Delft University of Technology

Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft Institute of Applied Mathematics

Developing a parallel solver for mechanical problems

A thesis submitted to the
Delft Institute of Applied Mathematics
in partial fulfillment of the requirements

for the degree

MASTER OF SCIENCE
in
APPLIED MATHEMATICS

by

Konrad Borys Kaliszka

Delft, The Netherlands

September 18, 2010

Delft University of Technology

MSc THESIS APPLIED MATHEMATICS

"Developing a parallel solver for mechanical problems"

Konrad Borys Kaliszka

Delft University of Technology

Daily supervisor	**Responsible professor**
Dr. ir. Martin van Gijzen	Prof.dr.ir. Kees Vuik

Committee members

Prof.dr.ir. Kees Vuik	Dr. Ir. Martin van Gijzen
Dr.ir. Hai Xiang Lin	

September 18, 2010 Delft, The Netherlands

Contents

Chapter 1

Description of the problem

1.1 Introduction

Plaxis B.V. is a company specialized in finite element software intended for 2D and 3D analysis of deformation, stability and groundwater flow in geotechnical engineering. Geotechnical applications require advanced constitutive models for the simulation of the non-linear and time-dependent behavior of soils. In addition, since soil is a multi-phase material, special procedures are required to deal with hydrostatic and non-hydrostatic pore pressures in the soil.

PLAXIS

Within the finite element formulation large linear systems have to be solved. At this moment fast and robust iterative solvers are available for sequential computing. Since more and more present day computers consists of more cores, Plaxis is working on parallelization of these solvers. It is not so easy to parallelize the current solver with the same amount of iterations. In this project other techniques are investigated in order to develop robust and efficient parallel solvers. As a first approach domain decomposition methods have to be studied. After a good DD method has been selected the properties of this method is investigated for the problems originating from geotechnical applications. A combination of this method with a second level preconditioner is the next step in this investigation. Finally, instead of an arbitrary data partitioning, the decomposition of the computational domain should be based on the physical properties of the domain. This decomposition can be used to have special preconditioners for certain subdomains and can also be used to develop a good second level preconditioner.

1.2 Acknowledgements

I would like to thank all people who were involved in this project. Special thanks are to due Kees Vuik, Martin van Gijzen, Paul Bonnier, Ronald Brinkgreve, Eric Jan Lingen who help me to finish this Master Thesis and to Dorota Kurowicka, Jolanta Misiewicz and Roger Cooke, without whom I would not be able to come to Delft.

At this moment I would express my gratitude to all the staff at the TU Delft. Also my family and my close friends did a splendid job in helping me to accommodate in the Netherlands.

However the biggest thanks are reserved to my mother, Bozena Kaliszka. Mamo,dziekuje za wszystko.

1.3 Overview

In this Master Thesis, we are going to present the theory which is will be used to solve the problem. We will also present numerical experiments which were done during the work on this master thesis. In the end we will draw conclusions and state any possible future directions of the research.

We will start this Maser Thesis with a short description of the Plaxis software and the systems which are solved by it. After that, we make a short introduction to the Finite Element Method, which is done in the next chapter. After that, we are going to describe the Conjugate Methods, one of ingredients of the approach chosen to be the solution for our problem. This chapter is followed by a description of the Deflation method and it's combination with PCG. After that we present the idea of the Domain Decomposition approach, through presentation of two basic methods from this block of mathematics.

After the theory, we will we will present the results of tests done on the data provided by Plaxis. In the end we will conclude everything with a suggested strategy, which should be applied to an arbitrary problem.

1.4 Short description of problem

As mentioned before, Plaxis B.V. is a company specialized in finite element software intended for 2D and 3D analysis of deformation, stability and groundwater flow in geotechnical engineering. The soil can consist of different layers and each can have a different material model and/or different parameters within it specification. In addition, there can be specified several structural elements, like sheet pile wall or anchors which may be taken into account when simulating. When a problem is created it is subdivided into a large number of finite elements and equilibrium is solved in several steps and iterations. It is also worth to mention, that Each of these nodes has the ability to move in the x,y and z directions. However some nodes are fixed on the boundary and have limited to no freedom of movement. We use the term *degree of freedom* as the unknowns in the linear system and they will correspond to the direction of nodes in which they are able to move. For each direction one row will be introduced in the matrix, so two or three adjacent rows can correspond to the same node.

An important part of the calculation time, especially for larger 3D projects, is solving a linear system of equations. For large systems, direct solution techniques cannot be used so iterative solution techniques are often used.
We will now present some basic equations and theory, which is used within the Plaxis software. More information can be found in [5].

1.4.1 Deformation Theory

The formula for the static equilibrium can be written in the following way:

$$L^T\sigma + b = 0 \tag{1.1}$$

where $\sigma = [\sigma_x \; \sigma_y \; \sigma_z \; \sigma_{xy} \; \sigma_{yz} \; \sigma_{zx}]^T$ is the stress vector, b is the body forces vector and L^T is the transpose of a differential operator defined as:

$$L^T = \begin{bmatrix} \frac{\partial}{\partial x} & 0 & 0 & \frac{\partial}{\partial y} & 0 & \frac{\partial}{\partial z} \\ 0 & \frac{\partial}{\partial y} & 0 & \frac{\partial}{\partial x} & \frac{\partial}{\partial z} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix} \tag{1.2}$$

We will also use the the kinematic relation, which can be formulated as:

$$\epsilon = Lu \tag{1.3}$$

where ϵ is the stress vector. The link between Eq. (1.1) and (1.3) is formed by a constitutive relation representing the material behaviour i.e., the relation between the rates of stress and strain, which can be written as:

$$\dot{\sigma} = M\dot{\epsilon}, \tag{1.4}$$

and where

The combination of Eqs. (1.1), (1.3) and (1.4) would lead to a second-order partial differential equation in the displacements u. However, instead of a direct combination, the equilibrium equation is reformulated in a weak form according to Galerkin's variation principle:

$$\int \delta u^T \left(L^T \sigma + b\right) dV = 0 \tag{1.5}$$

In this formulation δu represents a kinematically admissible variation of displacements. Applying Green's theorem for partial integration to the first term in Eq (1.5), we get:

$$\int \delta \epsilon^T \sigma dV = \int \sigma u^T b dV + \int \delta u^T t dS \tag{1.6}$$

This introduces a boundary integral in which the boundary traction appears. The three components of the boundary traction are assembled in the vector t. Eq. (1.6) is referred to as the virtual work equation.

The development of the stress state σ, can be seen as an incremental process:

$$\begin{aligned} \sigma^i &= \sigma^{i-1} + \Delta\sigma \\ \Delta\sigma &= \int \dot{\sigma} dt \end{aligned} \tag{1.7}$$

In this relation σ^i represents the actual state of stress which is unknown and σ^{i-1} represents the previous state of stress which is known. The stress increment $\Delta\sigma$ is the stress rate integrated over a small time increment.
If Eq. (1.6) is considered for the actual state i, the unknown stresses σ^i can be eliminated using (1.7):

$$\int \delta \epsilon^T \Delta\sigma dV = \int \delta u^T b^i dV + \int \delta u^T t^i dS - \int \delta \epsilon^T \sigma^{i-1} dV \tag{1.8}$$

The Eq. (1.8) is then solved with the use of Finite Element Method, which will be described further in this report and that is why we will omit the derivation associated with it. Nevertheless, as the result we end up with the following equation:

$$\int B \delta dV = \int N b^i dV + \int N^T t^i dS - \int B \sigma^{i-1} dV \tag{1.9}$$

,where B is the strain interpolation matrix, which contains the spatial derivatives and of the interpolation functions, and N is a matrix which stores interpolation functions of the displacement vector, known also as the shape functions.

The above equation is the elaborated equilibrium condition in discretised form. The first term on the right-hand side together with the second term represent the current external force vector and the last term represents the internal reaction vector from the previous step. A difference between the external force vector and the internal reaction vector should be balanced by a stress increment $\Delta\sigma$.

The relation between stress increments and strain increments is usually non-linear. As a result, strain increments can generally not be calculated directly, and global iterative procedures are required to satisfy the equilibrium condition (2.13) for all material points.

1.4.2 Global iterative solution procedure

The formula for the global iterative solution procedure has the following form:

$$K^i \Delta v^i = f_{ex}^i - f_{in}^{i-1} \tag{1.10}$$

where K is the stiffness matrix, Δv is the incremental displacement vector, f_{ex} is the external force vector and f_{in} is the internal reaction vector. The superscript i refers to the step number. However, because the relation between stress increments and strain increments is generally non-linear, the stiffness matrix cannot be formulated exactly beforehand. That is why the global iteration process can be written as:

$$K^j \delta v^j = f_{ex}^i - f_{in}^{j-1} \tag{1.11}$$

where the superscript j refers to the iteration number, δv is a vector containing subincremental displacements, which contribute to the displacement increment of step i:

$$\Delta v^i = \sum_{j=1}^{n} \delta v^j \tag{1.12}$$

where n is the number of iterations within step i. The stiffness matrix K, represents the material behavior in an approximated manner. The more accurate the stiffness matrix is, the fever iterations are required to obtain equilibrium within a certain tolerance.

In a its simplest form, K represents a linear-elastic response. In this case, the stiffness matrix can be formulated as:

$$K = \int B^T D^e B dW \tag{1.13}$$

where D^e is the elastic material matrix according to Hooke's law and B is the strain interpolation matrix.

Chapter 2

Finite Element Method

2.1 Introduction

When dealing with partial differential equations, we need to have a tool which will allow us to solve them or at least to get an approximation of the solution. In this chapter we will present a famous method called Finite Element Method, which let us create an approximation of the PDE we are dealing with, which preserves complex geometries and is quite easy to understand. FEM method is used all over the world in thousands of application.

We will illustrate the Finite Element method with the solution of a Poisson equation with Dirichlet boundary condition, where Ω is a bounded open domain in $\mathbb{R}^2$ and Γ is its boundary.

$$-\Delta u = f \tag{2.1}$$

2.2 Variational Equation

To solve this problem approximately, we will need to extract a system of algebraic equations which will yield the solution. To do that, we will use a common approach, namely the weak formulation of the problem. Denote:

$$\begin{aligned} a(u,u) &= \int_\Omega < \nabla u | \nabla u > dx \\ (f,v) &= \int_\Omega fv \ dx \end{aligned}$$

It is easy to show, that a is bilinear. Now, from Green's formula we get:

$$a(u,v) = -(\Delta u, v) = (\nabla u, \nabla v)$$

Hence, now we can reformulate the problem into following one

$$\text{Find } u \in V \quad \text{such that} \quad \forall v \in V: \ a(u,v) = (f,v) \tag{2.2}$$

where $V \subset L_2$ is the subspace of all functions whose derivatives up to first order are in L_2 and which have zeros on Γ. The resulting space is called $H_0^1(\Omega)$. The above condition is called a Variational Equation.

2.3 Galerkin Equations

Let Ω_h denote an approximation of the domain Ω by the union of the m triangles K_i, which come from the triangulation of Ω. Now, we can replace the space V with a finite dimensional space V_h, which is defined as the space of all functions which are piecewise linear and continuous on the polygonal region Ω_h, and which vanish on the boundary Γ. To be more precise:

$$V_h = \{\phi : \phi_{|\Omega_h} \text{ - continuous, } \phi_{I\Gamma_h} = 0,\ \phi_{|K_j} \text{ - linear for all } j\} \tag{2.3}$$

If x_j, where $j \in \{1, ...,n\}$ are the nodes of the triangulation, then a function ϕ_j in V_h, can be actually associated with each of them, so that it satisfies the following condition:

$$\phi_j(x_i) = \begin{cases} 1, \text{if } x_i = x_j \\ 0, \text{if } x_i \neq x_j \end{cases} . \tag{2.4}$$

The above condition makes $\phi_i, i = 1, ..., n$ defined uniquely. Also the ϕ_i's form a basis of the space V_h, so each function now can be represented as a linear combination of them:

$$\forall u \in V_h : \quad u(x) = \sum_{i=1}^{n} \xi_i \phi_i(x) \tag{2.5}$$

If we now recall the variational equation, and write it for the V_h space, then we will get:

$$\text{Find } u \in V_h \quad \text{such that} \qquad \forall v \in V_h : \ a(u, v) = (f, v) \tag{2.6}$$

by the linearity of a with respect to v, one can impose the condition $a(u, \phi_i) = (f, \phi_i)$, for $i = 1, ..., n$. But from (2.5), we know that u can be represented as the linear combination of the basis function. If we combine those two facts, we will get:

$$\sum_{i=1}^{n} \alpha_{ij} \xi_j = \beta_j, \text{ for all } i = 1, ...n. \tag{2.7}$$

where $\alpha_{ij} = a(\phi_i, \phi_j)$ and $\beta_i = (f, \phi_i)$.

The above equation allows us to formulate a linear system:

$$Ax = b \tag{2.8}$$

with $A = [\alpha_{ij}]_{n \times n}$ and $b = [\beta_1 \ ... \ \beta_n]^T$

The matrices generated by this method have some nice properties. The most important are the facts, that A is Symmetric Positive Definite and sparse. Knowing this, we may now use one of the CG variants for solving these linear systems.

Chapter 3

Conjugate Gradient Method

3.1 Introduction

One of our main problems is to solve a linear system:

$$Ax = b, \tag{3.1}$$

where $A \in \mathbb{R}^{n \times n}$ is called a coefficient matrix, $b \in \mathbb{R}^n$ a right-hand side vector, and where $n \in \mathbb{N}$. Keeping in mind the fact, that we are dealing with matrices from the discretization, we will assume that A is symmetric and semi-positive definite.

There are many ways to solve this problem. In this chapter we will focus on a well-known iterative method, called Conjugate Gradient or, in short-hand notation CG, developed by E. Stiefel and by M.R Hestenes[**3**], which allows us to compute the solution of the above mentioned system of equations. The success of this algorithm lies in it's simplicity. However, to describe the Conjugate Gradient method precisely, we have to know exactly what is a basic iterative method.

3.2 Basic Iterative Methods

Basic iterative solution methods are used to generate a sequence $(x_i, i = 0, 1, ...)$ which may be finite or not, consisting of the approximations of the exact solution x. The compute this sequence, the following recursive formula is used,

$$x_{i+1} = x_i + M^{-1}(b - Ax_i) \tag{3.2}$$

We can substitute $r_i = b - Ax_i$, to which we will from now on refer as the i-th residual, which is used to measure the difference of the i-th approximation and the exact solution and rewrite the above equation once again in a more pleasant way,

$$x_{i+1} = x_i + M^{-1}r_i \tag{3.3}$$

If we now write the first steps of this iteration process,

$$\begin{aligned}
&x_0 = x_0,\\
&x_1 = x_0 + M^{-1}r_0,\\
&x_2 = x_1 + M^{-1}r_1 = x_0 + M^{-1}r_0 + M^{-1}(b - Ax_0 - AM^{-1}r_0)\\
&\ = x_0 + 2M^{-1}r_0 - M^{-1}AM^{-1}r_0\\
&\quad\vdots
\end{aligned} \tag{3.4}$$

we can conclude that

$$x_i \in x_0 + \text{span}\{M^{-1}r_0, M^{-1}A(M^{-1}r_0), ..., (M^{-1}A)^{i-1}(M^{-1}r_0)\} \tag{3.5}$$

The subspace which occurs in the last formula is actually a special case of a Krylov-space, which is defined as $K^i(A, r_0) = \text{span}\{r_0, Ar_0, ..., A^{i-1}r_0\}$. From this we conclude that for each basic iterative method the following is fulfilled

$$x_i \in x_0 + K^i(M^{-1}A; M^{-1}Ar_0) \tag{3.6}$$

These methods are also called Krylov(-subspace) methods. We see that there are two problems which arise from the formula of the basic iterative method. Given the matrix M^{-1}(which also called a "preconditioner") the first problem is to find a suitable basis for $K^i(:, :)$ such that the iterative method has a fast convergence rate with a reasonable accuracy and efficiency with respect to memory storage and computational time. Second, is actually finding the x_i.

3.3 Conjugate Gradient Method

The present section will be devoted to a description of the Conjugate Gradient Method, which nowadays is probably the best known and mostly used iterative method for solving SPD linear systems.

To explain how the CG method works, let us define first what an A-inner product and what an A-norm is.

Definition 3.1. *The A-inner product is defined by*

$$< x|y >_A = x^T Ay$$

Definition 3.2. *The A-semi-norm is defined by*

$$||x||_A = \sqrt{< x|x >_A}$$

Whenever A is Positive Define, we may talk of an A-norm.

The underlying idea of CG is very simple. The sequence $(x_j, j = 0, 1, 2, ...)$ should have the following property:

$$||x - x_j||_A = \min_{y \in K^j(A; r_0)} ||x - y||_A, \text{ for all } j. \tag{3.7}$$

We are sure about the existence of the minimum only if A is SPD. However due to our knowledge about the matrices coming from the discretization of PDE's we do not have to worry about this.

Notice that

$$||x - x_1||_A^2 = (x - \alpha_0 r_0)^T A (x - \alpha_0 r_0) = x^T A x - 2\alpha_0 r_0^T A x + \alpha_0^2 r_0^T A r_0 \tag{3.8}$$

Which can be considered as a parabola of the variable α_0. Hence the minimum is achieved for $\alpha_0 = \frac{r_0^T A x}{r_0^T A r_0} = \frac{r_0^T b}{r_0^T A r_0}$.

In the steepest descent method, each next iteration step is determined by the formula

$$x_{k+1} = x_k + \alpha_k p_k, \tag{3.9}$$

where p_k is the direction of minimum search function for the energy of the system. If we multiply the above equation by A and subtract b from it, we will get

$$Ax_{k+1} - b = Ax_k - b + \alpha_k A p_k$$

But this nothing else but

$$r_{k+1} = r_k - \alpha_k A p_k. \tag{3.10}$$

If we now assume, that p and p_k are conjugate, then from

$$< p | r_{k+1} > = < p, | r_k > + \alpha_k < p, A p_k > \tag{3.11}$$

we see, that if $< p | r_k > = 0$, then also $< p | r_{k+1} > = 0$. This the main condition of the CG method, that for each $j = 0, 1, ..., k$ we have that $< p_j | r_{k+1} > = 0$. If we now define

$$p_0 = -r_0, \; \frac{r_0^T A x}{r_0^T A r_0} = \frac{< r_k | p_k >}{< p_k | p_k >} \text{ and } p_{k+1} = -r_k + \beta_k p_k \text{ for } k = 0, 1, 2, ... \tag{3.12}$$

where β_k induce that p_{k+1} and p_k will be conjugate, then we are done. The only thing which is left to do, is to find β_k. To do that, let us notice that

$$< p_{k+1} | p_k >_A = < -r_{k+1} + \beta_k p_k | p_k >_A = - < r_{k+1} | p_k >_A + \beta_k < p_k | p_k >_A = 0$$

$$\text{when } \beta_k = \frac{< r_{k+1} | p_k >}{< p_k | p_k >_A}.$$

Knowing all those facts, we can write a pseudo-code of the Conjugate Gradient algorithm. However, in most of the literature, for example [**9**], the coefficients α_k and β_k, are computed in a slightly different way. The pseudo-code which is presented, is using the ones which are used more often.

Conjugate Gradient Algorithm

```
Choose $x_0$, set $i = 0, r_0 = b - Ax_0$.
WHILE $r_k \neq 0$ DO
$i := i + 1$
IF $i = 0$ DO
    $p_1 = r_0$
ELSE
    $\beta_i = \frac{r_{i-1}^T r_{i-1}}{r_{i-2}^T r_{i-2}}$
    $p_i = r_{i-1} + \beta_i p_{i-1}$
ENDIF
$\alpha_i = \frac{r_{i-1}^T r_{i-1}}{p_i^T A p_i}$
$x_i = x_{i-1} + \alpha_i p_i$
$r_i = r_{i-1} - \alpha_i A p_i$
END WHILE                                        (3.13)
```

It is very important to notice, that to use CG we need only to remember four vectors and one matrix which makes it attractive in the usage of memory space.

From [**6**], we now that the convergence rate of the CG-method can be easily estimated using the following theorem:

Theorem 3.3. *Let A and x be the coefficient matrix and the solution of (1.1), and let $(x_i, i = 0, 1, 2...)$ be the sequence generated by the CG method. Then, elements of the sequence satisfy the following inequality:*

$$||x - x_i||_A \leq 2 \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^i ||x - x_0||_A, \tag{3.14}$$

where $\kappa(A)$ is the condition number of A in the 2 - norm.

We clearly see, that the convergence depends on the condition number of A, hence we can conclude that the closer $\kappa(A)$ is to 1, the faster we approach the solution of the (3.1). Therefore it is desired to have a matrix with as low as possible condition number. This leads us to a modification of the CG, called Preconditioned Conjugate Gradient Method.

3.4 Preconditioned Conjugate Gradient Method

The idea which helps us to escape the barrier created by Theorem 3.3 and improve the efficiency and robustness of CG is to transform the original linear system (3.1) into one which has the same solution, but is easier to solve with CG.
Let us consider the following problem:

$$A^* x^* = b^*, \tag{3.15}$$

where $A^* = P^{-1}AP^{-T}$, $x^* = P^{-T}x$ and $b^* = P^{-1}b$, where P is a non-singular matrix. The SPD matrix M defined by $M = PP^T$ is called the preconditioner. We can now use the original CG algorithm to solve our new system. The result is the algorithm for the Preconditioned Conjugate Gradient method, or in short-hand notation PCG-method. However, the presented pseudo-code will be rewritten in such a way, that we will only use quantities without the * sign occurs.

Preconditioned Conjugate Gradient Algorithm

```
Choose $x_0$, set $i = 0, r_0 = b - Ax_0$.
WHILE $r_i \neq 0$ DO
$z_i = M^{-1}r_i$
$i := i + 1$
IF $i = 0$ DO
    $p_1 = z_0$
ELSE
    $\beta_i = \frac{r_{i-1}^T z_{i-1}}{r_{i-2}^T z_{i-2}}$
    $p_i = z_{i-1} + \beta_i p_{i-1}$
ENDIF
$\alpha_i = \frac{r_{i-1}^T z_{i-1}}{p_i^T A p_i}$
$x_i = x_{i-1} + \alpha_i p_i$
$r_i = r_{i-1} - \alpha_i A p_i$
END WHILE                                   (3.16)
```

From Theorem (1.1), which determines the convergence rate, we see, that in PCG, $\kappa(P^{-1}AP^{-t})$ will be the coefficient telling us about the speed. That's why the success will be depending on a good choice of the matrix P.

There are two extreme choices, which show the range of PCG. If $P = I$, we will go back to the original CG-method, whereas if we choose $PP^T = A$, we will converge to the solution in one iteration. There are many possibilities of choosing the preconditioner. However, we should keep in mind, that the more complex our preconditioner will be, the more time we will spend on construction and application in the program. That's why, we will present only one easy precoditioner.

For example, if we, take as M, the diagonal of matrix A, we will be dealing with the most standard preconditioner, called Jacobi preconditioner, due to the origins in the Jacobi method. The reason to choose this matrix is the fact, that it is easy to construct, the matrix multiplication is very fast, because of the big number of zero elements. At last, but not least $diag(A^*) = 1$, which results in saving n multiplications in the matrix vector product.

Chapter 4

Deflation

Definition 4.1. *Let A be an SPD coefficient matrix from (1.1). Suppose that $Z \in \mathbb{R}^{n \times k}$, with full rank, and $k \geq n - d$ is given and d is the number of zero eigenvalues of A. Then the deflation matrix $P \in \mathbb{R}^{n \times n}$ is defined as follows:*

$$P := I - AQ \tag{4.1}$$

where:

- *$Q := ZE^{-1}Z^T$ is called the correction matrix.*
- *$E := Z^T AZ$ is called Galerkin matrix.*

Z is often called "deflation-subspace matrix" whose k columns are the "deflation vectors" or "projection vectors". Right now, they do not need to be specified. However, they will be chosen in such a way, that matrix E will be nonsingular[9].

We will now go back to our original linear problem, and solve it using the following decomposition of the solution vector.

$$x = (I - P^T)x + P^T x \tag{4.2}$$

notice, that

$$I - P = I - (I - AQ) = AQ \tag{4.3}$$

$$AP^T = A(I - AQ)^T = A(I - QA) = A - AQA = (I - AQ)A = PA \tag{4.4}$$

$$E^T = (Z^TAZ)^T = Z^TA^TZ = Z^TAZ = E \tag{4.5}$$
$$Q^T = (ZE^{-1}Z^T)^T = ZE^{-T}Z^T = ZE^{-1}Z^T = Q \tag{4.6}$$

Let us now go back to (2.2)

$$\begin{aligned}
x &= (I - P^T)x + P^Tx \\
x &= Qb + P^Tx \\
Ax &= AQb + AP^Tx \\
b &= AQb + PAx \\
(I - AQ)b &= PAx \\
Pb &= PAx
\end{aligned} \tag{4.7}$$

It is crucial to notice, that the solution of (4.7) does not have to be a solution of the original linear system(1.1), because PA is singular. That's why, we will denote the solution of (4.7) as $\bar{x}$ to distinguish from x. We may now formulate a deflated system of our original problem as:

$$PA\bar{x} = Pb, \tag{4.8}$$

and solve it using CG. However we need still to connect the solutions of (3.1) and (4.7), otherwise the whole procedure would not have any reason to exist. The following Lemma **[9]**will provide the needed link:

Lemma 4.2. *Let P be the deflation matrix and Q be the correction matrix of the (1.1) under the assumption that Z satisfies the requirements of Definition (2.1) and b is the right hand-side of (1.1). Suppose that x be the solution of (3.1) and $\bar{x}$ be the solution of (4.8). Then, the following formula holds*

$$x = Qb + P^T\bar{x} \tag{4.9}$$

Proof. Notice that if we decompose $\bar{x}$ as

$$\bar{x} = x + y,$$

where $y \in \mathcal{R}(Z) \subset \mathcal{N}(PA)$, then

$$P^T\bar{x} = P^Tx + P^Ty = P^Tx, \tag{4.10}$$

because $P^ty = \mathcal{O}_n$. This property have arisen from the fact that

$$P^TZ = (I - QA)Z = Z - QAZ = Z - Z = \mathcal{O}_{n\times k} \tag{4.11}$$

Hence, now it is easy to see that:

$$x = (1 - P)^Tx + P^Tx = Qb + P^T\bar{x}.$$

□

It can be shown, that PA is SPSD, hence it can be interpreted as the new coefficient matrix of the linear system (4.8).

4.1 Deflated CG and PCG Methods

We can now write the pseudo-code of the deflated CG method:

Deflated Conjugate Gradient Algorithm

```
Choose $\bar{x}_0$, set $i = 0, \bar{r}_0 = P(b - A\bar{x}_0)$.
**WHILE** $\bar{r}_k \not= 0$ **DO**
$i := i + 1$
**IF** $i = 0$ **DO**
    $p_1 = \bar{r}_0$
**ELSE**
    $\beta_i = \frac{\bar{r}_{i-1}^T \bar{r}_{i-1}}{\bar{r}_{i-2}^T \bar{r}_{i-2}}$
    $p_i = \bar{r}_{i-1} + \beta_i p_{i-1}$
**ENDIF**
$\alpha_i = \frac{\bar{r}_{i-1}^T \bar{r}_{i-1}}{p_i^T PAp_i}$
$\bar{x}_i = \bar{x}_{i-1} + \alpha_i p_i$
$\bar{r}_i = \bar{r}_{i-1} - \alpha_i PAp_i$
**END WHILE**
$x_{orginal} = Qb + P^T \bar{x}_{last}$                                  (4.12)
```

We see that the algorithm is barely touched, the are only little differences between it and the original CG algorithm.
We can also make a preconditioning of the system by using an SPD precondtioner M^{-1}, and then apply onto it Deflated CG method. As the result we get Deflated Preconditioned CG Method, for which present the pseudo-code on the next page.

Deflated Preconditioned Conjugate Gradient Algorithm

```
Choose $\bar{x}_0$, set $i = 0, \bar{r_0} = P(b - A\bar{x}_0)$.
WHILE $\bar{r_k} \neq 0$ DO
i := i + 1
IF $i = 1$ DO
    $y_0 = M^{-1}\bar{r}_0$
    $p_1 = y_0$
ELSE
    $y_{i-1} = M^{-1}\bar{r}_{i-1}$
    $\beta_i = \frac{\bar{r}_{i-1}^T y_{i-1}}{\bar{r}_{i-2}^T y_{i-2}}$
    $p_i = y_{i-1} + \beta_i p_{i-1}$
ENDIF
$\alpha_i = \frac{\bar{r}_{i-1}^T \bar{r}_{i-1}}{p_i^T P A p_i}$
$\bar{x}_i = \bar{x}_{i-1} + \alpha_i p_i$
$\bar{r}_i = \bar{r}_{i-1} - \alpha_i P A p_i$
END WHILE
$x_{orginal} = Qb + P^T \bar{x}_{last}$                                    (4.13)
```

4.2 Deflation Vectors

The choice of the deflation vectors is a very important part of the whole process of deflation methods. In literature [9], we can find several proposition for the candidates to use. The most known strategies for construction of those vectors are:

- Approximated Eigenvector Deflation Vectors
- Recycling Deflation Vectors
- Subdomain Deflation Vectors
- Multigrid and Multilevel Deflation Vectors
- Rigid Body Modes

It is worth to mention, that right know we do not have a universal strategy for constructing deflation vectors, which would give the best results, when applied to every problem.
In this chapter we will restrict ourself to present only two strategies, namely Subdomain Deflation and Rigid Body Modes.

4.2.1 Subdomain Deflation Vectors

In this variant of deflation, we choose the deflation vectors in the following way:
Let $q > 1$ and $j \in \{1 , . . . , q\}$. We divide the computational domain Ω into q subdomains

Ω_j, by the following rules:

$$\bar{\Omega} = \cup_{j=1}^{q} \bar{\Omega}_j \ \wedge \ \forall_{i \neq j} \ \Omega_i \cap \Omega_j = \emptyset \tag{4.14}$$

Let's also denote Ω_h and Ω_{h_j}, for the discretized domain and subdomains respectively. After that we can introduce the deflation vector z_j associated with the j-th subdomain as follows:

$$z_j(i) = \begin{cases} 0, & x_i \in \Omega_h \backslash \Omega_{h_j} \\ 1, & x_i \in \Omega_{h_j} \end{cases} . \tag{4.15}$$

After this step, we define $Z = [z_1 \ z_2 \ \ . \ . \ . \ z_q]$. This finish the construction.

This method is strongly related to approaches known as Domain Decomposition Methods.

4.2.2 Rigid Body Modes

In the recent research in the field of deflation, we can find another approach for choosing the deflation vectors. In [13], we may find an introduction to the Rigid Body Modes used as the engine for the deflation vectors. The main idea is to set for the i-th deflation vector the i-th vector of the null space of A_s, which is a submatrix created from the elements from the FEM discreatisation, which are composing the aggregate subdomains.

Chapter 5

Domain Decomposition Methods

5.1 Introduction

With the rapid growth of high speed computing, we get a powerful tool to our hand. Multi-core processors gives us a possibility to solve very big computation problems in a much faster way than the traditional sequential ones, using the advantages which come from the architecture of the machine used to compute. Among techniques which are based on the parallelization of the computation process, domain decomposition methods are undoubtedly the best known and perhaps the most promising for the problem studied by Plaxis. These methods combine ideas from Partial Differential Equations, linear algebra, mathematical analysis and some part of graph theory. In this chapter we will focus on the decomposition methods, which are based on the general concepts of graph partitioning.

Definition 5.1. *We will call a method a Domain Decomposition method, if its main idea will be based on the principle of divide and conquer applied on the domain of the problem.*

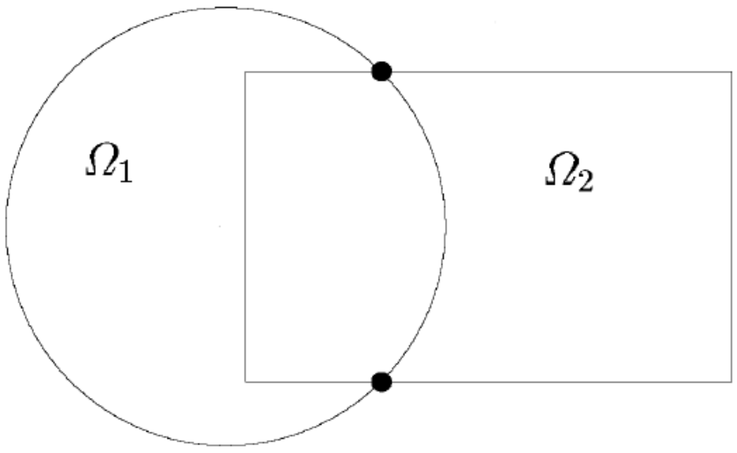

Figure 5.1: An example of domain decomposition

Let us consider the following problem. We want to solve the Laplace Equation on domain Ω partitioned as shown in the figure above. Domain Decomposition methods attempt to solve the problem on the entire domain

$$\Omega = \bigcup_{i=1}^{s} \Omega_i \tag{5.1}$$

from the problem solution on the subdomain Ω_i. There are several reasons why this approach can be advantageous. First of all, the subdomains may have a simpler geometry then Ω. Also sometimes the problem may have a natural split into smaller regions, in which we can have different equations that describe the model. However maybe the most important reason to use Domain Decomposition Methods is the fact, that they are the best choice for the solution of a problem, if we want to parallelize the computational process. Last, but not least, they allow us to deal with the lack of memory, by splitting the domain into parts which will fit into our computers.

There are several methods in the Domain Decomposition family. This report presents only some of them.

5.2 Schwarz Alternating Procedures

The earliest known domain decomposition method is the alternating method of H. Schwarz dating back to 1870. It consisted of three parts: alternating between two overlapping domains, solving the Dirichlet problem on one domain at each iteration and taking boundary conditions based on the most recent solution obtained from the other domain.
Let's consider a domain Ω as shown in Figure 5.1 with two overlapping subdomains Ω_1 and Ω_2 on which we want to solve a PDE of the following form:

$$\begin{cases} Lu = f, & \text{in } \Omega \\ u = g, & \text{on } \partial\Omega \end{cases} . \tag{5.2}$$

Let $\partial\Omega$ denote the boundary of Ω and the artificial boundaries, Γ_i, are the part of the boundary of Ω_i that is interior to Ω, and s is the number of subdomains. Schwarz Alternating Procedure (SAP) for s subdomain problem will be of the following form:

Schwarz Alternating Procedure

```
Choose $u_0$
WHILE no convergence DO
FOR $i = 1, ...s$ DO
    Solve $Lu = f$ in $\Omega_i$ with $u = u_{ij}$ in $\Gamma_{ij}$
    Update u values on $\Gamma_{ij}$, $\forall j$
END FOR
END WHILE                                                    (5.3)
```

In our case, $s = 2$.

In many applications, it is possible to use a matching grid in the overlap region to avoid the duplication of the unknowns on the overlap. The matching version of the alternating method is known as the **multiplicative Schwarz method** (MSM). Writing the linear system for the discretized problem as $Au = f$, we can write the iteration in two fractional steps:

$$\begin{aligned} u^{n+1/2} &= u^n + \begin{bmatrix} A_{\Omega_1}^{-1} & 0 \\ 0 & 0 \end{bmatrix} (f - Au^n) \\ u^{n+1} &= u^{n+1/2} + \begin{bmatrix} 0 & 0 \\ 0 & A_{\Omega_2}^{-1} \end{bmatrix} (f - Au^{n+1/2}) \end{aligned} \tag{5.4}$$

where A_{Ω_i} stays for the discrete form of the operator L restricted to Ω_i.
We can easily see, that the main part of the multiplicative Schwarz method is sequential, so it cannot directly use the benefits of the multi-core architecture and therefore it is not a suitable choice when making a parallel solver.

In literature **[10]** we can find also another approach to SAP, which is more parallel-oriented. This method, called **Additive Schwarz method** (ASM), can be considered as a parallel version of the multiplicative Schwarz method. Its main idea is to change presented previous algorithm by combining the computation of influences to the solution from of each subdomain into one iteration, instead of doing this in each step. For our our example with two domain, the iteration step can be written as:

$$u^{n+1} = u^n + \left(\begin{bmatrix} A_{\Omega_1}^{-1} & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & A_{\Omega_2}^{-1} \end{bmatrix} \right) (f - Au^n) \tag{5.5}$$

If we make a substitution of $B_i = R_i^t A_{\Omega_i}^{-1} R_i$, where R_i is the rectangular restriction matrix that returns the vector of components defined in the interior of Ω_i, then the above equation will be of the following form,

$$u^{n+1} = u^n + (B_1 + B_2)(f - Au^n) \tag{5.6}$$

Having this equation, we can easily generalized ASM for s number of subdomains, by simply adding $B_i(f - Au^n)$ to the right-hand side. As the result of this, for a domain $\Omega = \bigcup_{i=1}^{s} \Omega_i$,we can write the algorithm for Additive Schwarz Method in a following form

Additive Schwarz Method

```
Choose $u_0,\ i = 0$,
**WHILE** no convergence **DO**
$r_i = b - Au^n$
**FOR** $i = 1, ...s$ **DO**
    $\delta_i = B_i r_i$
**END FOR**
$u^{n+1} = u_n + \sum_{i=1}^{s} \delta_i$
$i = i + 1$
**END WHILE**                                        (5.7)
```

5.3 Schur Complement

Let's consider a following problem:

$$\begin{aligned} Lu &= f \text{ in } \Omega \\ u &= g \text{ on } \partial\Omega \end{aligned} \tag{5.8}$$

with the domain Ω partitioned onto s subdomains. After discretization of the problem, we can label the nodes by subdomain in a specific way, so the linear system will have a following structure:

$$\begin{bmatrix} B_1 & & & & E_1 \\ & B_2 & & & E_2 \\ & & \ddots & & \vdots \\ & & & & E_s \\ F_1 & F_2 & \dots & F_s & C \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_s \\ y \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_s \\ g \end{bmatrix} \tag{5.9}$$

where each x_i represents the subvector of unknowns that are interior to subdomain Ω_i, and y represents the vector of all interface unknowns. It is useful to write the system in a more simple form, i.e.

$$A \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} f \\ g \end{bmatrix} \text{,where } A = \begin{bmatrix} B & E \\ F & C \end{bmatrix} \tag{5.10}$$

and where E represents the subbdomain to interface coupling seen from the subdomains, while F represents the interface to subdomain coupling seen from the interface nodes. To illustrate this, let us consider a domain split into only two subdomains. Let's assume that the subdomains are of the same size and both are squared. Then an illustrative mesh and corresponding coefficient matrix A will look like

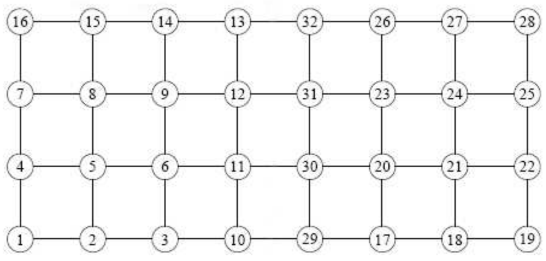

Figure 5.2: An exemplary mesh for the problem described above.

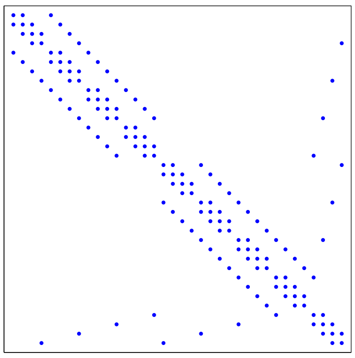

Figure 5.3: Matrix associated with the finite difference mesh of the figure 5.2

Now, we can easily express x with the new terms. From the the first equation it follows that

$$x = B^{-1}(f - Ey) \tag{5.11}$$

If we now substitute this into the second equation, we will obtain a *reduced system*,

$$Sy = g - FB^{-1}f \tag{5.12}$$

where the matrix S is called the *Schur complement* and is created by the following rule:

$$S = C - FB^{-1}E \tag{5.13}$$

If we can form S and solve the associated linear system, then the interface variable y can be obtained. From this we can easily obtain the remaining variable x. Because of the block structure of B, we notice that the solution of the system reduce to solving s separate systems. Because the sets of the variables in each of the system are disjoint, we can solve them simultaneously in parallel. This approach is called *Block Gaussian Elimination* (BGE). The algorithm for it will be now presented:

Block Gaussian Elimination Algorithm

```
SOLVE    $BE' = E$,   $Bf' = f$
COMPUTE  $g = g - Ff'$,   $S = C - FE'$
SOLVE    $Sy = g'$
COMPUTE  $x = f' - E'y$                    (5.14)
```

The partitioning used for the BGE method was edge-based. It means, that a given edge in the graph does not straddle two domains and if any two vertices are coupled, they have to belong to the same subdomain. In the graph theory, this point of view is less common than the vertex-based partitioning, in which a vertex is not shared by two subdomains (except when subdomains overlap).

We will call interface edges all edges which link vertices that are not in the same subdomain. Interface vertices will be those vertices in a given subdomain, that are adjacent to an interface edge. Now due to the fact, that we split the domain according to a new rule, we change the ordering of the nodes. Now the interface nodes are labeled as the last nodes in each subdomain. To illustrate this, let us recall the example used to present edge-based partitioning and apply new rules to it. As the result we will receive the following mesh and coefficient matrix:

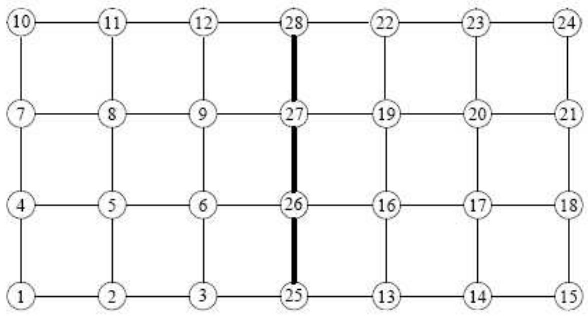

Figure 5.4: An exemplary mesh for the problem described above

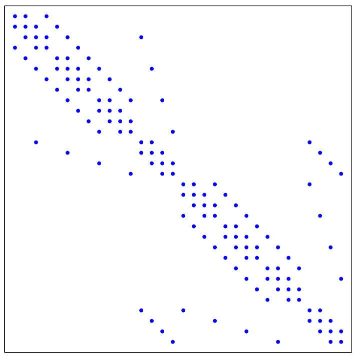

Figure 5.5: Matrix associated with the finite difference mesh of the figure 5.4

Let us consider now the Schur complement system obtained with the new numbering of the nodes. The coefficient matrix A now has a natural s-block structure. For example, if $s = 2$, the matrix will be of the following form:

$$A = \begin{bmatrix} A_1 & A_{12} \\ A_{21} & A_2 \end{bmatrix}. \tag{5.15}$$

Also for each subdomain, the variables will be now have it's own local structure, i.e.,

$$z_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

where x_i now denotes interior nodes, while y_i denotes the interface nodes associated with subdomain i. Now each matrix A_i, will be called local matrix and it will have similar structure to matrix A from (4.9),

$$A_i = \begin{bmatrix} B_1 & E_i \\ F_i & C_i \end{bmatrix}. \tag{5.16}$$

as before, B_i represents the matrix associated with the internal nodes of subdomain i, E_i and F_i represents the susbdomain to interface coupling seen from the subdomains and interface to subdomain coupling seen from the interface nodes respectively and C_i will be the local part of the interface matrix C, which represents the coupling between local interface nodes. Matrices A_{ij} contains zero sub-block in the part that acts on the variable x_j, therefore we can write that

$$A_{ij} = \begin{bmatrix} 0 \\ C_{ij} \end{bmatrix}$$

It is worth to mention, that most of the C_{ij} matrices are zero, since only those indices j of the subdomains that have coupling with subdomain i will yield a nonzero C_{ij}.

If we now write the part of linear system that is local to subdomain i, as

$$\begin{array}{lcl} B_i x_i + E_i y_i & = & f_i \\ F_i x_i + C_i y_i + \sum_{j \in N_i} C_{ij} y_j & = & g_i \end{array}. \tag{5.17}$$

The term $C_{ij} y_j$ is the influence coming from the neighboring subdomain with number j. N_j is a set of indexes of the subdomains which are adjacent to subdomain j.
If we assume that B_i are nonsingular, then we can apply the similar solution technique, which we used to develop the BGE. As the result of this, we receive a system of reduced systems

$$S_i y_i + \sum_{j \in N_i} E_i j y_j = g_i - F_i B_i^{-1} E_i \tag{5.18}$$

where S_i is the "local" Schur complement, and is defined as

$$S_i = C_i - F_i B_i^{-1} E_i. \tag{5.19}$$

5.4 Numerical Illustration

The presented numerical experiments in this section illustrate how does the theory from this paragraph acts in practice. Both presented methods, namely Schur Complement Method and Schwarz have been tested on the standard test equation, i.e. on the Poisson equation

$$-\Delta u = f, \tag{5.20}$$

on the domain Ω, which was chosen to be of a rectangular shape, and with Dirichlet boundary condition. However due to the fact, that we are going to work with the Schwarz method in the future, we restrict ourself only to present results only for this method.

The domain Ω after discretization has a $n \times 2n + 1$ size. We decided to split it into two subdomains Ω_1 and Ω_2 along the vertical middle, with a overlap at it. Below we can see a graphical illustration of this process, when $n = 4$.

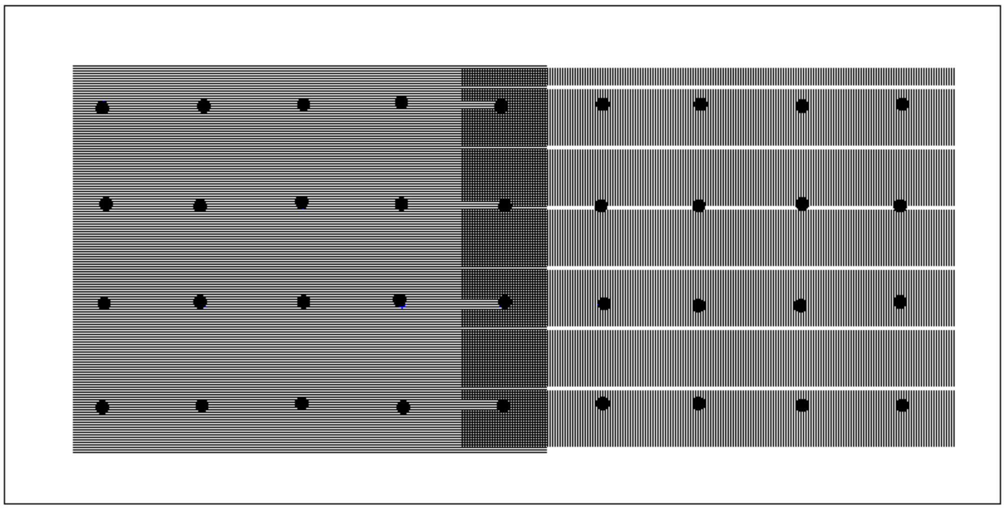

Figure 5.6: Domain Ω split into two subdomains, Ω_1 and Ω_2.

We decided to enumerate the nodes along the columns, i.e. the node at (i, j) position will be consider to be $i + (j - 1)n$-th node in the ordering.

Now we are going to see how does the Schwarz alternating process do works for this problem. We will check both standard variants of SAP, i.e. Schwarz Additive and Schwarz Multiplicative Method. We will not only present only how does the convergence rate for each of the methods looks like, we will also look into the effect of the overlap region's size. We will do this simply by adding a next column to each of the subdomain. We will say. Below we can see a example of adding two columns to the subdomains after discretization, for $n = 4$.

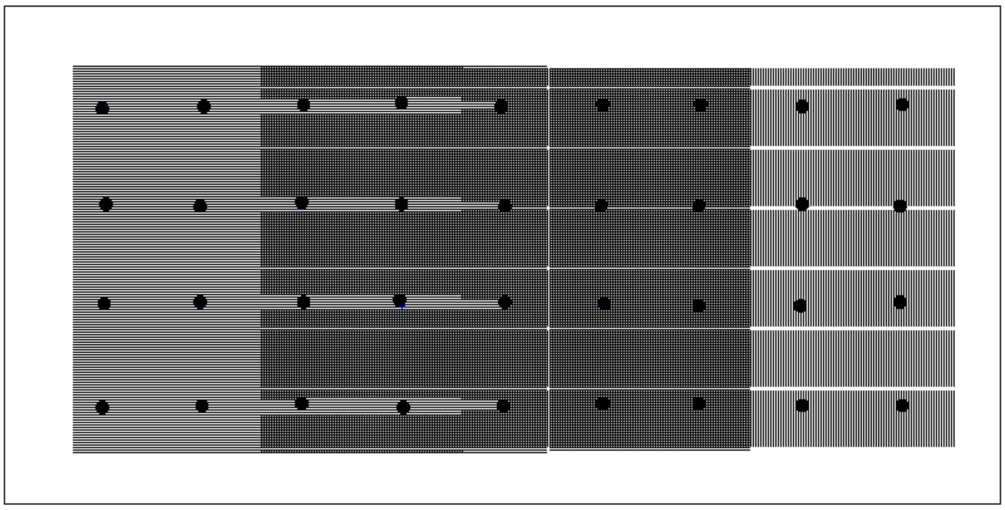

Figure 5.7: Domain Ω split into two subdomains, Ω_1 and Ω_2, with extended overlap of size 2.

We will say, that a subdomain has a extended overlap of size k, if it will consist of the base subdomain nodes and k additional columns.

5.4.1 Multiplicative Schwarz Method

First, we will look at the MSM method. We present how does the convergence behavior changes by increment of the overlapping region of two subdomains.

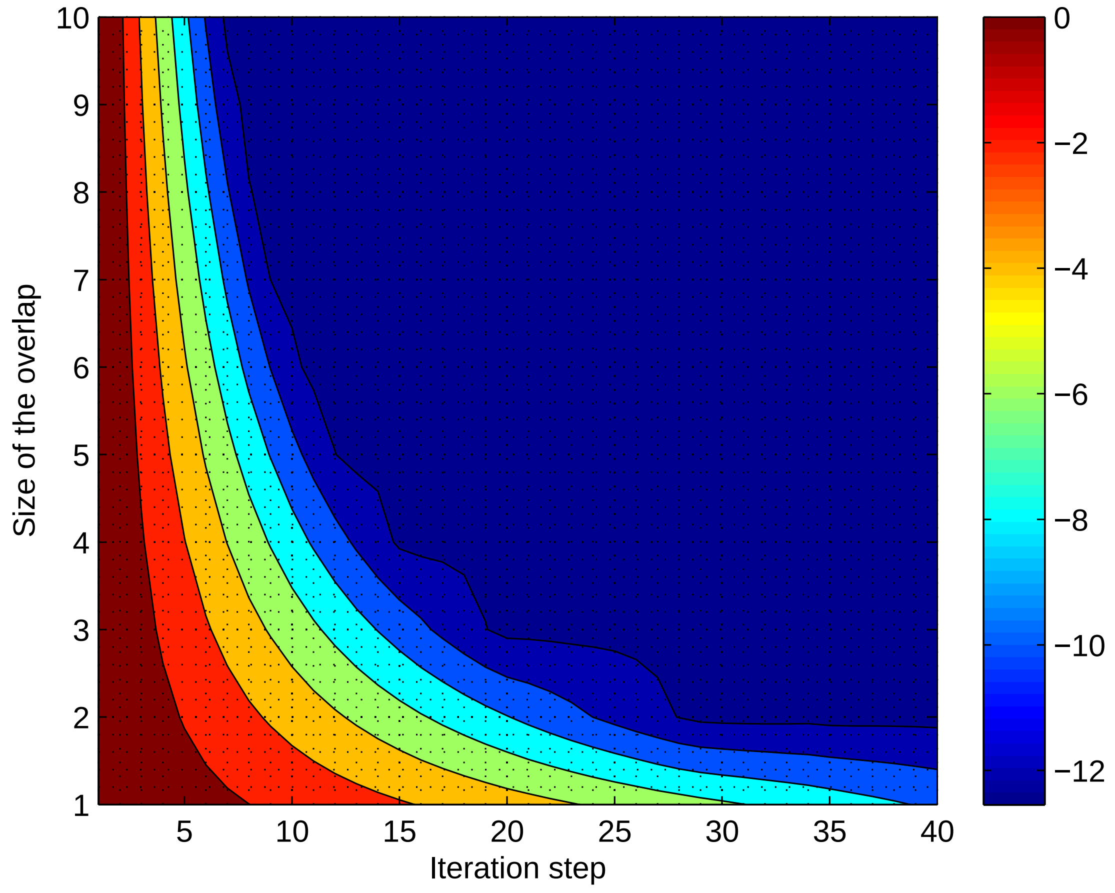

Figure 5.8: Contour plot of the log_{10} (i-th residuum) for MSM

The figure above shows us an essential information about the Multiplicative Schwarz Method. We see, that depending on the size of the overlap of two subdomains, we get faster or slover convergence of the method. We can notice, that the we are dealing here with a logarithmic dependency, between the size of overlap and number of iteration needed to achieve a certain error size.

5.4.2 Additive Schwarz Method

Now, we will present the result of the same experiment done for ASM.

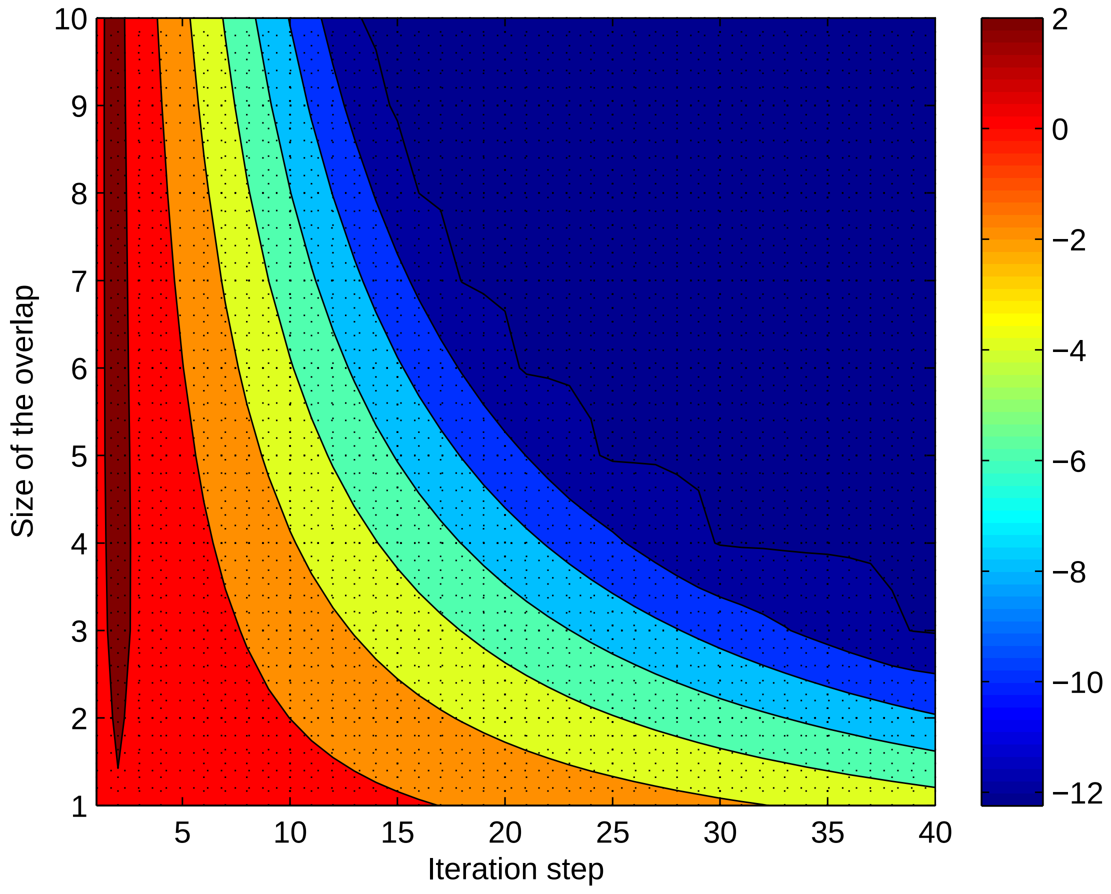

Figure 5.9: Contour plot of the log_{10} (i-th residuum) for ASM

As it was expected, we get a similar result as in MSM. Also here we have a logarithmic dependency between the number of iterations needed to achieve wanted error size and the size of overlap. However we may notice a slight slowdown of the convergence rate, which was expected, because ASM unlike MSM is computing the correction from each subdomain without any updates from neighboring subdomains. This is a drawback, however we can easily neglected it, because of the fact, that the corrections can be computed simultaneously, which can speeds up the whole processes in the sense of spent time.

We stop at this moment further investigation of the Schwarz method, due to the fact, that it is tested on a trivial problem. We will go deeper in the next chapter, when we will be working with test problems, provided by Plaxis.

Chapter 6

Numerical Tests and Experiments

6.1 Introduction

After presenting all the theory, which was needed, we can start to work in finding a good, parallel-friendly, solver for linear systems which are generated by Plaxis software. We have already done some tests, which are presented in the Literature Study Report and we will not show them again. However, we will recall our initial guess for the method which we will use to solve systems.

6.2 Choice of the Method

Initially we have chosen, a Domain Decomposition approach was a starting point for solving the problem, due to the fact of the good results noted in this field of mathematics. Nowadays, domain decomposition methods are getting more popular, because the underlying idea gives plenty of possibilities to parallelize the solution process, which is a wanted property when solving big systems of equations.

From the big family of methods of this branch of mathematics, we have chosen to use one of the Schwarz Alternating Processes approaches, i.e the Additive Schwarz Method (ASM) as our solver. First reason for it was the fact, that the code can be easily converted into a parallel program. Secondly, there was already a try to use a Schur Complement approach, but the results were not as good as it was expected. Also there are examples of many successful implementations of Schwarz methods in real life problems, which only encouraged us to take ASM as our framebox for the solution of the linear systems.

Now when we have chosen ASM, to be our main core of the solver, we are going to upgrade it, by incorporating onto it two methods, which were presented in this report, i.e. Preconditioned Conjugate Gradient Method, for the solution of the linear systems of the subdomains, and the Deflation method to improve the spread of the information coming from one subdomain to other ones. This combined together is going to be our tool in solving the linear systems.

The choice of the subdomains on which we will perform the ASM is going to be done not by an arbitrary cut of the coefficient matrix. Instead, we are going to create the blocks in the preconditioner, by adding the elements from the FEM process, which contribute to an area of the same material type. In this way, we will have a possibility to incorporate an intelligent partition of the domain, which will preserve the physical structure of the problem.

Remark Unless stated otherwise, all implementations of the algorithms which were presented in this Master Thesis and are used in numerical experiments, are written by us in Matlab. Also, if not mentioned, the machine used for tests is a Lenovo computer (Model R61) with a Intel(R) Core(TM)2 Duo CPU T5550 @ 1.83GHz and 2 GB of RAM.

6.3 Region Growing Algorithm

We start by using simple problems in which layers are regular and the same number of elements. We were able to set subdomains easily, by looking at the picture of the problem. Now, it is natural to start working with more complicated examples. It is obvious, that the first thing which should be changed is the structure of the layer. Instead of having rectangular layers, which are parallel to each other, we consider problems in which the parts of the same material type have different shapes and sizes. Some of them may be embedded in other layers. Because of that, we need to have a tool, which we will use to recognize those areas inside the domain. For this purpose, we develop a Region Growing approach. However, we need additional information about the connectivity of Elements from the FEM discretisation, to be to perform in an efficient way.

6.3.1 Region Growing 1.0 algorithm

The first approach to this issue, based on the available knowledge, was a direct, primitive algorithm. Due to the fact, that at the beginning all we knew about the Element was what kind of material it is made of, we could separate the set of all Elements into disjoint subsets of Elements of the same type. Next, we take the first Element, put it into an array and compare it with each of the other Elements from the same set. If the intersection of the indices of the degrees of freedom was not empty, we know that they are connected with each other. So we put the checked Element into the array and remove it from the set. After that we check the next one from the set till the last one. After that, we go the second Element in the array, and perform the procedure of comparing and removing from the set in a loop, until the point when there was no expansion of the array in an iteration. This would mean, that all Elements in the array are creating an area of the same material type, which is not connected with any other Element of the same type. After this, if the subset of Element with the same material type were still non empty, then we would perform the procedure again, as described above, till the moment when there were no more Elements of the same type, which would be in the base subset. Then we would go the next material type and do the same again etc.
As we see, this algorithm is quite primitive, because in each loop it has to compare the chosen Element with all the Elements of the same material type, which for big problems can be a serious drawback. Actually we can come up with a simple example of a problem, in which the number of needed comparisons is at least $\frac{(n-1)(n-2)}{2}$, where n is the number of Elements. If we consider Elements as nodes which are connected whenever the Elements share at least one degree of freedom, then we will see that for a path graph **[16]** we will have to do exactly $\frac{(n-1)(n-2)}{2}$ calls of the comparing function, because the first node will need to be compared with $(n-1)$ nodes, the second one with $(n-2)$ and so on. By induction we can state that ultimately we will end up with $\frac{(n-1)(n-2)}{2}$ comparisons.
However, we have an improved version of this algorithm, which works much faster.

6.3.2 Region Growing 1.1 algorithm

The main difference, between the improved version and the original one lies in the change in the comparison procedure. Rather then comparing in each outer loop, the degrees of freedom of the chosen Element with the Elements taken in the inner loop to see, whenever both Elements are connected, we now remember each degree of freedom of Elements already included in the array and compare them with the ones of the checked Element. This tightens up the whole procedure considerably, because in each call of the function we have already checked to see it if the Element is connected with any of the Elements from the area which are known at this time.We also notice, that if an Element is connected with the area, then it is included to the area, which means that the next Element which will be checked will be checked whenever it is also connected with the just included Element. Those two advantages makes this Region Growing algorithm a version which can and should be used instead of the first one.

6.3.3 Region Growing 2.0 algorithm

We see that both algorithms do not use any information about the connectivity of the Elements, which implies that if we have two disjointed areas of the same material type, then when we "discover" the first one, we will always check all Elements from the second one, if they are connected with it. This leads to the conclusion, that we will have plenty of unnecessary function calls. Also if we manipulate the example used to show how many comparisons we need to perform, namely we would then change the enumeration, then we could end up again with the same number.
For that reason we decided to get more information about the Elements, to have a better algorithm for recognition of the areas of the same material type. We receive information about the neighbours of an Element. With this, we can now rewrite our algorithm to a form where we will not do any unnecessary calls function. Actually, our problem can now be seen, as finding spanning trees in a graph, which is a well known problem in the Graph Theory and has several algorithms for finding them. Because in our case each call of the function is regarded as the same in the sense of cost to perform it, we can easily choose for our problem the *BFS* (Breadth-first search) algorithm as the optimal tool.
A spanning tree of a connected, undirected graph G, is a tree composed of its vertices. For more information, see **[15]**.

We conclude this section with a table which shows the CPU time for each of the presented algorithms to find each area for *Problem 6* [see *Appendix 8.1*]:

Algorithm	Time
Version 1.0	493.717990 seconds.
Version 1.1	5.24848 seconds
Version 2.0	0.673795 seconds.

Table 6.1: CPU time for Region Growth Algorithms

We can find Matlab implementations of versions 1.1 and 2.0 in the *Appendix* (8.2).

6.4 Deflation Tests

In the Literature Study Report, all of the presented results are from tests, without Deflation. The reason for this is that there were no reasonable results which could be shown. As we emphasized before, Deflation plays an important role in our study and that is why we investigated this subject. In this chapter we show the results obtained when analysing the influence of certain strategies while constructing deflation vectors. We would like also to mention, that the vectors are computed

6.4.1 The Choice of Deflation Vectors

We start by answering the question about the construction of the Deflation matrix, which plays an important role in the whole algorithm. At the beginning of this project, we decided to use the Rigid Body Modes approach for the creation of Deflation Vectors. We will now present the results, which will confirm that our decision was the right choice.
We do the following. For each area of the same material type, we assemble all associated Elements to it and add them together. The result of this operation will be then used to compute its null space which are used as Deflation(s) Vector(s). We will check whenever we should take all areas or we can neglect some of them. As our first test problem for this, we will use *Problem 3*. Below we can see, how does the inclusion of different layers affect the convergence rate of the method. First we will look at the case, when we do not usethe Schwarz method in preconditioning. Instead, we will use a diagonal scaling. This approach allows us to see the real effect of the Deflation on the convergence of DPCG.

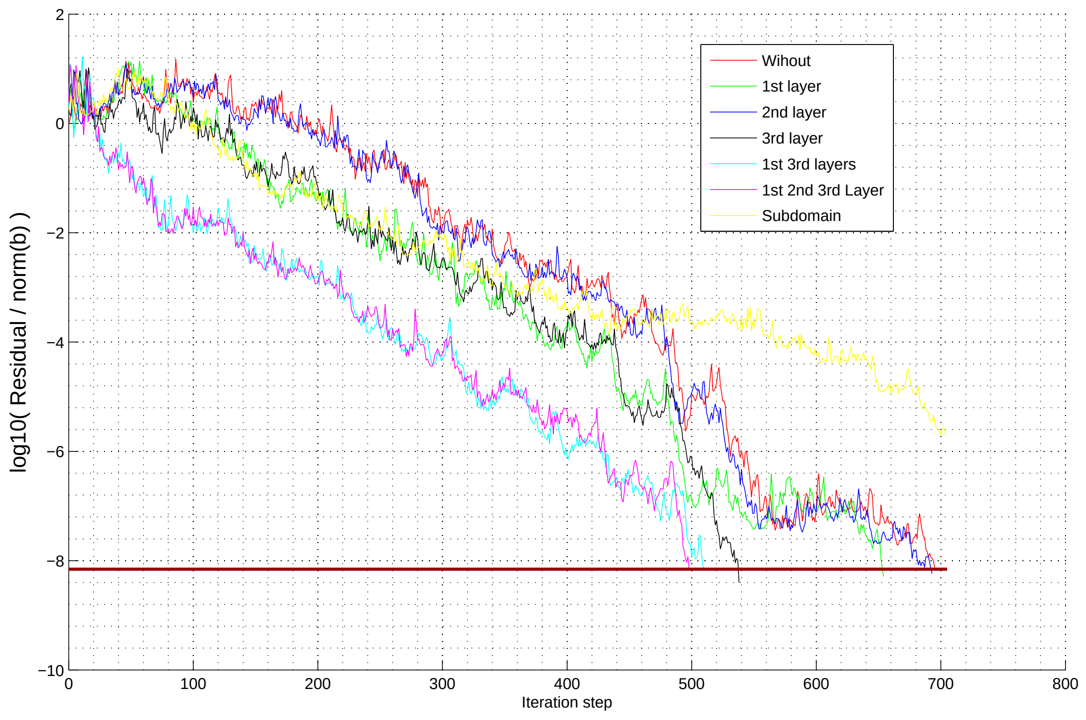

Figure 6.1: Problem 3 with diagonal scaling and several choices for deflation vectors

In the picture we can see, that the use of Deflation technique, can have a big impact on the convergence rate of the method. Because we have only 4 layers in this problem and the one on the bottom is fixed, we can have only several possibilities to test. The first and the third layer is composed of a much stiffer material than the second one. We see, that the best results we get, when we include those two stiff layers in the Deflation matrix. We also notice, that an additional inclusion of the second layer does not improve significantly the already achieved convergence rate.
The results of this test were stimulating to check whenever the observed property is a singularity or a regularity. For that we prepared a variation of *Problem 3*, namely *Problem 4*, in which we have more layers made of stiff and soft material types, which are ordered from the top. We present now the results of the same experiment for the new problem.

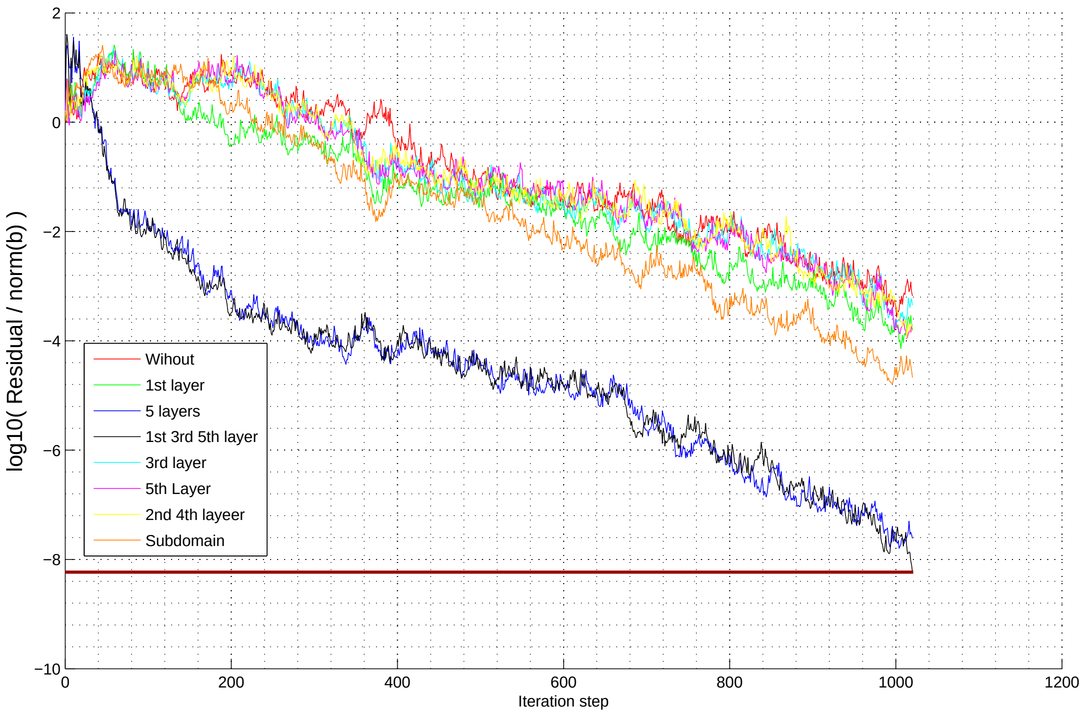

Figure 6.2: Problem 4 with diagonal scaling an several deflation vectors choices

Once again, we notice, that the best choice for the Deflation Vectors, is to take the null spaces of the stiff layers from the Problem. In this example, this are the 1st, 3rd and the 5th one. We see, that we deal here with a noticeable change of the error size. Also in this problem we can notice, that the addition to the Deflation Vectors the ones which correspond to the soft layers, when we have already included the stiff ones, does not improve the convergence rate.
At this point, we need to emphasize the fact, that those test were done without the Schwarz method as preconditioner. Therefore it is crucial, to check whenever the result which were just presented are valid for the Schwarz preconditioner. That is why, we will now show the output of the same tests, but with Additive Schwarz Method (ASM) used as the preconditioner.

We return now to the *Problem 3*.

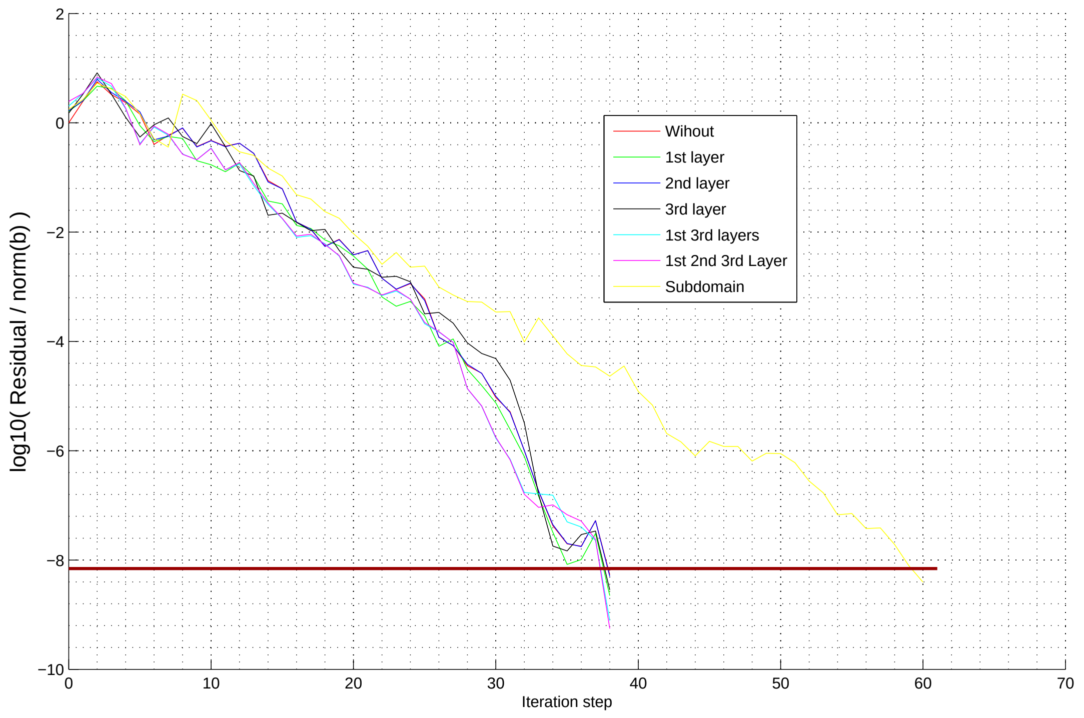

Figure 6.3: Problem 3 with ASM preconditioning and Complete Cholesky Factorization

In this example, we use 4 subdomains for the preconditioner, each corresponding to a layer of the same material type soil. We notice that the use of Deflation method which we aim to use, has no significant affect on the speed of finding the solution. We only see, that it does not deteriorate it, as it is in the Subdomain Deflation method. However, in this case we used Complete Cholesky Factorization for the precondioner, which forces us to remember all values of it, which is a serious drawback in the memory sense. Therefore, we will investigate now, if the change to an Incomplete Cholesky Factorization, will affect the observed results.

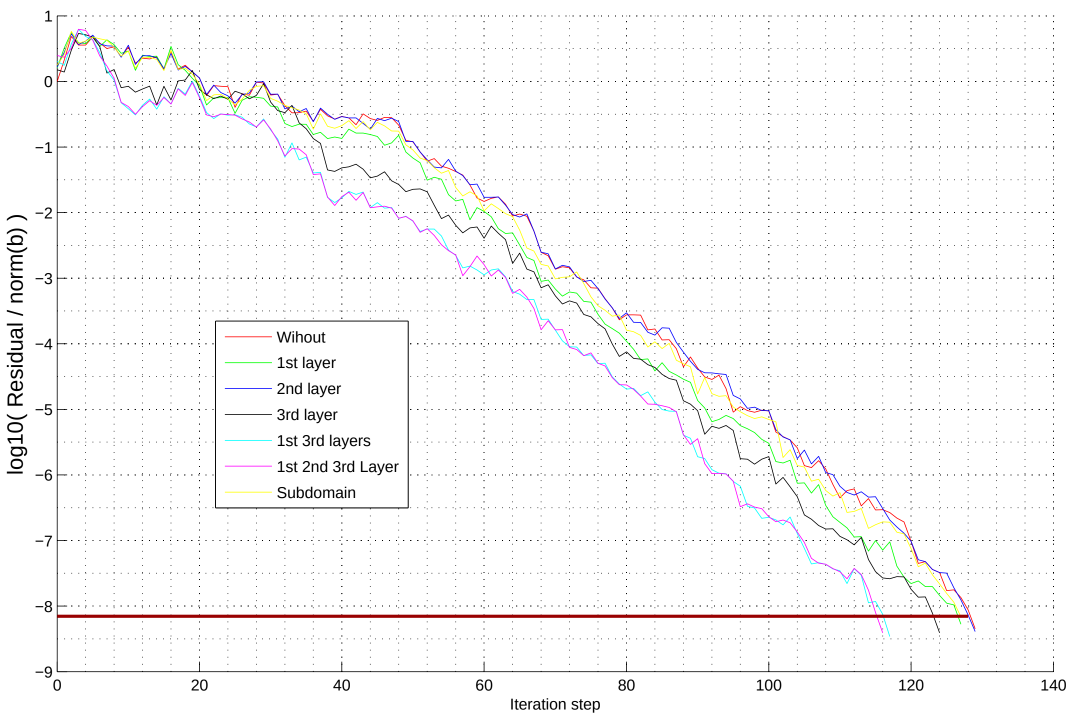

Figure 6.4: Problem 3 with ASM preconditioning and Incomplete Cholesky Factorization

The drop tolerance in Incomplete Cholesky is set to 10^{-3} and we use the same preconditioner as before. We now observe, that the Deflation is now making a difference in the speed of our method. Once again, we see that the best choice is to take the stiff layers for the Deflation Vectors, and once again we see that the addition of soft layers, if we already took the stiff ones does not drastically improve the rate of convergence. Nevertheless, we observe, that the difference in the number of iteration of DPCG with only stiff layers and PCG without any Deflation is equal 16. This may be regarded as not much, however the size of the problem is also small. We may expect that, for larger problems this number will be more perceptible.
Let us now check the same for the second problem used in this section, i.e. *Problem 4*.

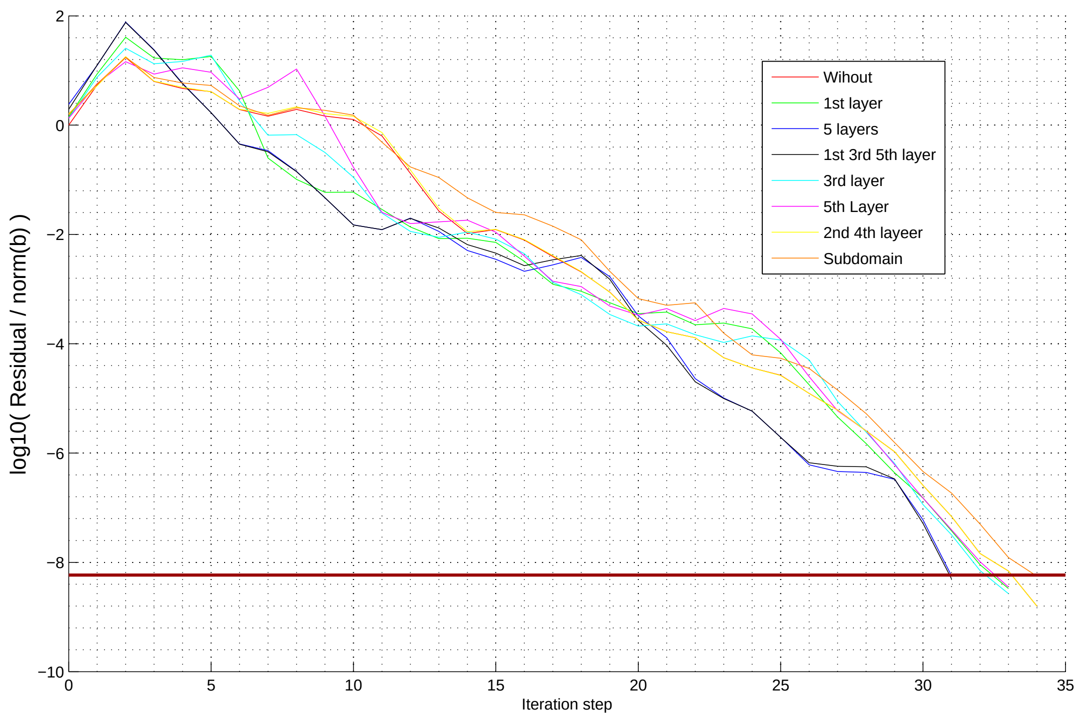

Figure 6.5: Problem 4 with ASM preconditioning and Complete Cholesky Factorization

As we could expect, the number of iterations needed to achieve the desired level for all chosen methods is quite close. This is similar to the results for *Problem 3*. However in this example, the numerical fluctuation is much stronger, therefore we have a difference in the speed. Because of that, we may expect a bigger gap between the DPCG for the stiff layers and PCG, when we will the use Incomplete Cholesky Factorization.

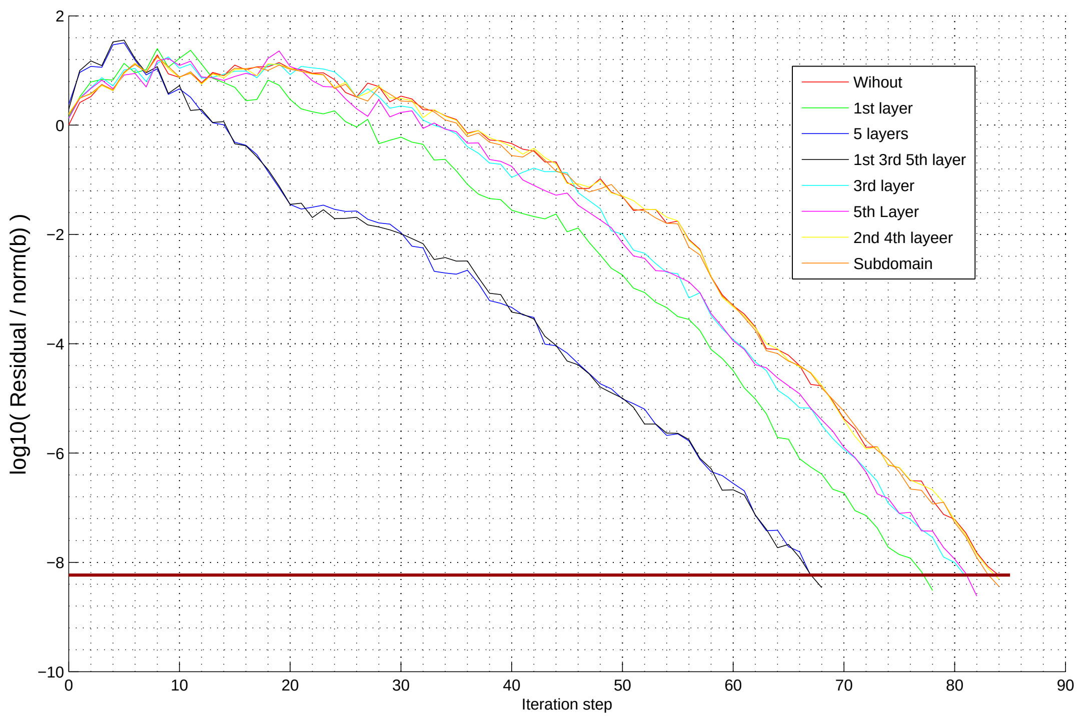

Figure 6.6: Problem 4 with ASM preconditioning and Incomplete Cholesky Factorization

Again, one more time we see, that the fastest method is the one with the stiff layers used to compute the Deflation Vectors. And once again we see that the addition of soft layers has no affect on the speed.
We conclude this section on the base of the presented results with a statement, that the best choice to create the Deflation Vectors is a physic based approach, which recognizes areas of the same material type. Also, we can neglect the soft areas due to the fact, that the influence of the stiff domain dominate the effect of Deflation.

6.5 Choice of the subdomains

We already presented some results of the tests which shown us how to choose the subdomains in the Literature Study Report. However, at that time we did not use Deflation method in it, which can speed up the whole process. Also, the fact that at that time we were working mainly with *Problem 3*, which is quite regular. Therefore, we show some of our new results in the case of the choice of the subdomain, which will confirm our consequent selection of the physic based decomposition of the domain.
We start with a study of the number of subdomains, for the *Problem 4*.The problem consist of 120 elements, which compose 6 disjoint layers of the same material type. Each layer can be decompose into 2 sublayers, hence we deal here with 12 sublayers which can be combined into 2, 3, 4, 6 and 12 subdomains. We now present the result of the tests which shows us how the number of the specified subdomains affects the convergence rate. We use the Deflation Vectors made from the 1st, 3rd and 5th layer (Stiff ones). For the following results, we do not use Cholesky Factorisation with a drop tolerance. Instead we use the whole information from it.

We also set the stop criterion to 10^{-6}.

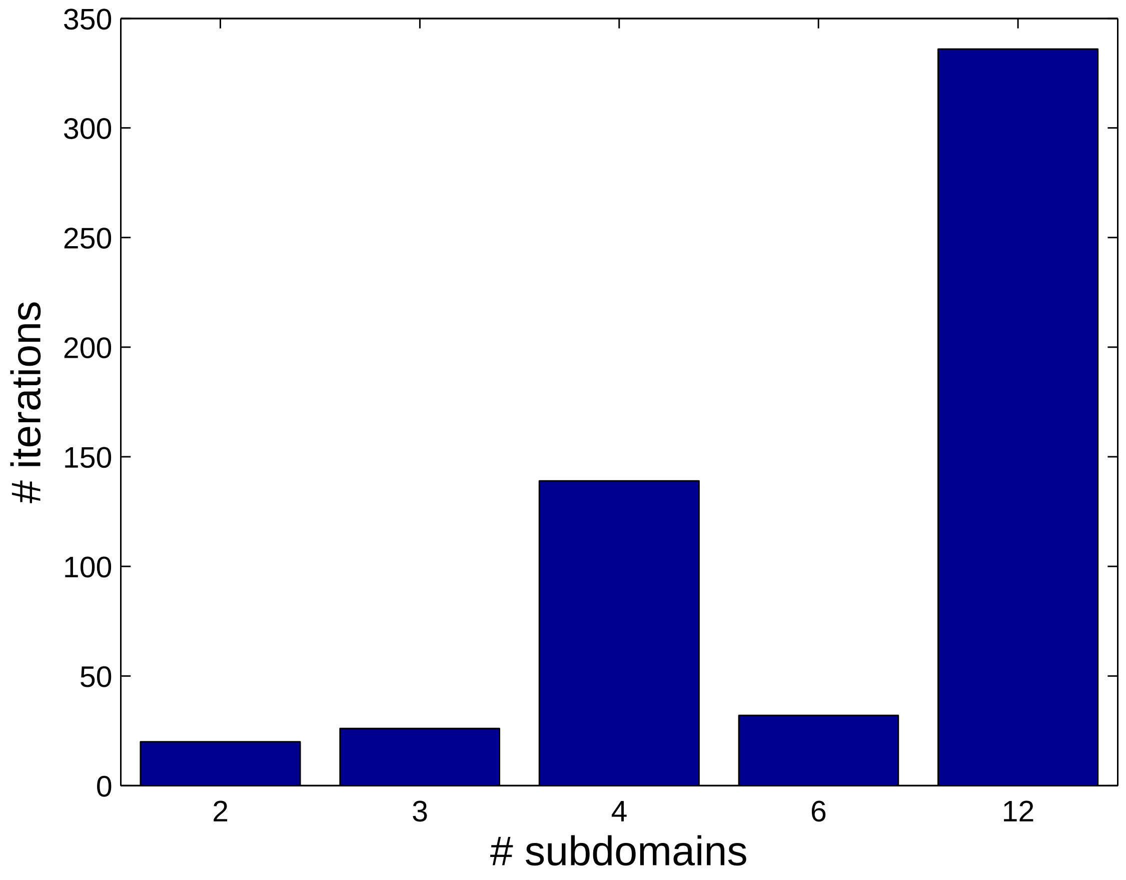

Figure 6.7: Problem 4 with Deflation and ASM preconditioning

We can see, that in two cases, we need much more iterations to complete the task of solving the system, i.e. for the case with 4 layers and 12 layers. Both of them have the same property, which is the decomposition of the layers of the same material type. In the case of only 2 subdomain we assembled first 3 layers to one of the subdomains and the next 3 layers to the second one. In the case of 3 subdomains we assembled 2 layers to each. In the case of 6 subdomains we just take each layer as a subdomain. But for 4 subdomains, we had to stop thinking in layers of material type and start to look at the problem from the angle of sublayers. Therefore we had to assemble 3 sublayers to each subdomain, which destroyed the physic based structure of the preconditioner. The same problem occurs with 12 layers. Hence we can conclude, that the decomposition of the layers of the same material type in the setting of the subdomains has a strong influence in deteriorating the rate of the convergence.
Due to the fact, that the aim is to use Incomplete Cholesky Factorisation within the solution of the problem, we will now present the results of the tests with the use of Cholesky. How does the size of the drop tolerance in the Cholesky Factorisation affects the convergence rate in this test. We set the drop tolerance to 10^{-3}.

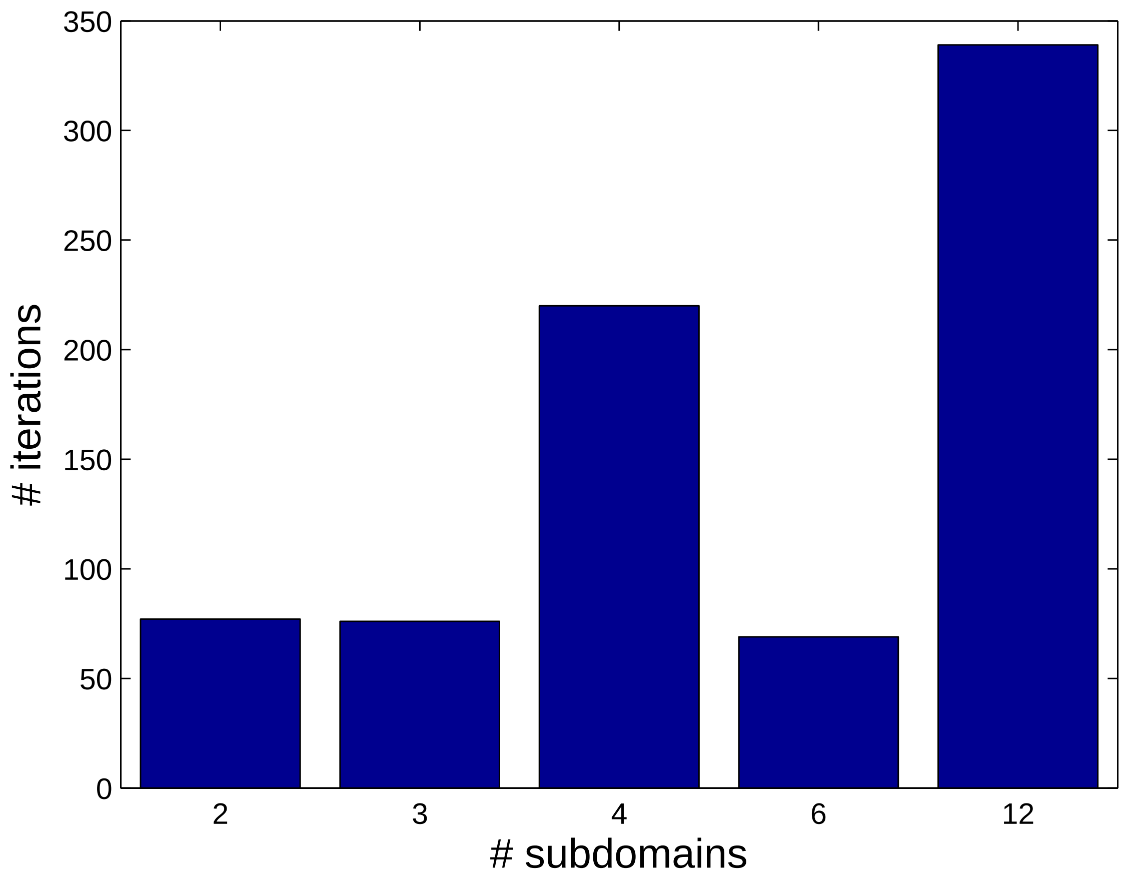

Figure 6.8: Problem 4 with Deflation and ASM preconditioning

As we could expect, the number of iteration has grown, because we have increased the size of the drop tolerance, hence we lose more information about the problem. However, the crucial aspect of those results is that we have preserved the behaviour which was noticed in the first test, i.e. that the smallest number of iterations is when we do not allow to decompose the areas of the same material within the problem. Also an interesting fact can be read from the plot above, namely the smallest number of iterations is achieved when we take 6 subdomains for creating the preconditioner. Although it is not visible in the picture, we need to perform 69 iteration steps for this case, when for 2 subdomains we need 77 iterations and for 3 subdomains 76.

6.5.1 Metis

As we seen on the page before, we need to have a strategy when decomposing the domain of the model. However we should also have a solution to a case, when one or more of the subdomains, which were recognized by the Region Growing algorithm, are much bigger then the rest. This can be a serious problem, because we may end up with a scenario, when we will need to wait for the computations associated with one of the subdomains, while all the others will be already done long time ago. Therefore we need to have an idea how to split those big ones.

One of the approaches for solving this problem, could be an algebraic split of the matrix A into number of blocks, with an overlap parameter. However this leads us to the following questions: where should we split? How big should be the overlap? We already know from the tests performed on the whole domains, that the algebraic split is a complex problem and it is not easy to deduct where should we cut, if we only look on the matrix. We did some tests with this approach only to conclude, that actually we move in the dark and there is no simply answers to questions stated before.

On the other hand, we could divide the Elements, which contribute to the big subdomains, into subsets, where each of them would be now an independent subdomain. But then what should be the criterion for creating subsets?
Definitely, the Elements from one set should be connected with each other. For that, we would need a tool which ensures that we do have this property. We could use the Region Growing algorithm and apply it to the big subdomain. After having almost the half of the Elements in the array, we would stop and set the input of it as one of the new subdomains. For the rest we would have to apply the algorithm one more time, to be sure that it is connected. This seems to be a good idea, but actually it does not gives us certainty, that we will end up with similar sized subdomains, due to the fact, that the region growths in each side. This can , combined with some structures of the mesh, to bad cases, when we would have one medium sized subdomain, and a few small ones. Also everything would depend on the seed Element, which would be chosen as the starting one. We could think of trying to modify the Region Growth algorithm, to somehow avoid those obstacles, for example try to use DFS (Depth First Search) instead of BFS, but it would lead us only to reformulation of the difficulties which occurs. Actually the roots of those problems lies in the essence of graph search algorithms, or to be more precisely in a graph itself and it's topological aspects, like the lack of recognition of the space in which it is. Therefore we need to neglect this approach due to it's internal barrier.

Fortunately for us, there is a tool for creating those subsets called the **MeTiS** software.
MeTiS is a set of serial programs for partitioning graphs, partitioning finite element meshes, and producing fill reducing orderings for sparse matrices. The algorithms implemented in MeTiS are based on the multilevel recursive-bisection, multilevel k-way, and multi-constraint partitioning schemes. For more information, see **[References to MeTiS Manual]**

We now present the results of partitioning the big layers via MeTiS. For the tests, we decide first to take X **Problem 8a** and **Problem 8b**. We set the drop tolerance in Incomplete Cholesky Factorisation to 10^{-2}. For the Deflation Vectors we take all of the recognizable layers. It means, that we do not use **MeTiS** at this step.
First, **Problem 8a**.

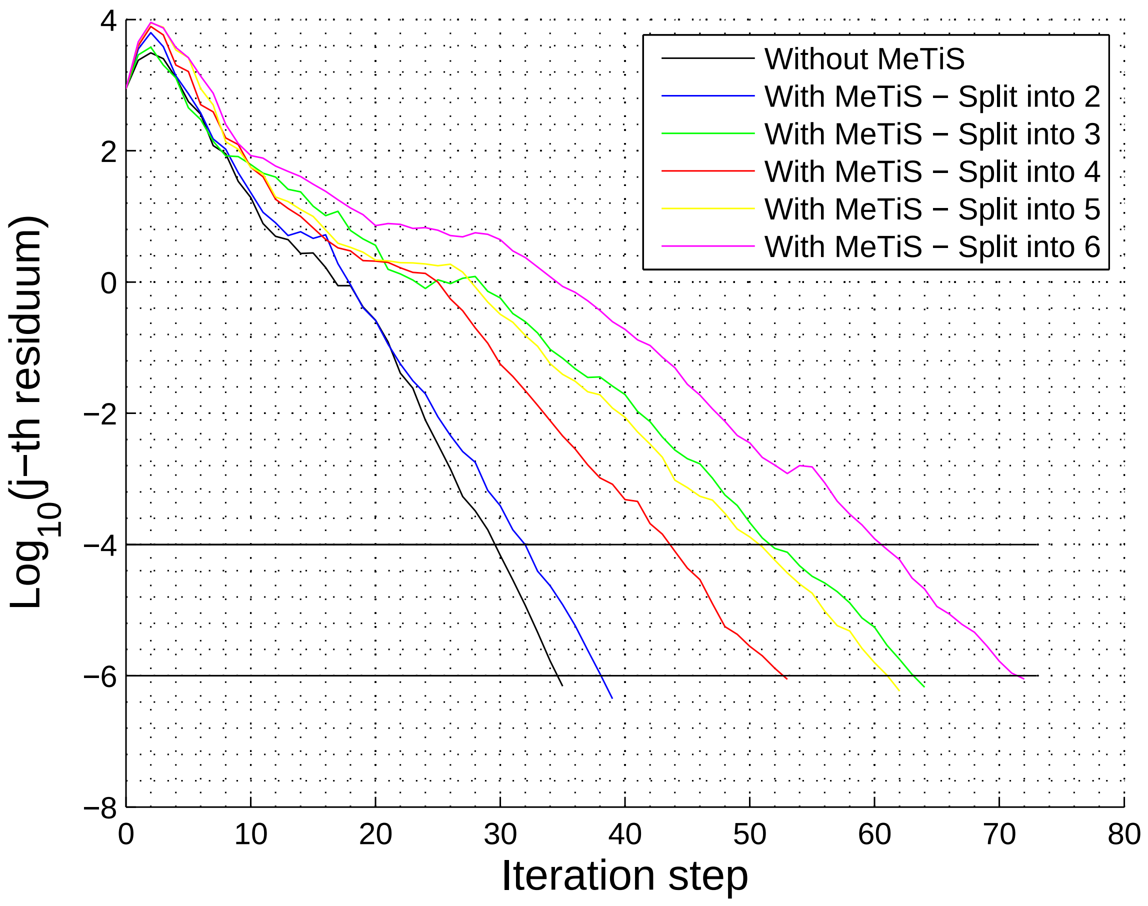

Figure 6.9: Problem 8a and split of the layer with MeTiS

From the figure above we clearly see, that MeTiS works splendid. The split of the big layers into two, has only a slight affect on the convergence rate. We see that they are almost the same, the difference is of the size of 4 iteration only, which is a really small number, if we like to compare it with several tests which were perform with the use of algebraic split. We also notice an interesting fact, which is that the split into 4 subdomains, is better than splitting it into 3. This is a result, which is at least intriguing, and for which we do not have an explanation for now.

Now, **Problem 8b**. Again, we are the witness of the magnificent work of MeTiS software.

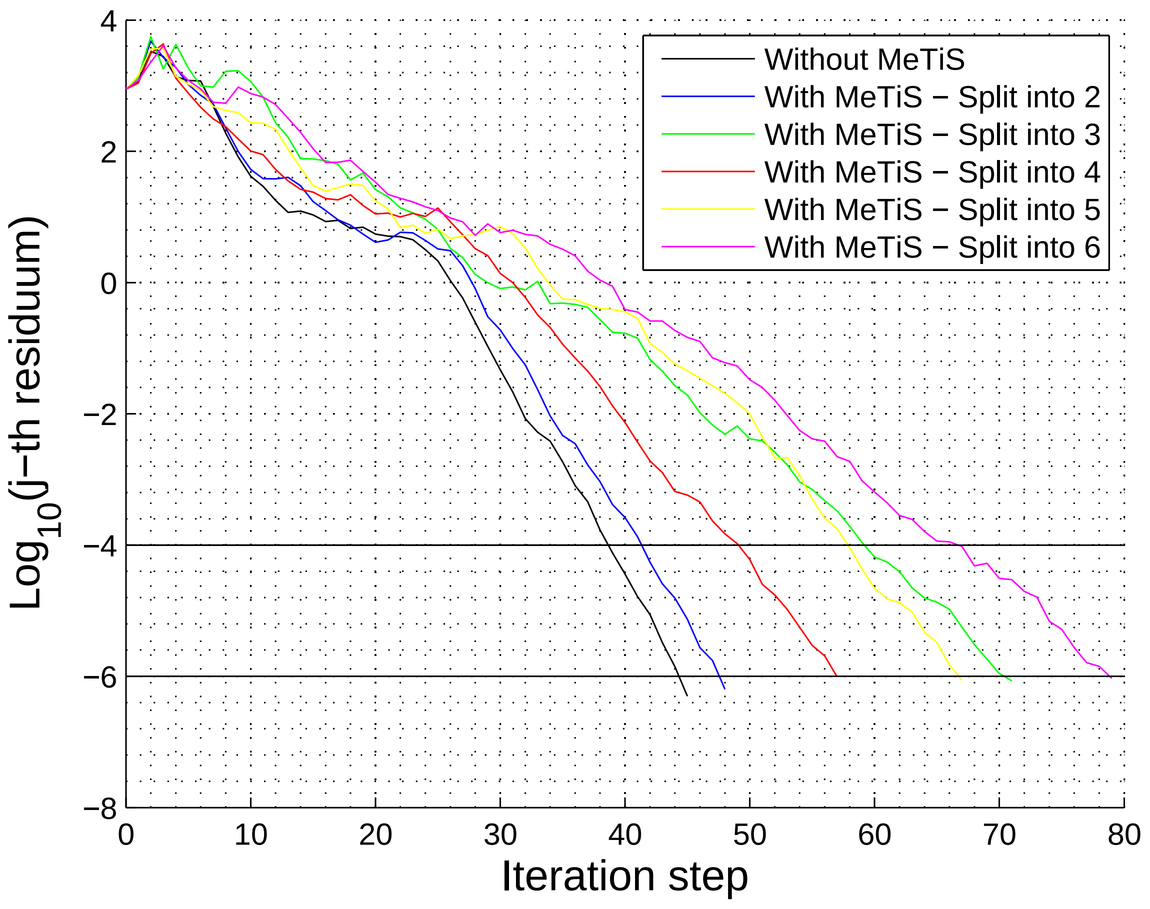

Figure 6.10: Problem 8b and split of the layer with MeTiS

In this case, the difference between the approach without splitting the biggest subdomain and the one with it, is only 3 iteration. Again, it is a big success for MeTiS. However we see, that it takes more iterations for solving the problem, than it took in the previous test, but this is due to some numerical problems with computing the Deflation Vectors which actually can be notice, if we look at the curves of the error functions. Also in this problem, we encounter the same behavior between the split into 3 and into 4 subdomains.

Presented results of both problems and the ones which were not, encourages us into further investigation and application of MeTiS as a essential tool for splitting bigger subdomains.

After seeing the results of the test with MeTiS done for two versions of Problem 8, we going one step further, to **Problem 9**, which also has one big layer. Once again we would like to see, how does the partition of it affects the convergence of the method. For the Deflation Vectors, we take subdomains which correspond to the stiff areas. We set the drop tolerance for Incomplete Cholesky to 10^{-2}. Below we see the results of this test.

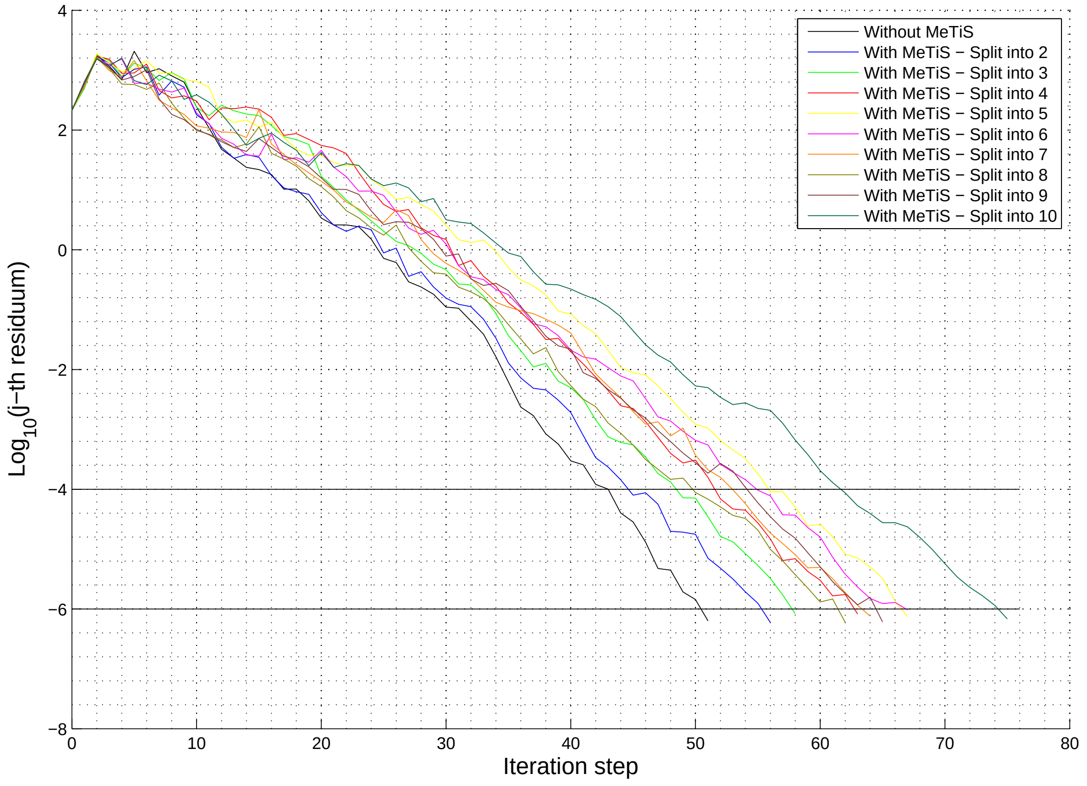

Figure 6.11: Problem 9 and split of the layer with MeTiS

Again, we see that the choice of using MeTiS as a tool to split big subdomains, is a right choice. After splitting the biggest part of the model into 2, the number of needed iteration is only increased by 5. This, combined with the fact, that we use a drop tolerance of size 10^{-2} is an achievement, which should not be neglected.
Also in this problem, we observe a behaviour which may seems odd, i.e. the speed of the case, when we split into 8 subdomains. It turns out, that it is much better to divide the big layer into 8 parts instead of 4,5,6 or 7. Unfortunately, we still do not know what is the reason for that. We may expect only, that it is connected with the structure of the graph which corresponds to the connections of the Elements. That it is much better in the sense of minimizing the loss of connections, to divide it into 8 parts. But this is a thing, which could not be predicted.

6.6 Contrast between an algebraical and a domain based preconditioning

We would like to show the contrast in using the investigated approach which incorporates the physics of the problem, with one which does not take it into account, i.e. an algebraical split. The following test was performed in Matlab. To illustrate this, we use *Problem 7* as our test problem.

- In the **first** solution, we will use our Region Growing Algorithm to identify areas of the same material type, and use them to create preconditioner blocks.
- In the **second** approach, we will create the blocks by simply dividing the coefficient matrix A into 6 blocks in the following way:
  We will take the dimension size n, divide it by six and call it l. Then we will create the first block by cutting from matrix A a squared matrix, starting from the position $(1,1)$ and ending at $(l + overlap, l + overlap)$. The second block will be created in the same way, but now the starting point will be $(l - overlap, l - overlap)$ and the ending one at $(2l + overlap, 2l + overlap)$, and so one. In our case we have taken the overlap of size 20.

For this test we do not use Deflation technique, neither Cholesky Factorisation. Below we can see the result of this test:

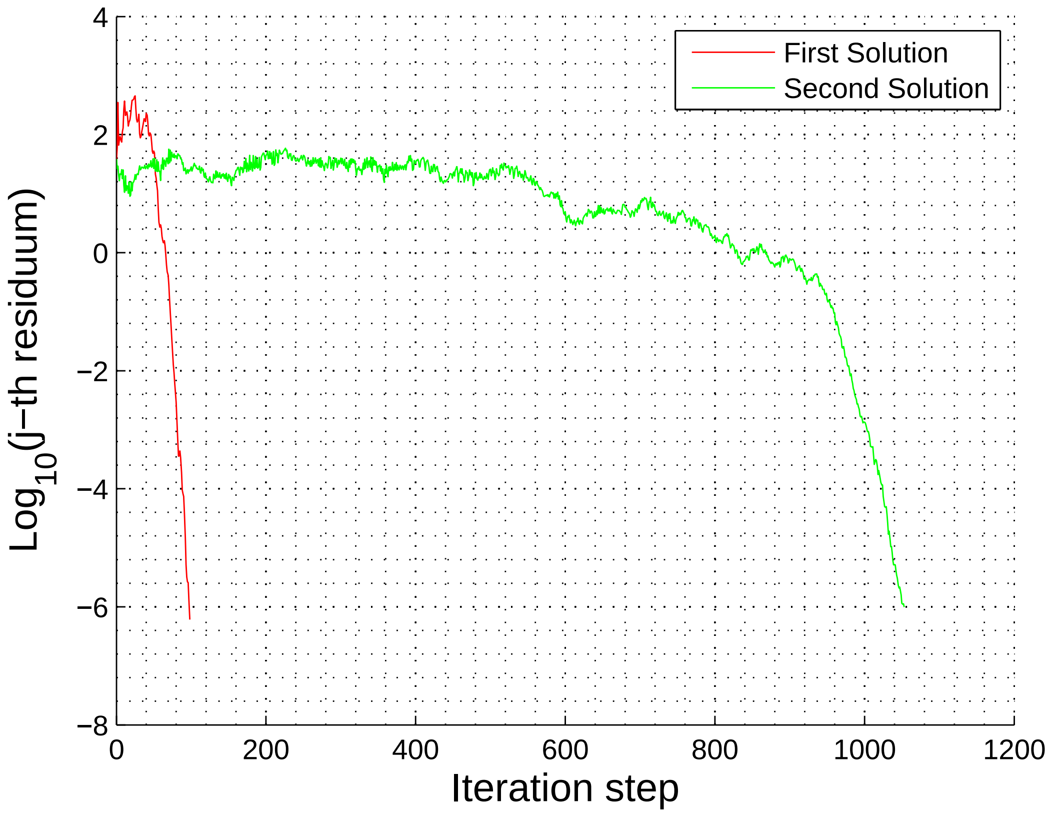

Figure 6.12: Problem 7 without Deflation

This plot shows us the true influence of the physic based approach to the problem. We can see, that the number of iteration needed to achieve the desired size of the residual, which in our case is 10^{-6}, is for the first solution a really small number, if we compare it with the second one. We see here the power of the inclusion of physics in choosing how to partition the problem into subdomains. The plot above is one more proof, that the algebraic split of the coefficient matrix A, which does not incorporate the knowledge about the modelled phenomenon, is no match to the one based on physical knowledge.

6.7 Tests with the Habanera solver

Some of the test cases, which we obtained from Plaxis, were of the size, that we could not use Matlab any more, due to its limitation. Therefore we switched to C, and rewrote most of the code, but there occurred some problems with the implementation of the Incomplete Cholesky Factorisation. However, by the kindness of Habenera, the company which is writing the parallel version of the solver, which is going to be used in the Plaxis software, we are able to use a preliminary version of it, to check, how does our partitioning work in the real environment.

Picos (**P**laxis **I**terative **Co**ncurrent Solver) [14]can be used to solve sparse linear systems of equations of the form

$$Kx = b \tag{6.1}$$

efficiently on computers with multiple processing cores. It is based on the domain decomposition method in which concurrency is obtained by dividing the degrees of freedom (DOFs) into slightly overlapping groups, called subdomains, that are assigned to different threads of execution. PICOS applies a local, domain-wise preconditioner in combination with a global algebraic coarse grid preconditioner to improve the convergence rate of the solution process.
In the early stage of the development of PICOS, the preconditioner was based on the MeTiS software applied to the nodes of model. We will refer from now on to this partition scheme as *MeTiS based.* Two types of solvers are provided by PICOS: one based on the Conjugate Gradient (CG) method, and another based on the Generalized Minimum Residual (GMRES) method. The former can be used only to solve linear systems involving a symmetric, positive definite coefficient matrix. The latter can handle non-symmetric coefficient matrices and is more robust than CG, but requires more memory. In our tests we will work only with the CG based method.
The preconditioner is based on the Restrictive Additive Schwarz method, a variation of the normal ASM, with non overlapping subdomains. Then, it adds to each of the subdomain one an additional layer of Degrees of Freedom, which are associated with the nodes of the outer sphere of the subdomain. They are use to compute a better correction for the subdomain. They are not treated as the part of the subdomain, they are only use to achieve a more accurate solution, i.e. when computing the correction for the subdomain, we use the original Degrees of Freedom from the subdomain **and** the additional ones. But after this, when we apply the correction, we apply it only to the original ones.

6.7.1 Problem 6

For the first test, we decided to check the performance of our approach with the *Problem 6*, an example with which we already worked and we know what expect. We could find 6 subdomains, each of them corresponding to one area which consist of one material type.
The results of applying the physics based partitioning in this solver were beyond our expectations. It took only 15 iterations, to obtained the desired solution, while with the MeTiS based partitioning, with the same number of subdomains, had to perform 57 of them. Also, with this partitioning within the Habanera solver, we were able to beat the case, when for the preconditioner we took the whole coefficient matrix A, because it needed 53 iteration to obtain the solution of the same error size.

The PICOS solver incorporates the IC factorisation used by the Plaxis company in their software.

6.7.2 Problem 10

After the encouraging results with *Problem 6*, we moved to a more complex model, namely **Problem 10**, which had some plates and anchors in it. There were 13 types of materials used in this project. We decided that first we will use all of them as independent material types. After applying the Region Growth Algorithm, we received 13 subdomains. When we used the partitioning in the solver, we managed to get 42 iterations. When we used the MeTiS based approach, it took 52.
However, when we looked closely at the sizes of subdomains, we noticed, that couple of the subdomains were really small. They were the ones which were connected to the anchors and plates. We decided, to combine them to see how much it affects the convergence. With this partition, the solver took 24 iterations, when the MeTiS based partitions needed 41.

6.7.3 Problem 5

For the next project, we decided to take an problem which we wanted to solve at the very beginning of this Master Thesis, i.e. *Problem 5*, which is a project that can be described as tunnel embedded in the ground composed of the same material type. In this project, we could recognize 3 subdomains. The first one was the subdomain connected with soil. The second one, was the subdomain corresponding to the interface elements, which are used in the Plaxis software to connect the plate elements with the soil elements. And the last one, was for the tunnel, i.e. the plate elements.
Once again, we were able to achieve great results with our physics-based decomposition approach. To solve the problem we had to perform only 46 iteration steps, while with the MeTiS based approach, with the same number of subdomains, we needed 70. And in the case, when we do not use any partitioning of the domain, and take the whole matrix A as the preconditioner,we needed 125 iteration to achieve the desired solution.
And once again, if we look at the sizes of preconditioner blocks, we can notice that one of the subdomains in the physics based approach is much bigger then the rest. It is the one which were made from the soil material type. The size of this subdomain is a natural situation, because our partition was based on recognizing the areas of the same material type and the model is actually a big area of soil with a tunnel in it. Therefore we decided to split the area into two parts and see the result. For the partitioning, we used the approach described before, i.e. we use the MeTiS software and apply it to the connection graph of Elements from which the coefficient matrix is composed. As the result of this, we have now 4 subdomains.
With the physics-based approach, we need 49 iteration, which compared to the previous 46 is not a loss. The sizes of subdomains are more or less balanced, three out of four are around the size of 12000 DoF, so managed to overcome the non-equal distribution of the subdomain sizes in the partition, while keeping the efficiency. On the other hand, the MeTiS based approach, with 4 subdomains needed 79 iterations to get to the solution. So we can see that it also had to perform more instances of the CG loop in order to solve the problem.

We also split the big layer into 3 layers and ended with 51 iterations with the physic based approach and 80 with the MeTiS based.

6.7.4 Problem 11

After dealing with **Problem 5**, we moved to yet another complex project, which could be created by a regular user of the Plaxis software. We are referring here to problem called Problem 11. In this project, we may recognize several layers of different soil material type, plate elements and beam elements. Overall, this allows us to identify 7 subdomains, based on this knowledge. Once again, we check how much iterations do we need in order to get the solution, when we take the whole coefficient matrix as the preconditioner. The result of this was 125 instances. The MeTiS based approach worked much better for this project, because it needed only 108 iterations. However, when we applied the physics based approach we were able to decrease the number of iterations to only 30, which once again proves the effectiveness of our method.
Based on the situation which happened in the previous example, we were expecting, that one of the subdomains will be much bigger the the rest. And we were right. Therefore, we applied the MeTiS software on the connectivity graph of the Elements in this layer in order to split them. When we run the solver with the new partititon, we needed now 35 iterations, where the MeTiS based approach set to generate 8 subdomains, had to perform 71 more of them, which once again is a remarkable result.

6.7.5 Variations of Problem 6

In the end, we would like to go back to the *Problem 6.* We would like to see if the results which we achieved with our partition scheme are consistent, when changing the parameters of the model. Therefore we created *Problems 6n, 6n1, 6n2, 6n3, 6n4.* Theirs description, with the numbers of Elements, nodes, etc. can be found in the *Appendix 8.3* . Like in *Problem 6*, we have six subdomains, which correspond to the areas which we could recognize. Now we will only shortly characterize them:

- *Problem 6n* - The top layer is composed from a material which volume cannot change.
- *Problem 6n1* - The mesh is 4 times finer than in *Problem 6.*
- *Problem 6n2* - The mesh is 10 times finer than in *Problem 6.*
- *Problem 6n3* - The top layer is 10 times stiffer than in *Problem 6n.*
- *Problem 6n4* - The top layer is 100 times stiffer than in *Problem 6n.*

On the next page we present the table with the number of iterations, which we obtained by applying the our physic based scheme of partitioning, compared with the MeTiS based approach and when taking the whole matrix as the preconditioner:

We can notice that the material parameters have a higher impact on the speed of convergence in the physics-based approach. This could be expected, due to the fact that the when changing the values, we automatically change the condition number of all blocks composed from this affected material. If we increase the stiffness, we deal with matrices, which are harder

	Project 6	Project 6n	Project 6n1	Project 6n2	Project 6n3	Project 6n4
Coefficient Matrix	53	61	77	87	56	50
MeTiS based	57	105	154	195	96	81
Physic based	15	30	19	24	41	61

to factorize and which become more numerically unstable. Also the fact, that the stiffness level of the changed material started to get close to the value of the neighbouring layer may explain the deterioration of the speed. When the layers become closer in stiffness, it is more likely that we would like to keep the connection between them, since they are becoming being one rigid object. We may also notice, that the MeTiS approach is starting to work better, when the layers are starting to be less recognizable within theirs stiffness specification. Again, we could expect that, because the values in the matrix are getting closer, the connectivity of the Degrees of Freedom starts to have the bigger affect in the proper partitioning. And the MeTiS based approach is optimized to deal with this problem.
We may also see, that the size of the problem isn't a big factor in our method. It has an influence, but it is of a much smaller factor then the variation of the stiffness parameter.

Remark We would like to mention the following thing about the Picos solver. The subdomains are used in this software are not allowed to share DoF with the other subdomains. Therefore we had to come up with an idea how to convert our approach of creating subdomains, which were created by adding up Elements. Therefore our subdomains had an natural overlap on the boundaries. However, we were not able to preserve this property. Therefore we decided to assign the all of DoF which correspond to the boundary to one of the subdomains which shares them. We did this just by simply including them in the first subdomain in which them occur. When we did this, we got a really bad result in the number of iterations. The investigation of this problem lead to the conclusion, that we need to attach the boundary DoF carefully. Because of the ordering in the subdomains, the first ones were always the ones which corresponded to the softest material type. After we corrected this and started to attach those DoF to the stiffest subdomain, we obtained the results, which were presented in this section.

We would also to remind, that the description of the Projects used in tests can be found in the Appendix.

Chapter 7

Strategy and Conclusions

7.1 Strategy

We can summarize the results which were presented, with a draft of the strategy which should be applied to an arbitrary problem. In all of the problems which we tested, we followed the scheme below.

- Read data and put it into the right format.
- Apply the region growth algorithm for the recognition of areas of the same material type.
- Create the Deflation Vectors and Deflation Matrix, by taking only the stiff areas.
- Check which of the subdomains should be split. If a one or more subdomains are much bigger then the rest, in the sense of numbers of degrees of freedom which create them, we choose mark them as the ones to split.
- Split the chosen domains with MeTiS.
- Build the block preconditioner according to the Additive Schwarz Procedure.
- Use the Deflated Preconditoned Conjugate Gradient Method as the iterative solver.

7.2 Conclusions and Future Research questions

As we could saw in the last chapter of the *Numerical Test and Experiments*, the approach proposed in this Master Thesis, to solve mechanical problems with a parallel solver, based on the knowledge of the physical structure of the modelled project leads to astonishing results. With this method, we were not only able to achieve consistent results while testing. With the inclusion of the information about the problem we were able to get a faster convergence, we were able to get a faster convergence with a parallel solver, then with the singular one. This is a result which were not expected, when we started this project. We may only predict, that the further development and investigation of the direction which we had chosen at the beginning, will lead to a full understanding of the marvel achievement in which we were able to take a role.

The second conclusion which we may state based on the several experiments perform within

this project is the following one: The use of the Deflation technique, and especially the Rigid Body Modes approach, does help in solving the linear systems which come from the real-life applications. We notice in all of the cases that the convergence rate receives a speed up in the number of iterations needed to solve the problem.
Also based on the experiments which were done for for this project, we may allow ourselves to claim, that we do not need to use all of the subdomains for the Deflation. We can restrict only to use the ones composed of stiff materials. The inclusion of the other subdomains in the deflation vectors does not improve or deteriorate the convergence of the method. It only adds a numerical fluctuation to the solution, which is of the size which can allow us to neglect it importance due to the lack of significance in the whole process of solving the problem. Therefore we state it once again, that within the use of Deflation method we need only to use the subdomains composed of the stiff materials.

There are some questions, which were not answered and this project and which are definitely something, which should be researched. We will allow ourselves to present only some of them, which in our opinion are the most important:

1. *Partition of the big subdomains.* We can distinguish at least three approach to split the big areas of one material type, i.e. based on Elements, Degrees of Freedom or based on nodes. Due to lack of time, we were not able to investigate this question. This is a problem of a great matter, cause we have seen, that the a reckless split of a subdomain may lead to a great deterioration in the convergence speed. Also within this Master Thesis we limited ourselves only to use MeTiS software to perform the actual split. Therefore we were not able to see precisely, how does the partition is being made and although the mentioned software allows to use weightening for the vertices and edges, we were not using this.

2. *Transfer of the scheme.* There is a natural question about the use of the method which we develop into other branches of science. Does the inclusion of the physics of the modelled phenomena is an inevitably approach in the area of solvers of linear systems? And to be more concrete, does the method which we used can be applied to other problems without any major changes?

3. *The impact of overlap.* The Habanera solver does not allows to have any overlap when specifying the subdomains. Instead, the overlap is then added, in order go get better corrections of each subdomains. However, the addition is done only with one layer of nodes. So naturally we could ask what would happen if we would add more layers into it. Or if we would allow to create subdomains which could share the degrees of freedom on the border where they stick to.

4. *The Region Growth Algorithm.* The presented algorithm to recognize structures of the same material type, was presented in three variants. However each of them was based more or less on the same approach. Hoverer there may be a different solution to this. Also the implementation of this algorithm was done sequential, why it could be done (At least for different material types) parallel, due to the fact that different materials do not share data with each other on the recognition in one of them does not affect other ones.

Chapter 8

Appendix

8.1 Test Problems

In this section we will describe test problems, which were used as a starting point in finding the best solver for the linear system.

Problem 3: Description

The following diagram is a graphical representation of one of the sample problems given by Plaxis for numerical experiments. From now on we shall will refer to it as *Problem 3*.

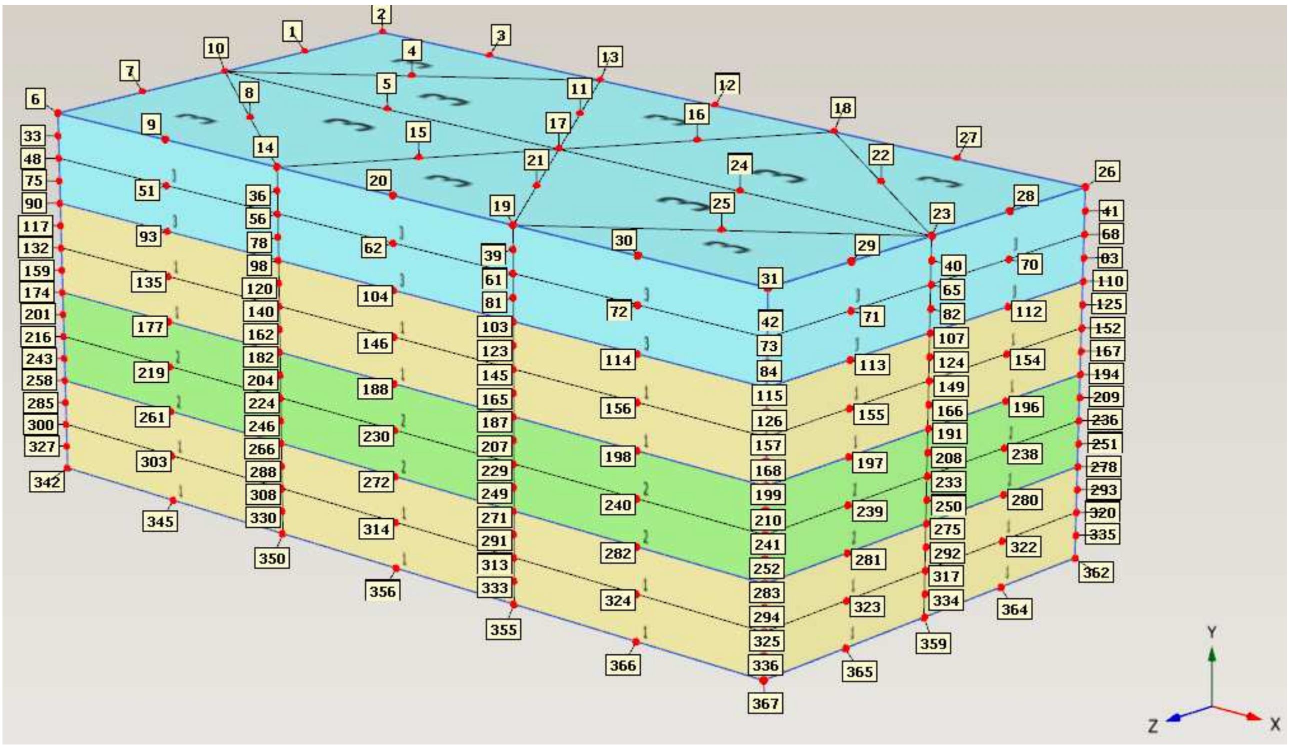

Figure 8.1: Problem 3

From the picture of Problem 3, we can clearly see that the shape of the problem is regular. Each layer consist of the same number of elements, which are regularly distributed . The color of an element is used to distinguish the type of soil, i.e. green is a stiff soil, yellow is a soft soil and bright blue is a soil which cannot change its volumes size.

# of Nonzeros	50483
# of DoF	705
# of nodes	367
# of Elements	80
# of distinguishable layers	4
# of Material types	3

Table 8.1: Information about the problem

Problem 4: Description

The following diagram is a graphical representation of one of the sample problems given by Plaxis for numerical experiments. From now on we shall will refer to it as *Problem 4*.

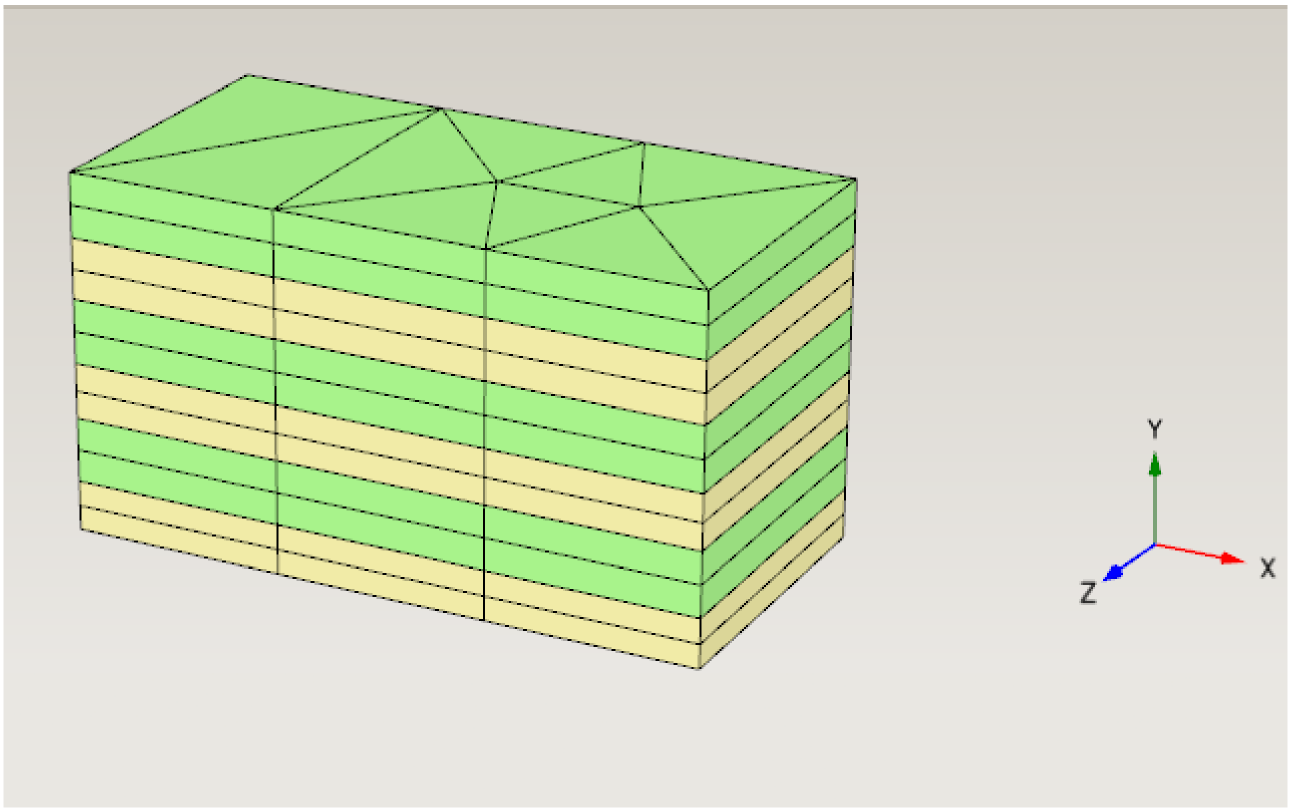

Figure 8.2: Problem 4

Problem 4 consists of 6 regular layers placed one on top of each other. In the picture we can see that they are interchangeably determined by the material type, whereas before the green color represented a stiff soil and yellow represented a soft soil.

# of Nonzeros	78929
# of DoF	1021
# of nodes	497
# of Elements	120
# of distinguishable layers	6
# of Material types	2

Table 8.2: Information about the problem

Problem 5: Description

The following diagram is a graphical representation of one of the sample problems given by Plaxis for numerical experiments. From now on we shall will refer to it as *Problem 6n1*.

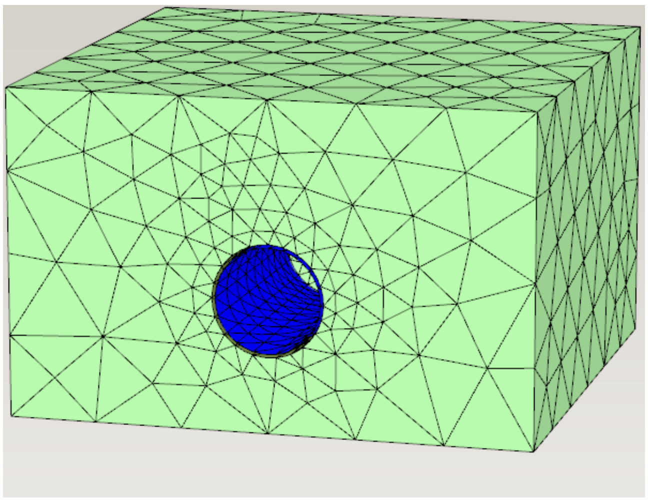

Figure 8.3: Problem 5

# of Nonzeros	2209740
# of DoF	27472
# of nodes	10778
# of Soil Elements	6872
# of Interface Elements	401
# of Plate Elements	401
# of distinguishable subdomains	3
# of soil types	1

Table 8.3: Information about the problem

Project 5 was a one of our goals at the beginning. We really wanted to test how does the chosen scheme of solving problems is working with the projects in which we are dealing with a tunnel embedded in a large area of soil. In this model we can distinguish a different type of Elements, i.e. the Interface Elements and Plate Elements. In the preconditioning, we treat them as they would stiff soil material types.

Problem 6: Description

The following diagram is a graphical representation of one of the sample problems given by Plaxis for numerical experiments. From now on we shall will refer to it as *Problem 6*.

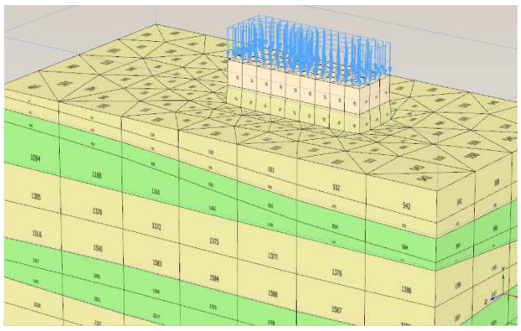

Figure 8.4: Problem 6 without two top layers of "yellow" elements

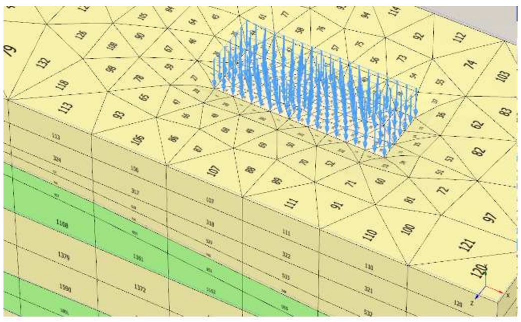

Figure 8.5: Problem 6

# of Nonzeros	1199164
# of DoF	10426
# of nodes	3970
# of Elements	1351
# of distinguishable layers	6
# of Material types	3

Table 8.4: Information about the problem

Problem 6 is a variation of Problem 4. The purpose of this problem is to see if the test results performed on Problem 4 are comparable to the results for on a degenerated case, where the layers have different sizes, and are not so regular as before. We can see that in the top layer, there is a sublayer of concrete material. It was also the first problem, where we could not set the subdomains by ourselves without having additional information. This leads us to implementation of an Region Growing algorithm within the whole program in order to find the various layers automatically.

Problem 6n1 and Problem 6n2: Description

The following diagrams are a graphical representation of two of the sample problems given by Plaxis for numerical experiments. From now on we shall will refer to them as *Problem 6n1* and *Problem 6n1*.

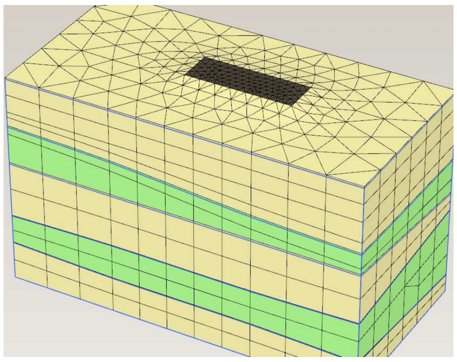

Figure 8.6: Problem 6n1

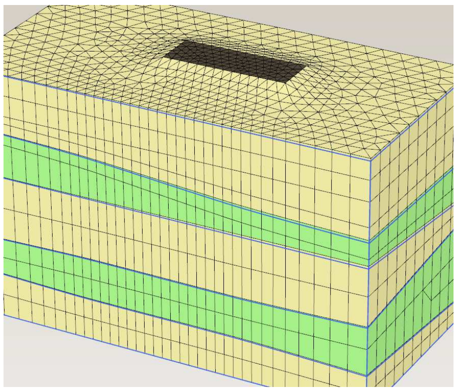

Figure 8.7: Problem 6n2

	6n1	6n2
# of Nonzeros	4981645	13215378
# of DoF	41185	108838
# of nodes	15110	39756
# of Elements	5415	14325
# of distinguishable layers	6	3
# of Material types	3	3

Table 8.5: Information about the problem

Problem 6n1 and Problem 6n2 are exaclty the same as Problem 6, but with bigger meshes. Problem 6n1 is almost 4 time bigger in the sense of DoF and Problem 6n2 10 times bigger. We created those projects, to see how much impact has the size of the problem on the solution process.

Problem 6n and 6n3 and 6n4: Description

The following diagram is a graphical representation of one of the sample problems given by Plaxis for numerical experiments. From now on we shall will refer to it as *Problem 6n*.

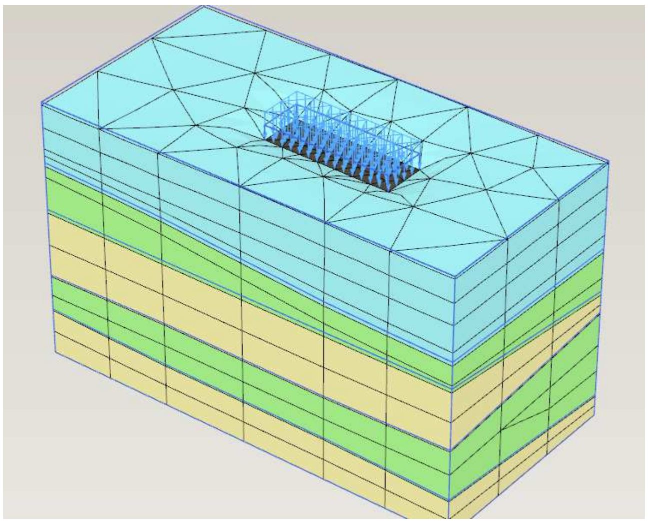

Figure 8.8: Problem 6n

# of Nonzeros	1199514
# of DoF	10426
# of nodes	3970
# of Elements	1351
# of distinguishable layers	6
# of Material types	4

Table 8.6: Information about the problem

Problem 6n is a variation of Problem 6. The material from which the top layer was created got changed, to check the impact of this altering on the time of solution. The result of this experiment lead as to an analysis of the dependency between the value of the stiffness of the material on the number of iteration needed. We did by creating 2 more examples with larger stifness values. Project 6n3 is Project 6n with a 10 times stiffer blue layer. Project 6n4 is Project 6n with a 100 times stiffer blue layer.

Problem 7: Description

The following diagram is a graphical representation of one of the sample problems given by Plaxis for numerical experiments. From now on we shall will refer to it as *Problem 7*.

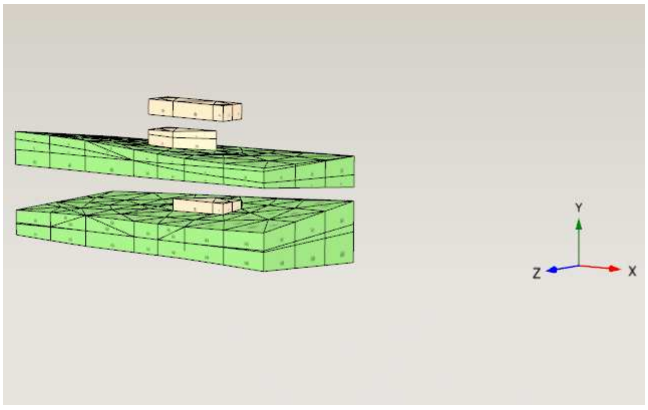

Figure 8.9: Problem 7 without soft layers

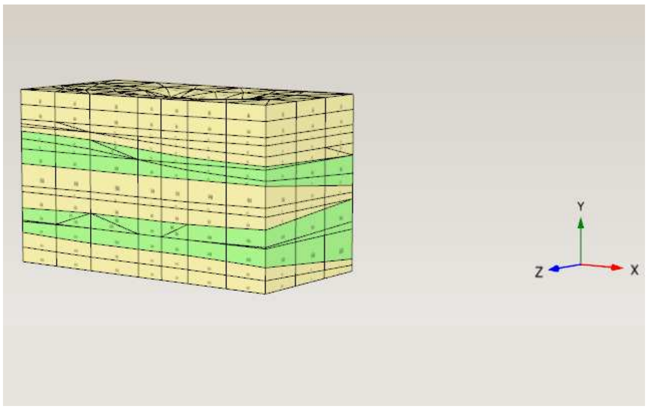

Figure 8.10: Problem 7

# of Nonzeros	1139552
# of DoF	10460
# of nodes	4022
# of Elements	1389
# of distinguishable layers	8
# of Material types	3

Table 8.7: Information about the problem

Problem 7 is another generalization of Problem 4. It was created to confirm that the Region Growing algorithm is working correctly. In the picture, we can see that we have added 2 more concrete blocks inside the structure.

Problem 8a: Description

The following diagram is a graphical representation of one of the sample problems given by Plaxis for numerical experiments. From now on we shall will refer to it as *Problem 8a*.

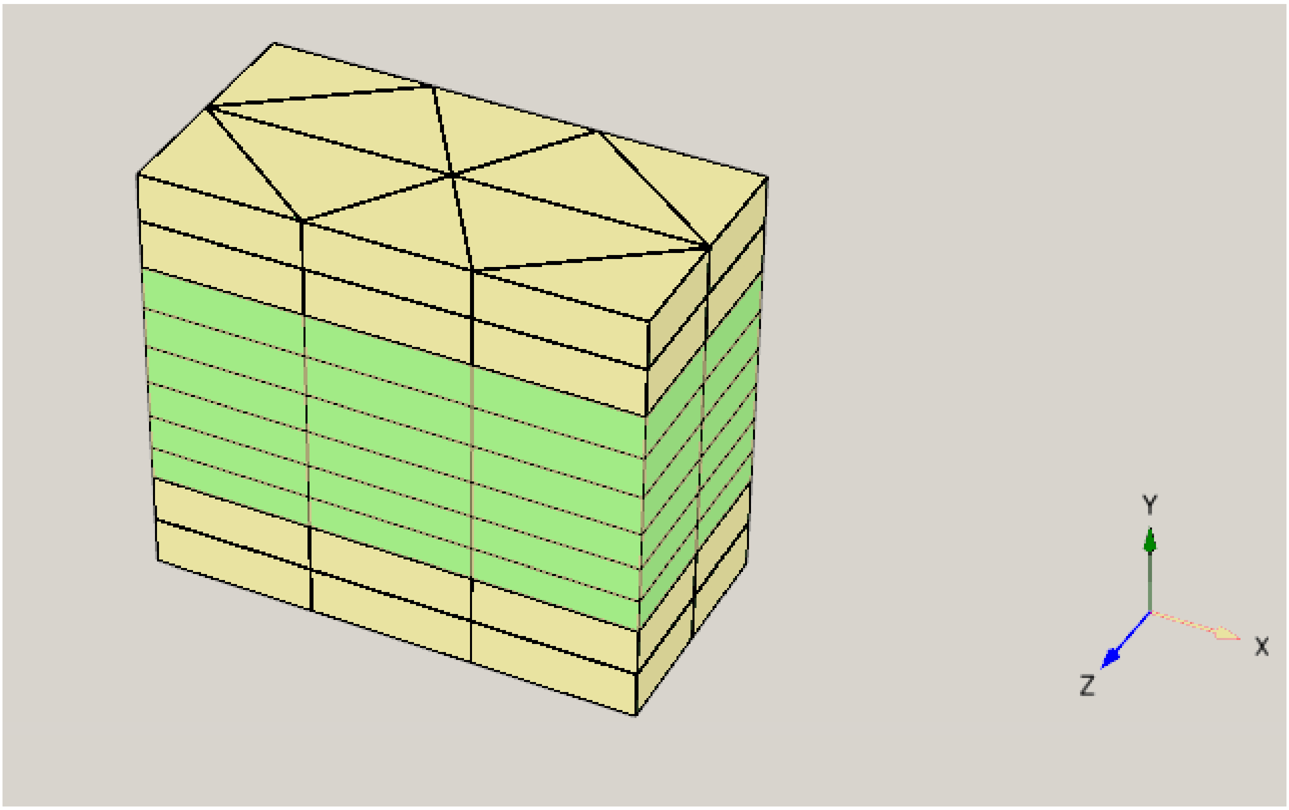

Figure 8.11: Problem 8a

# of Nonzeros	64159
# of DoF	881
# of nodes	451
# of Elements	100
# of distinguishable layers	3
# of Material types	2

Table 8.8: Information about the problem

Problem 8a is a problem that was especially created to investigate the question what is the effect of splitting bigger sublayers of the same material type. We can see that the middle layer is three times bigger than the other two. In this project, the stiff layer is inside.

Problem 8b: Description

The following diagram is a graphical representation of one of the sample problems given by Plaxis for numerical experiments. From now on we shall will refer to it as *Problem 8b*.

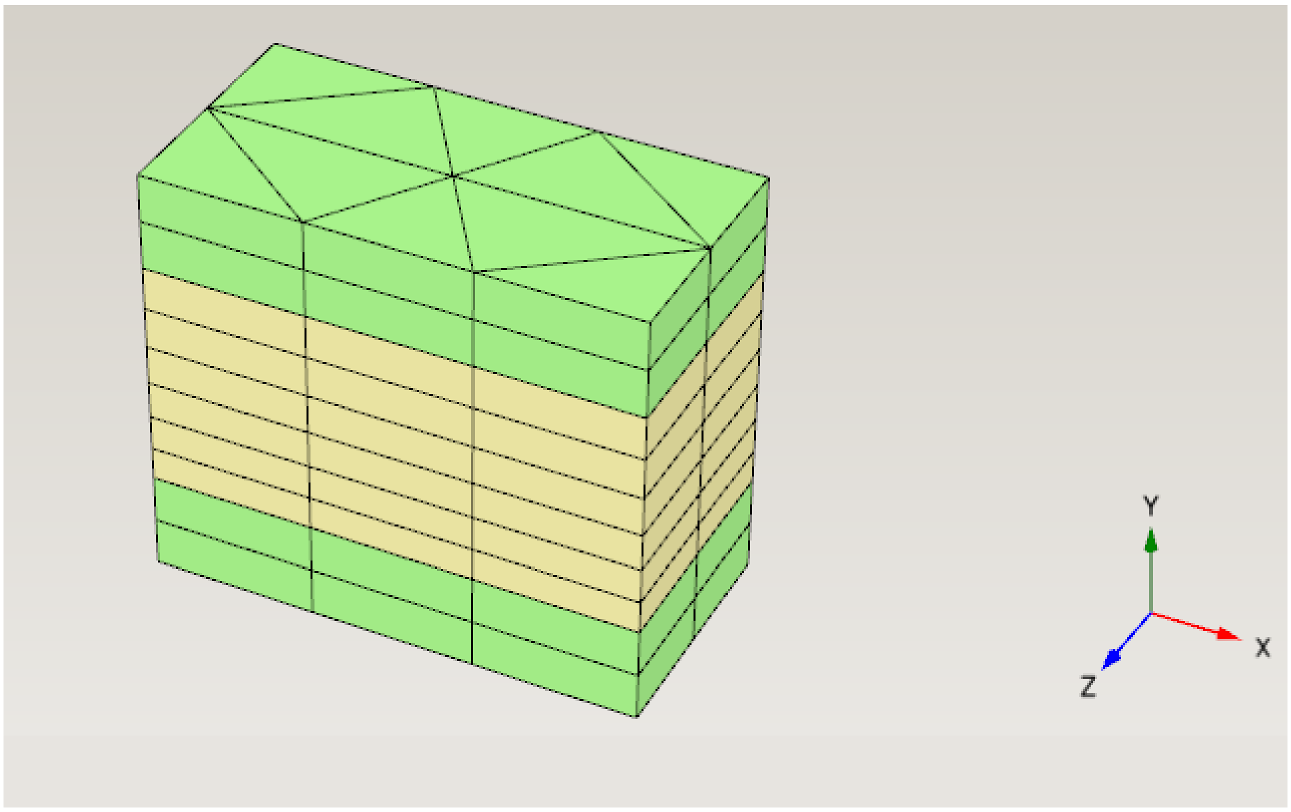

Figure 8.12: Problem 8b

# of Nonzeros	64153
# of DoF	881
# of nodes	451
# of Elements	100
# of distinguishable layers	3
# of Material types	2

Table 8.9: Information about the problem

Problem 8b is only a slight modification of Problem 8a. The only difference is the change in the material types. In this project, the stiff layers are outside.

Problem 9: Description

The following diagram is a graphical representation of one of the sample problems given by Plaxis for numerical experiments. From now on we shall will refer to it as *Problem 9*.

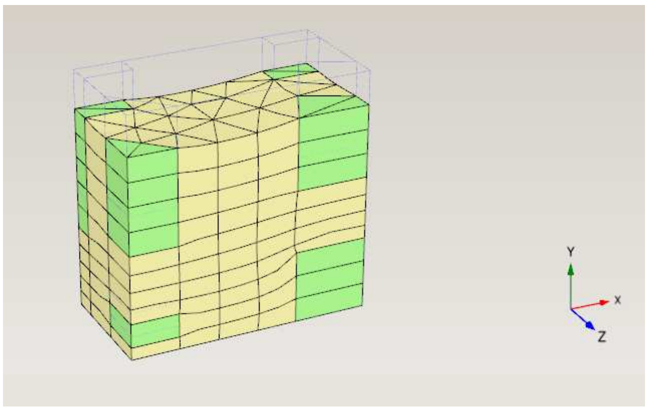

Figure 8.13: Problem 9

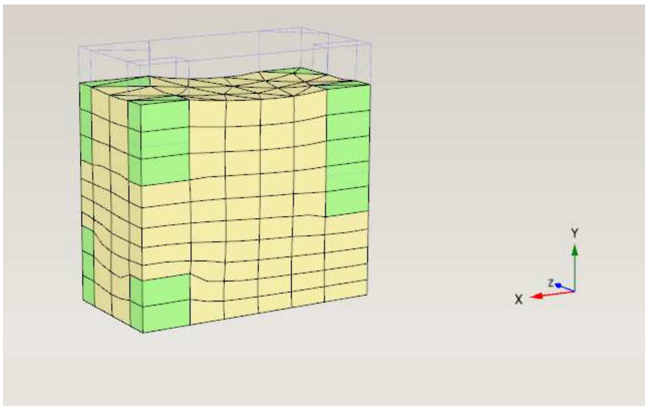

Figure 8.14: Problem 9 from a different angle

# of Nonzeros	296807
# of DoF	3101
# of nodes	1335
# of Elements	380
# of distinguishable layers	8
# of Material types	2

Table 8.10: Information about the problem

Problem 9 has two purposes. One of them is to check the Region Growing algorithm, as a case where there are many small regions. The second one is to be a variation of the Problem 8a and 8b. We can see that we are dealing here with one area of the same material type, which covers almost the whole project.

Problem 11: Description

The following diagram is a graphical representation of one of the sample problems given by Plaxis for numerical experiments. From now on we shall will refer to it as *Problem 11*.

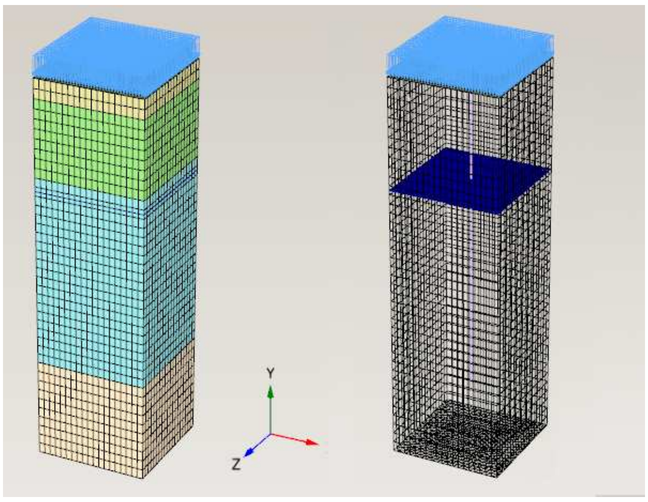

Figure 8.15: Problem 11

# of Nonzeros	19163035
# of DoF	156039
# of nodes	54187
# of Soil Elements	19926
# of Plate Elements	486
# of Embedded Pile Elements	33
# of Beam Elements	33
# of distinguishable layers	7
# of Material types	4

Table 8.11: Information about the problem

This Project was used as one of the validation of the Habanera Solver and our partitioning scheme. It is one of the typical problems, which could be created by a user of the Plaxis software. We have here in it several soil material types, plate elements, beam elements, embedded pile elements.

8.2 Region Growth Algorithms Implementations

Below we present two examples of Region Growth algorithm implemetations in Matlab.

8.2.1 Region Growth algorithm ver. 1.1

```
label = 0;

soil_type_vector = unique(Element_type);

soil_type_number = length(soil_type_vector);
for soil_type=1:soil_type_number;
    Type_Position{soil_type} = find(Element_type == soil_type_vector(soil_type));
end;
--------------------------------------------------------------------------------
% Main loop
for soil_type=1:soil_type_number;


disp(sprintf('Soil type = %d',soil_type));
    if isempty(Type_Position{soil_type})
        continue;
    end;

        label =label +1;
        Buf_Position = Type_Position{soil_type}(1);

        [Infected not_important_variable] = find(Elements{Buf_Position});

        S{label}(1) = Buf_Position;

        disp(sprintf('Label = %d',label));

        % S{label} - vector corresponding to the subdomain


        i = 1;
        for j=setdiff(Type_Position{soil_type},Buf_Position)

            [was_infected Infected] =  Infect(Infected,Elements{j});
            if was_infected
                i=i+1;
                S{label}(i) = j;
            end;
        end;


        if length( S{label}) > 1
```

```
            to_infect = i;
            while  to_infect && ~isempty(setdiff(Type_Position{soil_type},S{label}));
                to_infect = i;
                for j=setdiff(Type_Position{soil_type},S{label});

                    [was_infected Infected ] = Infect(Infected,Elements{j});
                    if was_infected
                       i=i+1;
                       S{label}(i) = j;
                    end;
                end;
                if (to_infect - i) == 0
                    to_infect = 0;
                end;

             end;
        end;

        Buf_Type_Position = setdiff(Type_Position{soil_type},S{label});
--------------------------------------------------------------------------------
%% The rest of elements from the same subdomain

        while (1 - isempty(Buf_Type_Position));

            label = label +1;
            disp(sprintf('Label = %d',label));

            Buf_Position = Buf_Type_Position(1);
            [Infected  not_important_variable]= find(Elements{Buf_Position});

            S{label}(1) = Buf_Position;

            i = 1;
            for j=setdiff(Buf_Type_Position,Buf_Position);

                [was_infected Infected] =  Infect(Infected,Elements{j});
                if was_infected
                    i=i+1;
                    S{label}(i) = j;
                end;
            end;

            if length( S{label}) > 1
                to_infect = i;

                while  to_infect && ~isempty(intersect(Buf_Type_Position,S{label}));
                     to_infect = i;
                     for j=setdiff(Type_Position{soil_type},S{label});
```

```
                        [was_infected Infected ] = Infect(Infected,Elements{j});
                        if was_infected
                            i=i+1;
                            S{label}(i) = j;
                        end;
                    end;
                    if (to_infect - i) == 0
                        to_infect = 0;
                    end;

                end;
            end;

            Buf_Type_Position = setdiff(Buf_Type_Position,S{label});

        end;


end;
--------------------------------------------------------------------------------
number_of_blocks = label;
```

```
function [Result Infected] = Infect(Infected,Element)

    [i not_important_variable]= find(Element);

    if isempty(intersect(Infected,i))

        Result = 0;
    else
        Result =1;
        Infected = union(Infected,i);
    end;
end
```

8.2.2 Region Growth algorithm ver. 2.0

```
label = 0;

how_many_nodes_in_a_element = 4;
soil_type_vector = unique(Element_type);
soil_type_number = length(soil_type_vector);
for soil_type=1:soil_type_number;
    Type_Position{soil_type} = find(Element_type == soil_type_vector(soil_type));
end;

--------------------------------------------------------------------------------
%% Main Loop

for soil_type=1:soil_type_number;


    disp(sprintf('Soil type = %d',soil_type));

    Buf_Type_Position = Type_Position{soil_type};

    if isempty(Buf_Type_Position)
        continue;
    end;

        label =label +1;

        Buf_Position = Buf_Type_Position(1);

        S{label}(1) = Buf_Position;
        S_info(label) = soil_type; % We keep here information about
                                   % the Subdomain soil type
        disp(sprintf('Label = %d',label));

        % S{label} - vector corresponding to the subdomain

       Check = 1; % Variable use to say, if we need to make another
                  % instance of the loop
       i = 1;
       while Check
           New_Neighbors = intersect(Buf_Type_Position,nonzeros( ..
           Neighbors_storage(S{label}(i),2:how_many_nodes_in_a_element+1) ));

           New_Neighbors = unique(New_Neighbors); %Sorting and droping multiple values
           New_Neighbors = setdiff(New_Neighbors,S{label});

           Check = length(New_Neighbors);

           if Check
```

```
                S{label} = [S{label}  New_Neighbors];
                i = i(end)  +  (1:Check)  ;
            end;

        end;

        Buf_Type_Position = setdiff(Type_Position{soil_type},S{label});
--------------------------------------------------------------------------------
%% Rest of Elements from the same subdomain

         while (1 - isempty(Buf_Type_Position));

             label = label +1;
             disp(sprintf('Label = %d',label));

             Buf_Position = Buf_Type_Position(1);
             S{label}(1) = Buf_Position;
             S_info(label) = soil_type;


             Check = 1;
             i = 1;
             while Check

                 New_Neighbors = intersect(Buf_Type_Position,nonzeros( ..
                 Neighbors_storage(S{label}(i),2:how_many_nodes_in_a_element+1) ));
                 New_Neighbors = unique(New_Neighbors);
                 New_Neighbors = setdiff(New_Neighbors,S{label});

                 Check = length(New_Neighbors);

                 if Check
                     S{label} = [S{label}   New_Neighbors];
                     i = i(end)  +  (1:Check)  ;

                 end;
             end;

             Buf_Type_Position = setdiff(Buf_Type_Position,S{label});

        end;


end;
--------------------------------------------------------------------------------
number_of_blocks = label;
```

8.3 Logs from test with Habanera solver

Problem 5

```
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
1 subdomain

Setting up the preconditioner ... 4.71094 sec (4.66 CPU sec).
Executing the solver ........... 3.96191 sec (3.94 CPU sec).
Solver info:
  threads    ... 1
  iterations ... 125
  residual   ... 8.581032e-06
Relative error = 8.581032e-06
Memory usage:
  matrix ....... 1.342327e+07
  precon ....... 2.465375e+07
  solver ....... 3.868058e+07

--------------------------------------------------------------------------------

3 subdomains:

     Metis on the nodes

subdomain 0 contains 11023 DOFs subdomain 1 contains 11240 DOFs
subdomain 2 contains 11504 DOFs
Setting up the preconditioner ... 2.82959 sec (5.16 CPU sec).
Executing the solver ........... 2.58849 sec (4.26 CPU sec).
Solver info:
  threads    ... 3
  iterations ... 70
  residual   ... 8.791751e-06
Relative error = 8.791751e-06
Memory usage:
  matrix ....... 1.588810e+07
  precon ....... 3.298598e+07
  solver ....... 2.593306e+07

 Physics Based Domains

subdomain 0 contains 22666 DOFs subdomain 1 contains 8840 DOFs
subdomain 2 contains 11460 DOFs
Setting up the preconditioner ... 3.50673 sec (5.35 CPU sec).
Executing the solver ........... 2.31269 sec (3.5 CPU sec).
Solver info:
  threads    ... 3
  iterations ... 46
  residual   ... 7.963149e-06
Relative error = 7.963149e-06
Memory usage:
  matrix ....... 2.018612e+07
  precon ....... 4.197390e+07
  solver ....... 1.649894e+07



--------------------------------------------------------------------------------

4 subdomains:

  Metis on the nodes Domains

subdomain 0 contains 9692 DOFs subdomain 1 contains 8546 DOFs
subdomain 2 contains 9035 DOFs subdomain 3 contains 8861 DOFs
Setting up the preconditioner ... 2.44054 sec (4.57 CPU sec).
Executing the solver ........... 3.08434 sec (5.47 CPU sec).
Solver info:
  threads    ... 4
  iterations ... 79
  residual   ... 6.547770e-06
Relative error = 6.547770e-06
Memory usage:
  matrix ....... 1.684483e+07
  precon ....... 3.520984e+07
  solver ....... 2.775091e+07

  Physics Based Domains

subdomain 0 contains 12159 DOFs
subdomain 1 contains 13756 DOFs
subdomain 2 contains 8840 DOFs
subdomain 3 contains 11460 DOFs
Setting up the preconditioner ... 3.11325 sec (5.71 CPU sec).
Executing the solver ........... 2.40156 sec (4.24 CPU sec).
Solver info:
  threads    ... 4
  iterations ... 49
  residual   ... 8.176662e-06
Relative error = 8.176662e-06
Memory usage:
  matrix ....... 2.143604e+07
  precon ....... 4.454828e+07
  solver ....... 3.549312e+07
```

```
--------------------------------------------------------------------------------

5 subdomains:

  Metis on the nodes Domains

subdomain 0 contains 7128 DOFs subdomain 1 contains 8172 DOFs
subdomain 2 contains 7558 DOFs subdomain 3 contains 7739 DOFs
subdomain 4 contains 7327 DOFs
Setting up the preconditioner ... 2.58386 sec (4.79 CPU sec).
Executing the solver ........... 3.3323 sec (5.75 CPU sec).
Solver info:
  threads    ... 5
  iterations ... 80
  residual   ... 8.635928e-06
Relative error = 8.635927e-06
Memory usage:
  matrix ....... 1.755151e+07
  precon ....... 3.658103e+07
  solver ....... 2.912563e+07

  Physics Based Domains

subdomain 0 contains 8435 DOFs  subdomain 1 contains 9263 DOFs
subdomain 2 contains 11643 DOFs subdomain 3 contains 8840 DOFs
subdomain 4 contains 11460 DOFs
Setting up the preconditioner ... 3.21581 sec (5.99 CPU sec).
Executing the solver ........... 2.71131 sec (4.64 CPU sec).
Solver info:
  threads    ... 5
  iterations ... 51
  residual   ... 7.664127e-06
Relative error = 7.664127e-06
  matrix ....... 2.274922e+07
  precon ....... 4.750062e+07
  solver ....... 3.494726e+07
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

Problem 11

```
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

1 subdomain

subdomain 0 contains 156039 DOFs
Setting up the preconditioner ... 57.6367 sec (54.59 CPU sec).
```

```
Executing the solver ........... 32.6221 sec (32.18 CPU sec).
Solver info:
  threads    ... 1
  iterations ... 123
  residual   ... 8.750293e-06
Relative error = 8.750294e-06
Memory usage:
  matrix ....... 1.159144e+08
  precon ....... 2.045206e+08
  solver ....... 1.597839e+08
--------------------------------------------------------------------------------

7 subdomains

  Metis on the nodes Domains

subdomain 0 contains 28319 DOFs  subdomain 1 contains 29331 DOFs
subdomain 2 contains 29373 DOFs  subdomain 3 contains 26503 DOFs
subdomain 4 contains 26307 DOFs  subdomain 5 contains 28858 DOFs
subdomain 6 contains 28636 DOFs
Setting up the preconditioner ... 28.708 sec (54.66 CPU sec).
Executing the solver ........... 37.8795 sec (70.92 CPU sec).
Solver info:
  threads    ... 7
  iterations ... 108
  residual   ... 8.754482e-06
Relative error = 8.754490e-06
Memory usage:
  matrix ....... 1.390505e+08
  precon ....... 2.731998e+08
  solver ....... 2.146918e+08

  Physics Based Domains

subdomain 0 contains 32735 DOFs  subdomain 1 contains 84829 DOFs
subdomain 2 contains 17206 DOFs  subdomain 3 contains 40931 DOFs
subdomain 4 contains 9764 DOFs   subdomain 5 contains 1311 DOFs
subdomain 6 contains 5979 DOFs
Setting up the preconditioner ... 30.5872 sec (47.8 CPU sec).
Executing the solver ........... 8.55941 sec (14.06 CPU sec).
Solver info:
  threads    ... 7
  iterations ... 30
  residual   ... 8.343765e-06
Relative error = 8.343758e-06
Memory usage:
  matrix ....... 1.354006e+08
  precon ....... 2.708914e+08
  solver ....... 1.048587e+08
--------------------------------------------------------------------------------

8 subdomains

    Metis on the nodes Domains

subdomain 0 contains 25003 DOFs  subdomain 1 contains 24403 DOFs
subdomain 2 contains 24087 DOFs  subdomain 3 contains 26795 DOFs
subdomain 4 contains 25810 DOFs  subdomain 5 contains 24433 DOFs
subdomain 6 contains 27102 DOFs  subdomain 7 contains 26429 DOFs
Setting up the preconditioner ... 32.9677 sec (60.79 CPU sec).
Executing the solver ........... 49.1155 sec (91.71 CPU sec).
Solver info:
  threads    ... 8
  iterations ... 106
  residual   ... 8.939803e-06
Relative error = 8.939799e-06
Memory usage:
  matrix ....... 1.436574e+08
  precon ....... 2.870635e+08
  solver ....... 2.220195e+08

  Physics Based Domains

subdomain 0 contains 32735 DOFs  subdomain 1 contains 43837 DOFs
subdomain 2 contains 17206 DOFs  subdomain 3 contains 39040 DOFs
subdomain 4 contains 53376 DOFs  subdomain 5 contains 9764 DOFs
subdomain 6 contains 1311 DOFs   subdomain 7 contains 5979 DOFs
Setting up the preconditioner ... 46.2319 sec (84.54 CPU sec).
Executing the solver ........... 16.8142 sec (30.41 CPU sec).
Solver info:
  threads    ... 8
  iterations ... 35
  residual   ... 7.282611e-06
Relative error = 7.282613e-06
Memory usage:
  matrix ....... 1.415436e+08
  precon ....... 2.837366e+08
  solver ....... 1.105669e+08
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

Problem 6n

```
1 subdomain

subdomain 0 contains 10426 DOFs
```

```
Setting up the preconditioner ... 3.09262 sec (3.04 CPU sec).
Executing the solver ........... 0.859629 sec (0.86 CPU sec).
Solver info:
  threads    ... 1
  iterations ... 61
  residual   ... 2.372472e-06
Relative error = 2.372475e-06
Memory usage:
  matrix ....... 7.258560e+06
  precon ....... 1.461126e+07
  solver ....... 5.671744e+06
--------------------------------------------------------------------------------

6 subdomains

    Metis on the nodes Domains

subdomain 0 contains 2793 DOFs subdomain 1 contains 3164 DOFs
subdomain 2 contains 2802 DOFs subdomain 3 contains 2604 DOFs
subdomain 4 contains 3064 DOFs subdomain 5 contains 2585 DOFs
Setting up the preconditioner ... 1.707 sec (3 CPU sec).
Executing the solver ........... 2.38158 sec (3.93 CPU sec).
Solver info:
  threads    ... 6
  iterations ... 105
  residual   ... 7.451430e-06
Relative error = 7.451437e-06
Memory usage:
  matrix ....... 1.064609e+07
  precon ....... 2.201982e+07
  solver ....... 1.633152e+07

  Physics Based Domains

subdomain 0 contains 2419 DOFs subdomain 1 contains 1730 DOFs
subdomain 2 contains 4149 DOFs subdomain 3 contains 4194 DOFs
subdomain 4 contains 3910 DOFs subdomain 5 contains 1737 DOFs
Setting up the preconditioner ... 1.62862 sec (3.12 CPU sec).
Executing the solver ........... 0.653403 sec (1.12 CPU sec).
Solver info:
  threads    ... 6
  iterations ... 30
  residual   ... 9.199300e-06
Relative error = 9.199310e-06
Memory usage:
  matrix ....... 1.116456e+07
  precon ....... 2.305551e+07
  solver ....... 8.706720e+06
```

```
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

Project 6n1

```
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

1 subdomain

subdomain 0 contains 41185 DOFs
Setting up the preconditioner ... 14.4395 sec (14.29 CPU sec).
Executing the solver ........... 4.58893 sec (4.57 CPU sec).
Solver info:
  threads    ... 1
  iterations ... 77
  residual   ... 7.050555e-06
Relative error = 7.050524e-06
Memory usage:
  matrix ....... 3.013698e+07
  precon ....... 6.005480e+07
  solver ....... 4.480928e+07
--------------------------------------------------------------------------------

6 subdomains

    Metis on the nodes Domains

subdomain 0 contains 10471 DOFs subdomain 1 contains 9870 DOFs
subdomain 2 contains 10179 DOFs subdomain 3 contains 10055 DOFs
subdomain 4 contains 9282 DOFs subdomain 5 contains 9545 DOFs
Setting up the preconditioner ... 7.16859 sec (13.27 CPU sec).
Executing the solver ........... 14.9015 sec (26.74 CPU sec).
Solver info:
  threads    ... 6
  iterations ... 154
  residual   ... 8.477394e-06
Relative error = 8.477374e-06
Memory usage:
  matrix ....... 4.040156e+07
  precon ....... 8.345800e+07
  solver ....... 1.292588e+08

  Physics Based Domains

subdomain 0 contains 15200 DOFs subdomain 1 contains 9581 DOFs
subdomain 2 contains 6848 DOFs  subdomain 3 contains 16429 DOFs
subdomain 4 contains 16525 DOFs subdomain 5 contains 5511 DOFs
Setting up the preconditioner ... 7.58133 sec (14.49 CPU sec).
```

```
Executing the solver ........... 1.70718 sec (3.2 CPU sec).
Solver info:
  threads    ... 6
  iterations ... 19
  residual   ... 6.917198e-06
Relative error = 6.917216e-06
Memory usage:
  matrix ....... 4.529844e+07
  precon ....... 9.349035e+07
  solver ....... 3.588813e+07
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

Project 6n2

```
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

1 subdomain

subdomain 0 contains 108838 DOFs
Setting up the preconditioner ... 37.0956 sec (36.57 CPU sec).
Executing the solver ........... 15.2002 sec (15.04 CPU sec).
Solver info:
  threads    ... 1
  iterations ... 87
  residual   ... 3.414488e-06
Relative error = 3.414477e-06
Memory usage:
  matrix ....... 7.994530e+07
  precon ....... 1.603534e+08
  solver ....... 1.253814e+08

--------------------------------------------------------------------------------

6 subdomains

    Metis on the nodes Domains

subdomain 0 contains 21891 DOFs subdomain 1 contains 22962 DOFs
subdomain 2 contains 22587 DOFs subdomain 3 contains 22164 DOFs
subdomain 4 contains 21991 DOFs subdomain 5 contains 21129 DOFs
Setting up the preconditioner ... 16.1834 sec (31.63 CPU sec).
Executing the solver ........... 45.6606 sec (86.73 CPU sec).
Solver info:
  threads    ... 6
  iterations ... 195
  residual   ... 8.495129e-06
Relative error = 8.495133e-06
```

```
Memory usage:
  matrix ....... 9.341614e+07
  precon ....... 1.922354e+08
  solver ....... 2.888074e+08

  Physics Based Domains

subdomain 0 contains 42133 DOFs subdomain 1 contains 25288 DOFs
subdomain 2 contains 18074 DOFs subdomain 3 contains 43362 DOFs
subdomain 4 contains 43755 DOFs subdomain 5 contains 6105 DOFs
Setting up the preconditioner ... 20.5723 sec (38.19 CPU sec).
Executing the solver ........... 5.59561 sec (10.58 CPU sec).
Solver info:
  threads    ... 6
  iterations ... 24
  residual   ... 9.287363e-06
Relative error = 9.287366e-06
Memory usage:
  matrix ....... 1.164810e+08
  precon ....... 2.417763e+08
  solver ....... 9.150310e+07
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

Project 6n3

```
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

1 subdomain

subdomain 0 contains 10426 DOFs
Setting up the preconditioner ... 3.26235 sec (3.17 CPU sec).
Executing the solver ........... 0.816661 sec (0.78 CPU sec).
Solver info:
  threads    ... 1
  iterations ... 56
  residual   ... 6.421310e-06
Relative error = 6.421316e-06
Memory usage:
  matrix ....... 7.258560e+06
  precon ....... 1.435961e+07
  solver ....... 5.671744e+06

--------------------------------------------------------------------------------

6 subdomains

    Metis on the nodes Domains
```

```
subdomain 0 contains 2793 DOFs subdomain 1 contains 3164 DOFs
subdomain 2 contains 2802 DOFs subdomain 3 contains 2604 DOFs
subdomain 4 contains 3064 DOFs subdomain 5 contains 2585 DOFs
Setting up the preconditioner ...  1.89165 sec (2.9 CPU sec).
Executing the solver ........... 2.29658 sec (3.47 CPU sec).
Solver info:
  threads    ... 6
  iterations ... 96
  residual   ... 4.667178e-06
Relative error = 4.667179e-06
Memory usage:
  matrix ....... 1.064609e+07
  precon ....... 2.198946e+07
  solver ....... 1.633152e+07

  Physics Based Domains

subdomain 0 contains 2419 DOFs subdomain 1 contains 1730 DOFs
subdomain 2 contains 4149 DOFs subdomain 3 contains 4194 DOFs
subdomain 4 contains 3910 DOFs subdomain 5 contains 1737 DOFs
Setting up the preconditioner ... 1.83257 sec (3.09 CPU sec).
Executing the solver ........... 0.957748 sec (1.46 CPU sec).
Solver info:
  threads    ... 6
  iterations ... 41
  residual   ... 9.181167e-06
Relative error = 9.181169e-06
Memory usage:
  matrix ....... 1.116456e+07
  precon ....... 2.299729e+07
  solver ....... 8.706720e+06
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

Problem 6n4

```
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
1 subdomain

subdomain 0 contains 10426 DOFs
Setting up the preconditioner ... 3.25482 sec (3.17 CPU sec).
Executing the solver ........... 0.726382 sec (0.69 CPU sec).
Solver info:
  threads    ... 1
  iterations ... 50
  residual   ... 7.898489e-06
Relative error = 7.898487e-06
```

```
Memory usage:
  matrix ....... 7.258560e+06
  precon ....... 1.434010e+07
  solver ....... 5.671744e+06
--------------------------------------------------------------------------------

6 subdomains

    Metis on the nodes Domains

subdomain 0 contains 2793 DOFs subdomain 1 contains 3164 DOFs
subdomain 2 contains 2802 DOFs subdomain 3 contains 2604 DOFs
subdomain 4 contains 3064 DOFs subdomain 5 contains 2585 DOFs
Setting up the preconditioner ... 1.9318 sec (3.08 CPU sec).
Executing the solver ........... 1.87377 sec (2.81 CPU sec).
Solver info:
  threads    ... 6
  iterations ... 81
  residual   ... 9.379617e-06
Relative error = 9.379619e-06
Memory usage:
  matrix ....... 1.064609e+07
  precon ....... 2.174573e+07
  solver ....... 1.633152e+07

  Physics Based Domains

subdomain 0 contains 2419 DOFs subdomain 1 contains 1730 DOFs
subdomain 2 contains 4149 DOFs subdomain 3 contains 4194 DOFs
subdomain 4 contains 3910 DOFs subdomain 5 contains 1737 DOFs
Setting up the preconditioner ... 2.1284 sec (3 CPU sec).
Executing the solver ........... 1.70401 sec (1.78 CPU sec).
Solver info:
  threads    ... 6
  iterations ... 61
  residual   ... 8.256644e-06
Relative error = 8.256645e-06
Memory usage:
  matrix ....... 1.116456e+07
  precon ....... 2.315868e+07
  solver ....... 1.741344e+07
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

Bibliography

[1] P. Bjorstad, B. Smith, W. Gropp, *Domain Decomposition.* Cambridge University Press: Cambridge, 1996.

[2] A. Cegielski (Editor), *Numerical Aspects in Applied Mathematics.* ZP UZm Zielona Gora, Poland, 2005.

[3] M. R. Hestenes,E. Stiefel, *Methods of Conjugate Gradients for Solving Linear Systems.* Journal of Research of the National Bureau of Standards Vol 49, No 6.

[4] J. van Kan, A. Segal, F. Vermolen, *Numerical Methods in Scientific Computing* VSSD, Delft, The Netherlands, 2005.

[5] Plaxis B.V., *Plaxis 3D Foundation: Scientific Manual version 2*, Delft, The Netherlands

[6] Y. Saad, *Iterative Methods for Sparse Linear Systems.* SIAM, Philadelphia, PA, USA, 2000. Second edition.

[7] A. Segal, C. Vuik, *Computational Fluid Dynamics II.* Delft, The Netherlands, 2006.

[8] K.H. Tan, *Local Coupling in Domain Decomposition.* Utrecht, The Netherlands, 1995.

[9] J.M. Tang, *Two-Level Preconditioned Conjugate Gradient Methods with Applications to Bubbly Flow Problems.* Delft, The Netherlands, 2008.

[10] A. Toselli, O. Widlund, *Domain Decomposition Methods - Algorithms and Theory* Springer-Verlag Berlin Heidelberg, 2005.

[11] J. Willie, *Internship report*, Vortech Computing, Delft, The Netherlands, 2008.

[12] E. Vollebregt, *De incomplete Cholesky preconditioner en de parallellisatie van Plaxis3D via OpenMP*, Memo EV/M08.029, version 1.1, VORtech, Juni 2008.

[13] T.B Jonsthvel, M.B. van Gijzen, C.Vuik, C. Kasbergen, A. Scarpas, *Preconditioned Conjugate Gradient Method Enhanced by Deflation of Rigid Body Modes Applied to Composite Materials*, CMES, vol.47, no.2, pp.97-118, 2009

[14] Habanera (habanera.nl), *Manual for Picos*, Delft, The Netherlands, 2010.

[15] Robin J. Wilson, *Introduction to Graph Theory*, Addison Wesley; 4 edition, 2 May 1996.