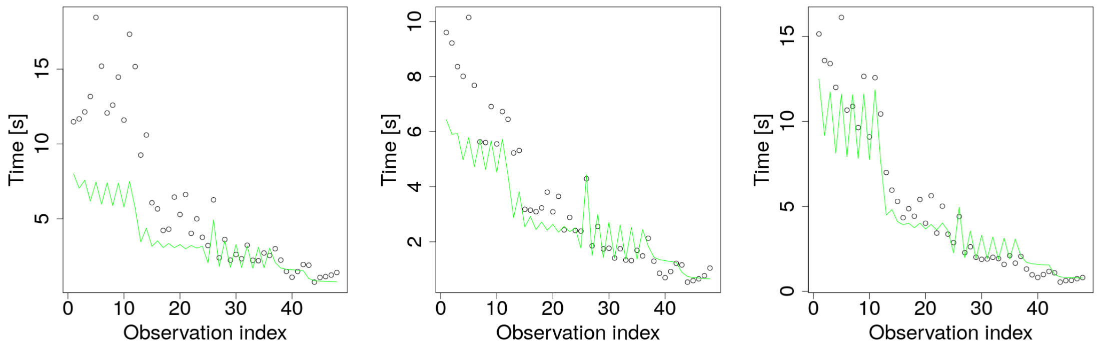


Figure 27: Hybrid Total FETI model fit with: no preconditioner (left), Lumped precondition (middle) and Dirichlet preconditioner (right).

Predicted settings in the Tab. 21 are the settings for which the estimated ESPRESO run-time *Predicted time* is minimal. The predicted time is estimated using the model. *Measured time from predicted settings* contains the measured value corresponding with the predicted settings, i.e. the value we would consider optimal if the model was used for prediction in practice.

Optimal settings are settings, for which the minimal run-time *Optimal time* was measured during tests. *Prediction error* is the difference of measured time from predicted settings and the optimal time.

Finally, RMSE and sMAPE are used to evaluate not only the optima-predictive capability of the model, but the overall model fit. They are both described in the Sec. 2.3.

	1728000 DOFs					
	TFETI			HTFETI		
Preconditioner	**None**	**Lumped**	**Dirichlet**	**None**	**Lumped**	**Dirichlet**
Predicted settings:						
# of compute nodes	4	4	4	16	16	16
# of MPI processes per node	24	24	24	24	24	24
# of Cilk threads per MPI p.	1	1	1	1	1	1
# of domains per MPI rank	18	18	18	12	12	12
domain size [DOF]	1029	1029	1029	375	375	375
Predicted time [s]	2.32	1.8	2.43	0.8	0.66	0.79
Measured time from predicted settings [s]	3.36	2.42	3.72	1.42	1.05	0.81
Optimal settings:						
# of compute nodes	4	4	16	16	16	16
# of MPI processes per node	6	6	4	4	4	4
# of Cilk threads per MPI p.	4	4	6	6	6	6
# of domains per MPI rank	70	70	72	72	72	72
domain size [DOF]	1029	1029	375	375	375	375
Optimal time [s]	2.68	2.19	2.31	0.77	0.53	0.54
RMSE	3.22	2.39	2.51	4.02	1.38	1.54
sMAPE	0.24	0.24	0.18	0.43	0.27	0.24
Prediction error [s]	0.69	0.23	1.41	0.65	0.51	0.27

Table 21: Evaluation of the final model prediction quality

6 Energy consumption evaluation of FETI solver

This thesis is mainly focused on the performance or a run-time model of the ESPRESO FETI solvers. However the optimal energy consumption is similarly important topic in high performance computing with the upcoming exascale computing era. The energy consumption of ESPRESO itself is investigated under the EU Horizon 2020 project called READEX. In this project we have implemented tools MERIC and RADAR used for energy measurements and their evaluation, respectively.

While it is outside of the scope of this thesis, energy measurement results obtained by the two tools mentioned above can be used to create models of optimal energy consumption in a very similar way as the performance model described here.

6.1 MERIC

MERIC is a C++ dynamic library that measures energy consumption and runtime of annotated regions inside a user application. It can also change the tuning parameters during the runtime. By running the code with different settings of the tuning parameters, we analyze possibilities for energy savings. Afterwards the best configurations are applied by changing the tuning parameters during the application runtime. MERIC wraps a list of libraries, that provide access to different hardware knobs and registers, operating system and runtime system variables in order to read or modify their values.

High Definition Energy Efficiency Monitoring (HDEEM) is a library, which enables energy-aware performance optimizations of parallel codes[13]. It provides energy consumption measurement in two different ways, and in MERIC it is possible to choose which one the user wants to use by setting the MERIC CONTINUAL parameter. In one mode, the energy consumed since HDEEM was started is taken from HDEEM Stats structure (a data structure used by the HDEEM library to provide measurements information to the user application). In this mode we read the structure at each region start and end. This solution is straightforward, however there is a delay of approximately 4 ms associated with every read from HDEEM API. To avoid the delay, we take advantage of the fact that during the measurement HDEEM stores energy samples in its internal memory. In this case the MERIC only needs to record timestamps at the beginning and the end of each region instead of calling the HDEEM API. This results in very small overhead of MERIC instrumentation during the application runtime because all samples are transferred from HDEEM memory at the end of the application runtime. The energy is than calculated from the samples based on the recorded timestamps.

Intel Running Average Power Limit (RAPL) is the interface for the energy consumption provided by contemporary Intel processors[5]. MERIC uses the RAPL counters to allow energy measurements on machines without the HDEEM infrastructure as well as to compare them with HDEEM measurements. RAPL counters are read by x86 adapt library.

6.2 RADAR

When the significant regions are annotated with MERIC probes we run the application for all combinations of the selected tuning parameters. Afterwards we analyze the measurement results with RADAR report generator tool. The report generator is a Python based tool which visualizes the MERIC measurements in form of the Latex/PDF document. The goal is to present results in easily readable format compliant with RADAR using aggregated tables, 2D plots and heat-maps. The report generator not only visualizes the measured results, but more importantly it also evaluates the energy consumption using both HDEEM or RAPL, runtime and arithmetical intensity for each significant regions. This analysis detects an optimal configuration of tuning parameters for each significant region and calculates the potential energy savings. The energy savings are calculated for both static and dynamic tuning.

In case of static tuning we evaluate the energy consumption of the entire application and find the single optimal configuration. For the dynamic tuning we evaluate each of the significant regions independently and calculate the additional savings over the static tuning. All the savings are then cumulated to report a single value for static savings and a single value for dynamic savings, as described in [29]. This software was mostly implemented by myself.

6.3 Results

There were performed many measurements using MERIC and RADAR, to evaluate effects of the chosen hardware and application parameters on the energy consumption of ESPRESO (see [29, 14]).

Main hardware parameters were:

- CPU Core frequency
- CPU Uncore frequency
- Number of activeCPU cores (using OpenMP threads)

The application parameter, which was used in the energy consumption evaluation is a preconditioner. In ESPRESO we have performed measurements for both Lumped and Dirichlet preconditioners, described in the Sec. 1.2.3. These results have been submitted as a conference paper. Comparison of various preconditioners from the paper can be seen in the Tab. 22. The paper itself can not be cited now, as it is being peer reviewed now (i.e. April 2017).

Preconditioner	# iterations	1 iteration		Solution	
none	172	125 ms	31.6 J	21.36 s	5 501.31 J
Weight function	100	130+2 ms	32.3+0.53 J	12.89 s	3 284.07 J
Lumped	45	130+10 ms	32.3+3.86 J	6.32 s	1 636.11 J
Light dirichlet	39	130+10 ms	32.3+3.74 J	5.46 s	1 409.82 J
Dirichlet	30	130+80 ms	32.3+20.62 J	6.34 s	1 594.50 J

Table 22: Comparison of preconditioners

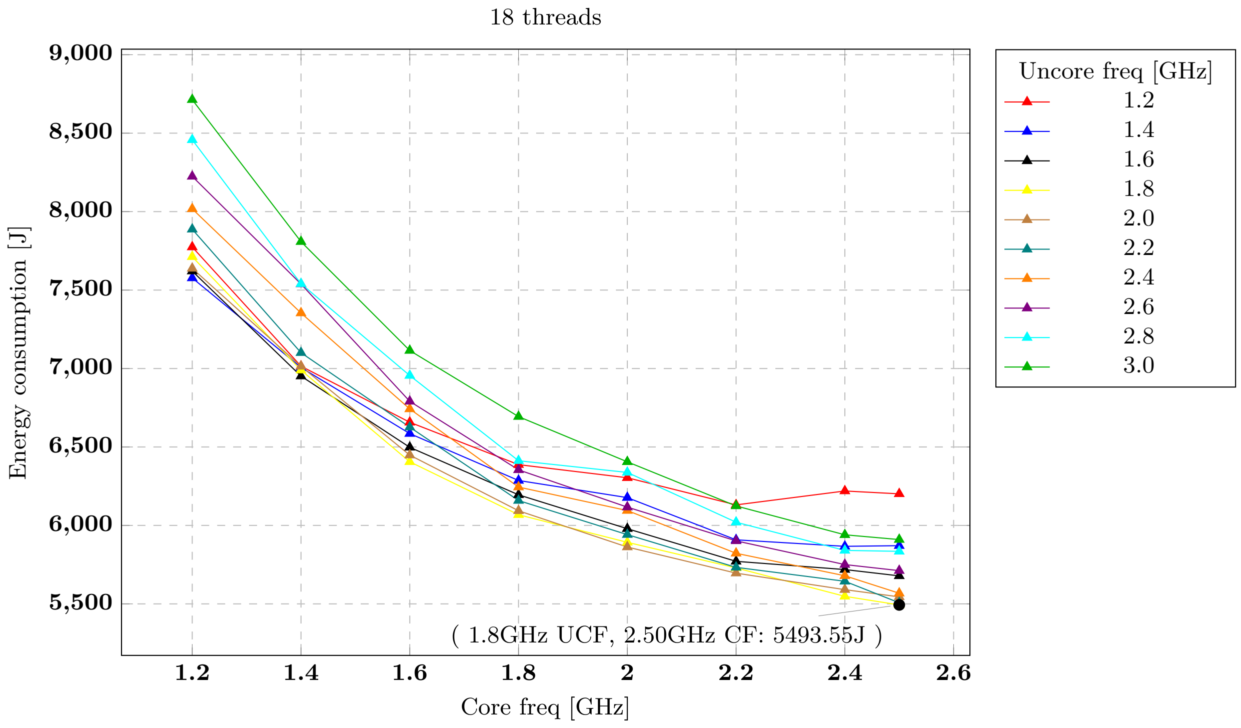
Those reports are out of scope of this thesis, but one example is shown below with following properties:

- Method: Hybrid Total FETI
- Preconditioner: Lumped (sparse)
- Stiffness matrix processing: PARDISO Sparse Direct Solver (sparse)
- Decomposition: 1x1x1 cluster; 8x8x8 subdomains per cluster; 11x11x11 elements per subdomain
- 1 node with 1 MPI process
- 2 to 24 OpenMP threads

Measurement reports are described in detail in [29].

Overall application evaluation

	Default settings	**Default values**	**Best static configuration**	**Static Savings**	**Dynamic Savings**
Energy consumption [J] , Blade summary	24 threads, 3.0 GHz UCF, 2.5 GHz CF	6265.18 J	18 threads, 1.8 GHz UCF, 2.5 GHz CF	771.63 J (12.32%)	499.2 J of 5493.6 J (9.09 %)
Runtime of function [s], Job info - hdeem	24 threads, 3.0 GHz UCF, 2.5 GHz CF	29.55 s	22 threads, 3.0 GHz UCF, 2.5 GHz CF	0.01 s (0.04%)	0.82 s of 29.54 s (2.76 %)



Uncore freq [GHz UCF] / Core freq [GHz]	1.2	1.4	1.6	1.8	2.0	2.2	2.4	2.6	2.8	3.0
1.2	**7,774.33**	**7,577.13**	**7,620.86**	**7,712.41**	**7,638.19**	**7,887.51**	**8,017.52**	**8,224.55**	**8,457.63**	**8,713.34**
1.4	**7,014.96**	**7,006.61**	**6,951.7**	**6,989.9**	**7,013.88**	**7,100.78**	**7,353.77**	**7,538.7**	**7,540.17**	**7,808.45**
1.6	**6,657.43**	**6,585.3**	**6,497.84**	**6,405.66**	**6,448.15**	**6,626.3**	**6,742.37**	**6,790.9**	**6,955.32**	**7,114.6**
1.8	**6,387.41**	**6,286.4**	**6,195.08**	**6,068.22**	**6,093.49**	**6,158.65**	**6,244.49**	**6,354.23**	**6,412.18**	**6,693.56**
2	**6,303.9**	**6,177.23**	**5,979.14**	**5,892.41**	**5,862.35**	**5,941.4**	**6,094.83**	**6,116.72**	**6,337.78**	**6,405.45**
2.2	**6,130.89**	**5,908.28**	**5,771.2**	**5,729.32**	**5,695.97**	**5,732.87**	**5,822.58**	**5,901.66**	**6,020.53**	**6,124.2**
2.4	**6,219.49**	**5,866.82**	**5,718.77**	**5,548.09**	**5,590.74**	**5,644.12**	**5,679.25**	**5,750.53**	**5,840.8**	**5,940.3**
2.5	**6,201.34**	**5,870.99**	**5,678.56**	**5,493.55**	**5,544.76**	**5,507.07**	**5,567.86**	**5,711.89**	**5,834.93**	**5,909.88**

Intra-Phase Dynamism Evaluation
Blade summary, Energy consumption [J]

Region	% of 1 phase	Stat. conf.	Stat. val.	Dyn. conf.	Dyn. val.	Savings
Assembler–AssembleStiffMat	14.32	18 threads, 1.8 GHz UCF, 2.5 GHz CF	733.73 J	20 threads, 2.0 GHz UCF, 2.5 GHz CF	731.22 J	2.51 J (0.34%)
Assembler–Assemble-B1	2.23	18 threads, 1.8 GHz UCF, 2.5 GHz CF	114.30 J	2 threads, 2.2 GHz UCF, 2.5 GHz CF	94.15 J	20.15 J (17.63%)
Cluster–CreateF0-FactF0	0.17	18 threads, 1.8 GHz UCF, 2.5 GHz CF	8.71 J	6 threads, 1.6 GHz UCF, 2.5 GHz CF	6.90 J	1.80 J (20.73%)
Assembler–SaveResults	3.10	18 threads, 1.8 GHz UCF, 2.5 GHz CF	158.81 J	2 threads, 1.2 GHz UCF, 2.5 GHz CF	147.66 J	11.16 J (7.03%)
Assembler-K__-Regularization	5.43	18 threads, 1.8 GHz UCF, 2.5 GHz CF	278.39 J	2 threads, 1.8 GHz UCF, 2.5 GHz CF	231.38 J	47.01 J (16.89%)
Cluster–CreateSa-SolveF0vG0	2.22	18 threads, 1.8 GHz UCF, 2.5 GHz CF	113.87 J	6 threads, 2.0 GHz UCF, 2.5 GHz CF	97.46 J	16.41 J (14.41%)
Create__GGT__-Inv	0.28	18 threads, 1.8 GHz UCF, 2.5 GHz CF	14.23 J	2 threads, 1.2 GHz UCF, 2.5 GHz CF	8.92 J	5.31 J (37.34%)
Cluster–Kfactorization	12.84	18 threads, 1.8 GHz UCF, 2.5 GHz CF	658.07 J	24 threads, 2.0 GHz UCF, 2.4 GHz CF	629.62 J	28.45 J (4.32%)
Assembler–SaveMeshtoVTK	6.36	18 threads, 1.8 GHz UCF, 2.5 GHz CF	325.69 J	2 threads, 1.2 GHz UCF, 2.5 GHz CF	296.66 J	29.03 J (8.91%)
Cluster–CreateSa-SaFactorization	1.95	18 threads, 1.8 GHz UCF, 2.5 GHz CF	99.93 J	4 threads, 2.2 GHz UCF, 2.5 GHz CF	80.85 J	19.08 J (19.09%)
Cluster–SetClusterPC	1.46	18 threads, 1.8 GHz UCF, 2.5 GHz CF	74.70 J	20 threads, 2.0 GHz UCF, 2.5 GHz CF	74.54 J	0.16 J (0.22%)
Assembler–PrepareMesh	12.53	18 threads, 1.8 GHz UCF, 2.5 GHz CF	641.88 J	22 threads, 1.8 GHz UCF, 2.5 GHz CF	639.39 J	2.49 J (0.39%)
Assembler–SolverSolve	30.79	18 threads, 1.8 GHz UCF, 2.5 GHz CF	1578.06 J	10 threads, 2.2 GHz UCF, 2.5 GHz CF	1289.85 J	288.21 J (18.26%)
Assembler–Assemble-B0	0.26	18 threads, 1.8 GHz UCF, 2.5 GHz CF	13.28 J	24 threads, 2.0 GHz UCF, 2.5 GHz CF	12.51 J	0.77 J (5.81%)

Cluster–CreateG1-perCluster	0.47	18 threads, 1.8 GHz UCF, 2.5 GHz CF	24.20 J	14 threads, 2.2 GHz UCF, 2.5 GHz CF	22.32 J	1.88 J (7.76%)
Cluster–CreateF0-AssembleF0	5.43	18 threads, 1.8 GHz UCF, 2.5 GHz CF	278.22 J	24 threads, 2.2 GHz UCF, 2.2 GHz CF	254.98 J	23.24 J (8.35%)
Cluster–CreateSa-SaReg	0.17	18 threads, 1.8 GHz UCF, 2.5 GHz CF	8.59 J	8 threads, 2.0 GHz UCF, 2.5 GHz CF	7.03 J	1.56 J (18.15%)
Total value for static tuning for significant regions			733.73 + 114.30 + 8.71 + 158.81 + 278.39 + 113.87 + 14.23 + 658.07 + 325.69 + 99.93 + 74.70 + 641.88 + 1578.06 + 13.28 + 24.20 + 278.22 + 8.59 = 5124.66 J			
Total savings for dynamic tuning for significant regions			2.51 + 20.15 + 1.80 + 11.16 + 47.01 + 16.41 + 5.31 + 28.45 + 29.03 + 19.08 + 0.16 + 2.49 + 288.21 + 0.77 + 1.88 + 23.24 + 1.56 = 499.22 J of 5124.66 J (9.74 %)			
Dynamic savings for application runtime			499.22 J of 5493.55 J (9.09 %)			
Total value after savings			4994.33 J (79.72 % of 6265.18 J)			

7 Conclusions

The performance model created in this thesis can be used for estimations of the optimal settings in the means of minimization computation time. It fits low values quite well, while there are some problems with fitting larger values, as can be see in Fig. 26 and 27.

The test used for final evaluation of the model was not designed ideally, so the situations occurred, where the prediction was made with non-cubic number of domains and so the measurement had to be performed with different number of domain (see Sec. 4.2), which caused some inaccuracy.

Another effect is a disturbance of measured data, as the measurement was performed only once to test model for "real-life" usage. I expect, that if the test was performed multiple times and the predicted values were compared with the averaged ones from the measurement, prediction errors would be significantly smaller.

Despite those imperfections, considering, that the optimal value is a minimum, this effect worsen the estimation of the optimal configuration by a small margin, so the estimates are quite close to the real optima. It is therefore possible to use the model to get an idea about the behavior of the ESPRESO library for problems arising from linear elasticity and to configure the solver with fairly accurate settings without time consuming manual effort. Please note that an inexperience user can configure the solver in a way that it can run several times slower. In these cases the model is very valuable.

As in the future the model will become part of the ESPRESO library, the calibration and testing methodology will be improved using METIS[19] for problem decomposition. The partial models are going to be also made more accurate, especially the models of assembling of the coarse problem $\boldsymbol{GG^T}$, described in the Sec. 5.1.10 and 5.1.9 for TFETI and HTFETI, respectively. An strategy for model improvement could be in decomposing these complex models into several smaller ones, describing their sub-regions, which would allow better model flexibility.

While the energy consumption model mentioned in the introduction section was not assembled in this thesis, the infrastructure and tools has been already developed and implemented. Both MERIC and RADAR (which was mainly implemented by me) tools developed for energy measurements are already successfully used within the Horizon 2020 project READEX, as described in the Sec. 6. Both tools are described in detail in [29] and the results obtained with their help were published at international conference and journals [16, 15, 14, 31] and [18].

Both the performance model and tools for the energy measurement will be further used by ESPRESO library. It will enable self-calibration of the library to estimate the optimal settings for both optimal run-time and energy requirements automatically. This will remove the need for strenuous testing of different settings, which is now often performed, i.e. not only the time, but also the energy will be saved.

A Source codes

```r
rmse=c()
mape=c()
for (i in 1:15) {
    # 1) Decompose data into 10 groups (e.i. "folds")
    folds <- createFolds(dataFact$time, k=10, list=TRUE, returnTrain=FALSE)
    names(folds)[1] <- "test" # First fold will be the test-set

    # 2) 90% for training a 10% for test
    dataFactTest <- dataFact[ folds$test, ]
    dataFactTrain <- dataFact[ c(folds[[2]],
                                 folds[[3]],
                                 folds[[4]],
                                 folds[[5]],
                                 folds[[6]],
                                 folds[[7]],
                                 folds[[8]],
                                 folds[[9]],
                                 folds[[10]]), ]

    # 3) Training model
    testFit <- glm(formula=formula(currentFit),
                   data=dataFactTrain,
                   family=gaussian(link="log"))

    # 4) Graphical comparion of fitted and observed values
    tmp <- predict(testFit, dataFactTest, se.fit=TRUE, type="response")
    X11()
    plot(tmp$fit, dataFactTest$time, col="blue", main=sprintf("Cross-validation - iter %d", i),
        xlab="Fitted values [s]", ylab="Observed values [s]")
    lines(dataFactTest$time, dataFactTest$time,col="black")

    mapePerIter = sMAPE(dataFactTest$time, tmp$fit)

    printf("%d iter: RMSE=%f, MAPE=%f\n", i, rmse(tmp$fit, dataFactTest$time), mapePerIter)
    rmse[i] = rmse(tmp$fit, dataFactTest$time)
    mape[i] = mapePerIter
}
printf("mean(RMSE)=%f, sd(RMSE)=%f\n", mean(rmse), sd(rmse))
printf("mean(MAPE)=%f, sd(MAPE)=%f\n", mean(mape), sd(mape))
```

Listing 3: Cross-validation R code

References

[1] Hirotugu Akaike. “A new look at the statistical model identification”. In: *IEEE transactions on automatic control* 19.6 (1974), pp. 716–723.

[2] Dimitri P Bertsekas. *Nonlinear programming.*

[3] Tomáš Brzobohatý et al. “Cholesky decomposition with fixing nodes to stable computation of a generalized inverse of the stiffness matrix of a floating structure”. In: *International journal for numerical methods in engineering* 88.5 (2011), pp. 493–509.

[4] David Roxbee Cox and David Victor Hinkley. *Theoretical statistics.* CRC Press, 1979.

[5] Howard David et al. “RAPL: memory power estimation and capping”. In: *Low-Power Electronics and Design (ISLPED), 2010 ACM/IEEE International Symposium on.* IEEE. 2010, pp. 189–194.

[6] Zdeněk Dostál, David Horák, and Radek Kučera. “Total FETI—an easier implementable variant of the FETI method for numerical solution of elliptic PDE”. In: *International Journal for Numerical Methods in Biomedical Engineering* 22.12 (2006), pp. 1155–1162.

[7] Zdeněk Dostál, David Horák, and Dan Stefanica. “A scalable FETI-DP algorithm for a coercive variational inequality”. In: *Applied numerical mathematics* 54.3-4 (2005), pp. 378–390.

[8] Charbel Farhat and Michel Géradin. “On the general solution by a direct method of a large-scale singular system of linear equations: application to the analysis of floating structures”. In: *International Journal for Numerical Methods in Engineering* 41.4 (1998), pp. 675–696.

[9] Charbel Farhat, Jan Mandel, and Francois Xavier Roux. “Optimal convergence properties of the FETI domain decomposition method”. In: *Computer methods in applied mechanics and engineering* 115.3-4 (1994), pp. 365–385.

[10] Charbel Farhat and Francois-Xavier Roux. “An unconventional domain decomposition method for an efficient parallel solution of large-scale finite element systems”. In: *SIAM Journal on Scientific and Statistical Computing* 13.1 (1992), pp. 379–396.

[11] Charbel Farhat et al. “FETI-DP: a dual–primal unified FETI method—part I: A faster alternative to the two-level FETI method”. In: *International journal for numerical methods in engineering* 50.7 (2001), pp. 1523–1544.

[12] Nicholas IM Gould, Jennifer A Scott, and Yifan Hu. “A numerical evaluation of sparse direct solvers for the solution of large sparse symmetric linear systems of equations”. In: *ACM Transactions on Mathematical Software (TOMS)* 33.2 (2007), p. 10.

[13] Daniel Hackenberg et al. “HDEEM: high definition energy efficiency monitoring”. In: *Proceedings of the 2nd International Workshop on Energy Efficient Supercomputing*. IEEE Press. 2014, pp. 1–10.

[14] David Horak et al. “Energy consumption optimization of the Total-FETI solver and BLAS routines by changing the CPU frequency”. In: *High Performance Computing & Simulation (HPCS), 2016 International Conference on*. IEEE. 2016, pp. 1031–1032. DOI: `10.1109/HPCSim.2016.7568453`.

[15] D. Horák et al. “Energy Consumption Optimization of the Total-FETI Solver and BLAS Routines by Changing the CPU Frequency”. In: *International Conference on High Performance Computing & Simulation (HPCS), Innsbruck, Austria*. 2016.

[16] D. Horák et al. “Energy consumption optimization of the Total-FETI solver by changing the CPU frequency”. In: *International Conference on High Performance Computing & Simulation (HPCS), Innsbruck, Austria*. 2016.

[17] M Jarošová et al. “Hybrid Total FETI method”. In: ECCOMAS. 2012.

[18] V. Kannan et al. “Investigating and Exploiting Application Dynamism For Energy-Efficient Exascale Computing”. In: *PRACE-4IP Whitepaper*. 2017.

[19] George Karypis and Vipin Kumar. “METIS–unstructured graph partitioning and sparse matrix ordering system, version 2.0”. In: (1995).

[20] Nicholas Longford. “A fast scoring algorithm for maximum likelihood estimation in unbalanced mixed models with nested random effects”. In: *ETS Research Report Series* 1987.1 (1987).

[21] Daniel Molka et al. “Cache coherence protocol and memory performance of the intel haswell-ep architecture”. In: *Parallel Processing (ICPP), 2015 44th International Conference on*. IEEE. 2015, pp. 739–748.

[22] *MPI_Scan*. URL: `https://www.mpich.org/static/docs/v3.1/www3/MPI_Scan.html` (visited on 04/11/2017).

[23] John A Nelder and R Jacob Baker. “Generalized linear models”. In: *Encyclopedia of statistical sciences* (1972).

[24] Cosmin G. Petra et al. “An augmented incomplete factorization approach for computing the Schur complement in stochastic optimization”. In: *SIAM Journal on Scientific Computing* 36.2 (2014), pp. C139–C162.

[25] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2016. URL: `https://www.R-project.org/`.

[26] Renji Remesan and Jimson Mathew. *Hydrological data driven modelling*. Springer, 2015.

[27] Lubomír Říha, Tomáš Brzobohatý, and Alexandros Markopoulos. “Hybrid parallelization of the total FETI solver”. In: *Advances in Engineering Software* 103 (2017), pp. 29–37.

[28] Lubomír Říha et al. “Massively Parallel Hybrid Total FETI (HTFETI) Solver”. In: *Proceedings of the Platform for Advanced Scientific Computing Conference*. ACM. 2016, p. 7.

[29] Joseph Schuchart et al. “The READEX formalism for automatic tuning for energy efficiency”. In: *Computing* (2017), pp. 1–19.

[30] Radim Sojka. “The hybrid parallelization of TFETI-1 method”. MA thesis. VŠB - Technical University of Ostrava, 2015. URL: `http://am.vsb.cz/export/sites/am/cs/theses/mgr/2015/SOJ0018.pdf` (visited on 04/13/2017).

[31] R. Sojka et al. “The energy consumption optimization of the BLAS routines”. In: *Proceedings from ICNAAM conference, Rhodos, Greece*. 2016.

[32] Chris Tofallis. “A better measure of relative prediction accuracy for model selection and model estimation”. In: *Journal of the Operational Research Society* 66.8 (2015), pp. 1352–1362. ISSN: 1476-9360. DOI: `10.1057/jors.2014.103`. URL: `http://dx.doi.org/10.1057/jors.2014.103`.