Reducing Communication in AMG for Reservoir Simulation

Aggressive Coarsening and Non-Galerkin Coarse-Grid Operators

E.D. Wobbes

Master of Science Thesis

Delft University of Technology

Delft Institute for Applied Mathematics

Reducing Communication in AMG for Reservoir Simulation

Aggressive Coarsening and Non-Galerkin Coarse-Grid Operators

Master of Science Thesis

For the degree of Master of Science in Applied Mathematics at Delft University of Technology

E.D. Wobbes

August 25, 2014

Faculty of Electrical Engineering, Mathematics and Computer Science (EWI) · Delft University of Technology

Schlumberger

The work in this thesis was supported by Schlumberger Oilfield UK Plc. The cooperation is hereby gratefully acknowledged.

TU Delft Delft University of Technology

Delft University of Technology
Department of
Delft Institute for Applied Mathematics (diam)

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science (EWI) for acceptance a thesis entitled

Reducing Communication in AMG for Reservoir Simulation

by

E.D. Wobbes

in partial fulfillment of the requirements for the degree of

Master of Science Applied Mathematics

Dated: August 25, 2014

Supervisor(s):

Prof.dr.ir. C. Vuik

Dr.ir. T.B. Jönsthövel

Dr. A.A. Lukyanov

Dr. S. MacLachlan

Reader(s):

Dr. D.J.P. Lahaye

Dr. H.M. Schuttelaars

Abstract

Algebraic Multigrid (AMG) is an efficient multigrid method for solving large problems, using only the information provided by the underlying matrices. Unfortunately, on a parallel machine the performance of AMG can be negatively affected by dense communication patterns. The exchange of extensive data sets is generally caused by a high number of non-zero entries contained by the coarse grid operators. We discuss two solution strategies, that can be applied in order to improve the sparsity patterns of these operators: aggressive coarsening and the non-Galerkin method. Aggressive coarsening reduces the number of coarse grid variables by modifying the basic concepts of the standard coarsening scheme. The non-Galerkin approach is entirely different. While preserving the row sums, the non-Galerkin method removes less important entries of the coarse level operators, generated by AMG. In fact, it can be seen as an extension of the standard multigrid algorithm. We explore both techniques in the frame of reservoir simulation. We demonstrate that aggressive coarsening and non-Galerkin algorithm significantly reduce the total number of non-zero entries in the AMG hierarchy. Although aggressive coarsening shows high performance, in terms of execution time, on one processor core, it is less effective for parallel simulations. The non-Galerkin has not been tested in parallel, but its serial results are very promissing.

Table of Contents

List of Figures

List of Tables

Acknowledgements

First off all, I would like to thank my supervisors. Without their help, this project would not have been realized. I would like to express my gratitude to my daily supervisor, Dr. Tom Jönsthövel, for his patient guidance, elaborate explanations and enthusiastic encouragement. Our discussions have motivated my work and fundamentally changed my attitude about scientific research.
I am particularly grateful to Dr. Alexander Lukyanov, for his original ideas and passionate belief in monotonicity. Moreover, Alex's academic guidance and technical assistance were critical in the successful completion of this work.
I would like to thank my other supervisors, Prof. Kees Vuik and Dr. Scott MacLachlan, for the support they provided on all levels of this research project.

My gratitude is extended to Gareth Shaw, who was my manager at Schlumberger, for the guidance and input on this project.

I am grateful to to Dr. Domenico Lahaye and Dr. Henk Schuttelaars for taking time to evaluate my research work and attend my final presentation.

Special thanks to Matteo Cusini, who was an intern at Schlumberger as well. Matteo helped make my time in th UK more interesting and fun.

Finally, I take the opportunity to thank my mother, Larisa Wobbes, and my boyfriend, Mathijs Rozemuller, for their constant love, encouragement and support.

Chapter 1

Introduction

1-1 Motivation

Reservoir simulation is an integrated part of oil-reservoir management. Physical and numerical models of reservoir simulation have been evolving through several decades, expanding from simple one-dimensional single phase flow variants to complicated three-dimensional compositional variants. The contemporary models consist of millions of grid cells and describe the flow in highly heterogeneous media with extremely complex geometries. In attempt to simultaneously obtain accuracy of the solution and minimization of computational costs for current models, reservoir simulation software is continuously adapted. In this thesis, we consider a high-resolution reservoir simulator, INTERSECT. The simulator is developed by Schlumberger, in partnership with Chevron and Total, and is based on the company's previous, yet still widely used, reservoir simulation software ECLIPSE.

INTERSECT utilizes a sophisticated mechanism to model fluid flow. An indispensable part of this mechanism is a Krylov-subspace method, required for the repeated solution of large linear systems of equations, originating from the linearization of coupled mass-balance equations. These systems are associated with several unknowns, such as pressure, saturation and composition of each fluid phase. Typically, the simulation is determined by the behaviour of the elliptic (or parabolic) pressure part. For this reason, INTERSECT uses a two-stage Constrained Pressure Residual (CPR) method as a preconitioner for the Krylov solver. The main idea behind the CPR preconditioner is to decouple the equations and solve the pressure subsystem separately from the hyperbolic non-pressure subsystem.

Multigrid methods solve elliptic linear equations in a very efficient manner. Within INTERSECT, the algebraic approach is used to obtain the pressure solution. There are several reasons for this choice. On the one hand, algebraic multigrid (AMG) method is well-suited for the porous-media-flow simulations, because it is able to deal with complex structures, anisotropies and discontinuous or varying coefficients. On the other hand, AMG does not require geometric information and, hence, can be easily integrated into existing simulation code. Unfortunately, experiments have shown that for large models AMG does not perform as efficient as expected due to dense communication patterns.

In this thesis, we aim to resolve the communication issue in AMG. The relevant previous work is presented in [37, 11, 14, 40]. In this project, the focus is on two solution strategies: aggressive coarsening and the non-Galerkin method. Since the exchange of large data sets is caused by a high number of non-zero entires in the coarse grid operators, both approaches aim to improve the sparsity patterns of these operators.

Aggressive coarsening was introduced in 1996 in [27]. Its application to AMG is discussed in [37, 12], whereby in [37] aggressive coarsening is recommended in order to decrease the ratio of the number of non-zero entries on a coarser level to that on a finer level. The aggressive coarsening approach modifies the concepts and heuristics, required by the standard coarsening scheme, and should be seen as its substitute.

In [2, 42], the non-Galerkin method is explored in settings where multigrid methods use geometric information to construct the sparsity pattern and choose matrix coefficients. The description of the method for the purely algebraic approach, applied to symmetric positive definite matrices, is provided in [15]. The non-Galerkin algorithm removes less important entries of the coarse grid matrices, produced by AMG, while preserving the row sums.

1-2 Scope

In this work, an extensive overview of the simulation process in INTERSECT is presented. It describes the formulation of the non-linear system, Newton-Raphson method, FGMRES and CPR preconditioner. After narrowing the focus to AMG, the techniques that can be used to enhance its performance are discussed.
To investigate the aggressive coarsening strategy, INTERSECT is interconnected with SAMG (Algebraic Multigrid Methods for Systems), a commercial software package developed at Fraunhofer SCAI. More precisely, the complete AMG cycle is replaced by the black-box AMG solver, provided by SAMG. Both serial and parallel versions of this approach are available.
The non-Galerkin method is first prototyped in MATLAB and is subsequently implemented in C++ as an extension of the AMG algorithm within INTERSECT. The non-Galerkin code is serial. Based on the observed properties of the non-Galerkin operators, we propose a promising modification of the algorithm.

1-3 Outline

The thesis is organized as follows:

Chapter 2: INTERSECT. In Chapter 2, the framework of INTERSECT is explored. After a schematic representation is provided, each individual component is discussed in more detail.

Chapter 3: AMG. In Chapter 3, we make a general characterization of AMG and give an overview of serial and parallel coarsening strategies. In addition, the basic properties of AMG within INTERSECT are summarized.

Chapter 4: Problem Description and Solution Strategies. The data communication problem is stated in Chapter 4. We introduce the ideas behind the solution strategies (aggressive coarsening and the non-Galerkin method) and present the possible realizations.

Chapter 5: Results. In this chapter, the results are discussed in terms of the operator and grid complexities, number of non-linear and linear iterations, and linear solver and total execution time.

Chapter 6: Conclusions. We conclude this thesis and provide suggestions for future research in Chapter 6.

Chapter 2

INTERSECT

2-1 Simulator Framework

INTERSECT models the flow of fluids such as water and oil, through the porous media comprising the reservoir. The flow is described in terms of the pressure, saturation and composition of each fluid phase.
In this section, we outline the structure of INTERSECT. The upcoming chapters describe each simulation phase in more detail.

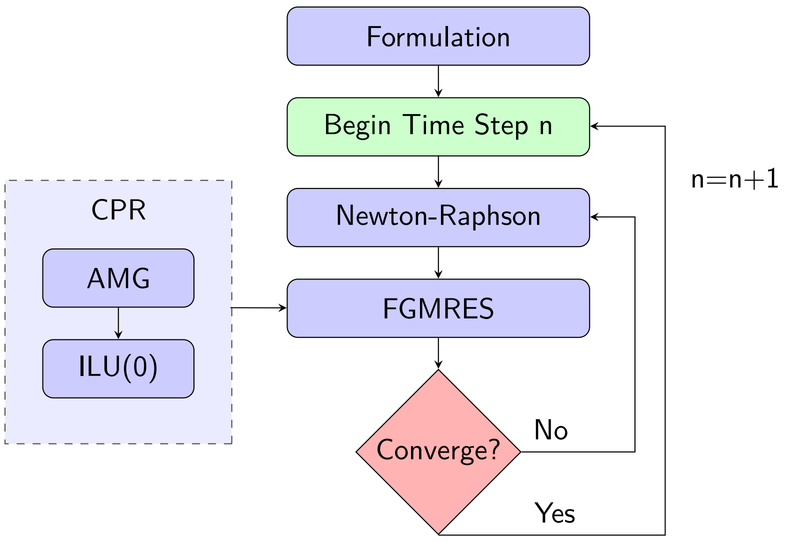


Figure 2.1: INTERSECT overview.

As shown in Figure 2.1, the first step in the simulation is the formulation of the equations describing the physical process. These equations are based on the conservation of mass in the reservoir, Darcy's law and the fact that the fluid phases are in thermodynamic equilibrium. In

Section 2-2 it is explained that the constructed system of equations is non-linear and coupled. After the formulation stage and given the initial guess, the solution is progressed over time. For each time step, the system is linearized by the Newton-Raphson method.
The resulting linear system is solved using FGMRES preconditioned by the Constrained Pressure Residual algorithm. The CPR method consists of two stages. Previous to the first stage, the system is decoupled so that the elliptic (or parabolic) pressure part can be solved separately from the remaining hyperbolic part of the discrete operator. In the first stage, the decoupled linear system is restricted to the elliptic (or parabolic) part and the pressure is approximated by applying an FGMRES-accelerated AMG preconditioner. The solution found by AMG is used as the initial guess for pressure in the second stage of the CPR scheme, which solves the complete system using ILU(0). In Section 2-4 the CPR method is described in more detail.
When the outer FGMRES iteration is completed, the solution is updated and a convergence check is performed. If the solution satisfies the convergence criteria, Intercest proceeds with the next time step. Otherwise, the linearization stage is repeated.

2-2 Formulation

The fluids and gas in a hydrocarbon reservoir contain several components in a number of phases. The formulation of reservoir equations within INTERSECT is general, i.e. it makes no assumption about phase-component partitioning. Any component can exist in any phase and there is no maximum number of components or phases. Furthermore, no particular ordering of components and phases is assumed. For this reason, the only input necessary for the simulator is the phase-component partitioning.

2-2-1 Darcy's Law

From a theoretical point of view, it is possible to simulate multiphase flow in a porous medium by solving Navier-Stokes equation at a pore level. However, due to complex geometry of porous media this approach is not used in practice. Multiphase flow models are generally based on macroscopic equations such as Darcy's law, a special case of momentum equation. Darcy's law describes a linear relation between the fluid velocity and the pressure gradient, [9]:

$$\mathbf{u} = -\frac{1}{\mu}\mathbf{k}\left(\nabla P - \rho g \nabla z\right), \tag{2.1}$$

where,
$\mathbf{u}$ is the fluid velocity,
μ is the fluid viscosity,
$\mathbf{k}$ is the permeability tensor of the porous medium,
P is the pressure,
ρ is the fluid density,
g is the magnitude of the gravitational acceleration,
z is the depth.

The absolute permeability is an average measure of the ability of the porous medium to transmit fluid.
Denoting the potential $p - \rho g z$ by Φ, we can rewrite Equation (2.1) for incompressible flow in the following form:

$$\mathbf{u} = -\frac{1}{\mu}\mathbf{k}\nabla\Phi. \tag{2.2}$$

To apply Darcy's law to multiphase flow in a porous medium, the simulation domain should be divided into small control volumes so that the fluid and rock properties can be considered constant.

2-2-2 Governing Equation

From the previous discussion, we know that within INTERSECT it is assumed that any component can exists in any phase. If the changes in the compounds of hydrocarbons are negligible, the fluid flow can be simulated by the black oil approach. Otherwise, the more complex compositional fluid model should be applied.
In both cases, we introduce the nonlinear residual R_i^j to describe the transfer of fluid component i in grid block j,

$$R_i^j = \frac{\partial M_i^j}{\partial t} + \sum_{\kappa} F_i^{j\kappa} + \sum_{l} Q_i^{jl}, \tag{2.3}$$

where
$\frac{\partial M_i^j}{\partial t}$ is the rate of change of the number of moles of component i in cell j,
$F_i^{j\kappa}$ is the flux of component i into cell j from connected cell κ,
Q_i^{jl} is the flow of component i into cell j contributed by external source or sink l.

The non-linear residual is defined for each cell and each component. To satisfy the law of conservation of mass, it is required that $R_i^j \to 0$ within a certain tolerance.

In Equation (2.3), M_i^j is defined as

$$M_i^j = V_p^j \sum_{\alpha} b_\alpha^j S_\alpha^j x_{\alpha i}^j,$$

where,
α is the phase,
V_p^j is the pore volume of cell j,
S_α^j is the saturation,
b_α^j is the molar density of phase α,
$x_{\alpha i}^j$ is the mole fraction of component i in phase α.

Let $b_{i\alpha}$ be the molar density of component i in phase α and denote the number of components by N_c. If W_i and $\rho_{i\alpha}$ are the molecular weight and mass density of component i, respectively, then $b_{i\alpha} = \frac{\rho_{i\alpha}}{W_i}$. The molar density of phase α is then $b_\alpha = \frac{\sum_{i=1}^{N_c} \rho_{i\alpha}}{W_\alpha}$. Hence, the mole fraction of component i in phase α is $x_{\alpha i} = \frac{b_{i\alpha}}{b_\alpha S_\alpha}$. Furthermore, the saturation of a fluid phase is defined as the fraction of the void volume of a porous medium filled by this phase, [9].

The flux can be expressed as follows,

$$F_i^{j\kappa} = \bar{T}^{j\kappa} \sum_\alpha x_{\alpha i}^j b_\alpha^j u_\alpha^{j\kappa}, \tag{2.4}$$

where $\bar{T}^{j\kappa}$ and u_α^j are the partial transmissibility between cells j and κ and the Darcy's velocity, respectively.

According to Equation (2.2), for incompressible flow u_α^j is equal to

$$u_\alpha^{j\kappa} = \frac{k_a k_{r_\alpha}}{\mu_\alpha} \nabla \Phi_\alpha^{j\kappa}. \tag{2.5}$$

Here, k_a is the absolute permeability of the porous medium that depends on the geological structure of the reservoir. The variable k_{r_α} is the relative permeability function and depends on the saturation.

Remark 1. The permeability $\mathbf{k}$ from Equation 2.1 is a product of absolute and relative permeability.

Substituting Equation (2.5) into Equation (2.4), we obtain

$$F_i^{j\kappa} = T^{j\kappa}(k_a) \sum_\alpha \frac{x_{\alpha i}^j b_\alpha^j k_{r_\alpha}}{\mu_\alpha} \nabla \Phi_\alpha^{j\kappa},$$

where $T^{j\kappa}$ is the transmissibility (ability of a medium to transmit a fluid, based on the absolute permeability).

2-2-3 Thermodynamic Equilibrium

Thermodynamic phase equilibrium is needed when a component exists in more than one phase. The equilibrium is reached when the fugacity of each component in phase α equals to the fugacity of the same component in phase β:

$$f_{\alpha,i} = f_{\beta,i}. \tag{2.6}$$

Equation (2.6) is known as Gibbs relation. When it holds, the phases are in equilibrium and therefore the Gibbs energy of the system is at an extremum.

2-2-4 Saturation and Mole Fraction Constraints

Saturation and mole fraction constraints are formulated as follows:

$$\sum_\alpha S_\alpha = 1,$$
$$\sum_{i=1}^{n_c} x_{\alpha i} = 1 \qquad \forall\ \alpha,$$

with n_c equal to the total number of components.

The fact that all the components existing in a given volume fill the voids implies the above constraints.

2-2-5 Variables

Table 2-1 provides the property dependencies on the natural variables for isothermal processes.

Property	Symbol	Dependencies
pore volume	V_p	P
absolute permeability	k_a	P
relative permeability	$k_{r\alpha}$	S_α
viscosity	μ_α	$P, x_{\alpha i}$
potential difference	$\Delta\Phi_\alpha$	$P, x_{\alpha i}, S_\alpha$
phase density	b_α	$P, x_{\alpha i}$

Table 2-1: Property dependencies on natural variables.

Therefore, the following variables are selected as unknowns:

- pressure P,
- saturation S_α,
- mole fraction $x_{\alpha i}$.

Defining N_c and N_p as the number of components and the number of possible phases, respectively, the total number of governing equations for cell i is equal to $N = N_c \cdot N_p + N_p + 1$. This means that to make the system solvable, N variables should be selected. This number of variables can be reached if we prescribe to each cell one pressure variable, N_p saturation variables and $N_c \cdot N_p$ mole fraction variables.

Primary Variables

Primary variables are the variables that are determined after the saturation, mole fraction and phase equilibrium constraints are applied. The number of primary variables depends on the number of variables that are solved implicitly. There exist four solution approaches.

- *IMPES (Implicit Pressure Explicit Saturations):* pressure is solved implicitly, while saturations and mole fractions are solved explicitly.
- *IMPSAT (Implicit Pressure and Saturations):* pressure and saturations are solved implicitly, whilst mole fractions are solved implicitly.
- *FIM (Fully Implicit):* all primary variables are solved implicitly.
- *AIM (Adaptive Implicit Method):* cells that violate the CFL constraint, are treated fully implicit, whereas IMPES and IMPSAT are applied to the variables in more stable cells.

In this thesis we only consider fully implicit method and AIM, whereby IMPES is used for stable cells. It should be noted that in fully implicit models the time step is assumed to be unconditionally stable, [35]. However, in practice the size of the time step is restricted due to mathematical and physical processes.

2-2-6 Black Oil Fluid Model

In this section, we describe the black oil model for a three component system. Consider a system that consists of three phases (two liquid phases and a gaseous phase) and three components (water, oil and gas). Water is only contained in one liquid phase, denoted by w. At the same time, gas and oil are contained in the gaseous phase g and the oil phase o. In this three component system, the solution vector X^j for grid block j generally consists of the oil pressure and water and gas saturation,

$$\mathbf{X}^j = [P_o^j, S_w^j, S_g^j]^T.$$

The residual vector R^j is a function of X^j and has three elements: oil, water and gas residual.

$$\mathbf{R}^j = [R_o^j, R_w^j, R_g^j]^T.$$

The accumulation term is defined as

$$\mathbf{M}^j = V_p^j \begin{bmatrix} \frac{S_o}{B_o} + \frac{R_v S_g}{B_g} \\ \\ \frac{S_w}{B_w} \\ \\ \frac{S_g}{B_g} + \frac{R_s S_o^j}{B_o} \end{bmatrix}_j,$$

where,
B_i is the formation volume factor of component i, i.e.

$$B_i = \frac{\text{volume of component } i \text{ at reservoir conditions}}{\text{volume of component } i \text{ at surface conditions}},$$

R_s is the solution oil-gas ratio, i.e.

$$R_s = \frac{\text{volume of surface gas dissolved in reservoir oil}}{\text{volume of surface oil from reservoir oil}},$$

R_v is the vapor oil-gas ratio, i.e.

$$R_v = \frac{\text{volume of surface oil in reservoir gas}}{\text{volume of surface gas in reservoir gas}}.$$

The rate of flow from a neighbouring cell k is

$$\mathbf{F}^{jk} = T^{jk} \begin{bmatrix} \frac{k_{ro}}{B_o\mu_o} & 0 & \frac{R_v k_{rg}}{B_g\mu_g} \\ \\ 0 & \frac{k_{rw}}{B_w\mu_w} & 0 \\ \\ \frac{R_s k_{ro}}{B_o\mu_o} & 0 & \frac{k_{rg}}{B_g\mu_g} \end{bmatrix}^* \times \begin{bmatrix} \Delta\Phi_o^{jk} \\ \\ \Delta\Phi_w^{jk} \\ \\ \Delta\Phi_g^{jk} \end{bmatrix}.$$

Here, the * symbol indicates that the fluid mobilities are to be evaluated in the upstream cell.

The black oil model is a special case of the compositional model. The compositional model treats each hydrocarbon component separately and therefore is more complex. Since our research is focused on the problems with the black oil formulation, we refer to [8] for more details about the compositional model.

Example 2.1. For illustrative purposes, we consider the case of two-phase flow in heterogeneous porous media described in [24]. The system consisting of oil and water, is immiscible, i.e. the components do not dissolve. Fluids and rock are incompressible. That is, their density and porosity remain constant. In addition, we ignore gravity effects and the effect of capillary action (the capillarity pressure can be used to flow opposite to gravity).
The governing conservation equations for the flow under these assumptions can be written as:

$$\begin{cases} \phi\frac{\partial S_w}{\partial t} - \nabla \cdot \mathbf{u}_w = -Q_w, \\ \\ \phi\frac{\partial S_o}{\partial t} - \nabla \cdot \mathbf{u}_o = -Q_o, \end{cases} \tag{2.7}$$

where ϕ denotes the porosity (the ratio of pore volume to total volume). Subscript w corresponds with water, whilst o corresponds with oil. The Darcy velocity in this case can be written as:

$$u_j = -k_a(\mathbf{x})\lambda_j \nabla P.$$

Here, λ_j is the phase mobility given by,

$$\lambda_j = \frac{k_{r_j}(S(\mathbf{x},t))}{\mu_j}. \tag{2.8}$$

Obviously, the saturations should sum up to one:

$$S_w + S_o = 1. \tag{2.9}$$

Using Equation (2.9), we can replace system presented in (2.7) by

$$\begin{cases} -\nabla \cdot \mathbf{u} = Q, \\ \\ \phi\frac{\partial S_w}{\partial t} - \nabla \cdot (f_w \mathbf{u}) = -q_w, \end{cases} \tag{2.10}$$

where,
$\mathbf{u} = \mathbf{u}_w + \mathbf{u}_o$ is the total Darcy velocity,
$Q = Q_w + Q_o$ is the total contribution of the source terms,
f_w is the fractional flow of water, i.e. $\mathbf{u}_w = f_w\mathbf{u}$.

Darcy's law provides a useful expression for the total Darcy velocity:

$$\mathbf{u} = -k_a(\mathbf{x})\lambda\nabla P, \tag{2.11}$$

where $\lambda = \lambda_w + \lambda_o$ is the total mobility. It should be noted that from Equation (2.8) follows that the total mobility depends on the saturation.

Substituting Equation (2.11) into the first equation in System (2.10) gives us the pressure equation:

$$\nabla \cdot k_a(\mathbf{x})\lambda \nabla P = Q.$$

In reservoir simulations, the velocity in System (2.10) is usually obtained by first solving the pressure equation for the pressure field and then computing the total velocity using Equation (2.11). The non-linear system

$$\begin{cases} \nabla \cdot k_a(\mathbf{x})\lambda \nabla P = Q, \\ \mathbf{u} = -k_a(\mathbf{x})\lambda \nabla P, \\ \phi \frac{\partial S_w}{\partial t} - \nabla \cdot (f_w \mathbf{u}) = -Q_w \end{cases}$$

is coupled due to the dependence of the mobility on the saturation.

2-3 Non-linear Solver

2-3-1 Newton-Raphson

In this section we discuss the Newton-Raphson method. The method is a very efficient root finding method if the initial guess is sufficiently close to the solution [29].

For a vector $\mathbf{X} \in \mathbb{R}^n$ and differentiable vector function $\mathbf{R} : \mathbb{R}^n \to \mathbb{R}^n$ Newton-Raphson attempts to solve the non-linear system

$$\mathbf{R}(\mathbf{X}) = \mathbf{0}. \tag{2.12}$$

The function $\mathbf{R}$ represents the equations obtained in the formulation stage (see Equation (2.3)), while the vector $\mathbf{X}$ corresponds to pressure, saturation and mole fraction variables.

A Taylor expansion of Equation (2.12) around $\mathbf{X}$ yields

$$\mathbf{R}(\mathbf{X} + \Delta\mathbf{X}) = \mathbf{R}(\mathbf{X}) + \Delta\mathbf{X} \cdot \nabla\mathbf{R}|_{\mathbf{X}} + \mathcal{O}\left(\Delta\mathbf{X}^2\right),$$

where $\Delta\mathbf{X}$ is the difference between $\mathbf{X}$ and an arbitrary point in the neighbourhood of $\mathbf{X}$. Ignoring higher order terms, we obtain

$$\nabla\mathbf{R}(\mathbf{X})\Delta\mathbf{X} = -\mathbf{R}(\mathbf{X}).$$

By defining $\Delta\mathbf{X}$ as $\mathbf{X}_{k+1} - \mathbf{X}_k$, an iterative method is constructed:

$$\nabla\mathbf{R}(\mathbf{X}_k)\Delta\mathbf{X} = -\mathbf{R}(\mathbf{X}_k).$$

Obviously, $\mathbf{R}(\mathbf{X})$ represents the residuals, while $\Delta\mathbf{X}$ is the solution deltas (the correction required to take the coupled system towards the solution). For a particular time step, the use of Newton-Raphson can be outlined as follows:

1. Calculate $\mathbf{R}(\mathbf{X})$.

2. Apply a linear solver to solve the linear system $A\Delta\mathbf{X} = -\mathbf{R}(\mathbf{X})$, where A is the Jacobian $\nabla\mathbf{R}(\mathbf{X})$.

3. Update the solution: $\mathbf{X} \to \mathbf{X} + \Delta\mathbf{X}$.

This process is iterated to convergence.

After the discretization, which is performed using the finite volume method, the coupled linear system can be written as:

$$\begin{bmatrix} A_{rr} & A_{rw} \\ A_{wr} & A_{ww} \end{bmatrix} \begin{bmatrix} \Delta X_r \\ \Delta X_w \end{bmatrix} = - \begin{bmatrix} R_r \\ R_w \end{bmatrix}, \tag{2.13}$$

where the subscript r is for the reservoir and w for the wells (see Appendix A). For instance, A_{wr} is the submatrix of derivatives of the well equations with respect to the reservoir variables. For the details of discretization see [35].

The matrix A is typically very sparse, mainly due to the fact that the each cell is only connected to its nearest neighbours. The order of equations may vary per time step.
The solution of the full system is outside the scope of this thesis. In the upcoming chapters only the pressure part corresponding to the submatrix A_{rr} will be considered.

2-3-2 Convergence Criteria

In each iteration of the Newton-Raphson method, we have a linear approximation to a non-linear system. For convergence a few iterations may be required. INTERSECT uses the following convergence criteria:

- *Solution delta limit.* For each solution variable, the maximum absolute solution delta over all reservoir cells is bounded by a user-defined limit.

- *Overall material balance error.* The overall reservoir material balance error, which is defined as the sum of residuals from Equation (2.3) over all cells is smaller than a user-defined limit.

- *Maximum/minimum number of Newton-Raphson iterations.* A minimum number of Newton iterations may be required to accept the time step as converged. In addition, there is a maximum number of iterations to reject and halve the time step.

For the convergence of the solution, at least the first or the second convergence criterion should be satisfied.

2-3-3 Time Step Selection

If the solution converges, a new time step size has to be selected. To choose an appropriate time step the following criteria are used:

- *Maximum/minimum time step size.* An upper and lower limit on the time step size are provided by the maximum and minimum time step, respectively.
- *Maximum increase factor.* The ratio of the current time step to the last converged time step is bounded by the maximum increase factor.
- *Report step synchronization.* The time step size depends on the upcoming report date (the user-requested time at which the solution has to be computed. Sometimes two time steps are required to reach the report date. In this case, it can be more beneficial to use two equal medium-sized time steps instead of a large step followed by a small one.
- *Time truncation error limit.* The time truncation error limit prevents propagation of truncation error due to linearization.
- *Solution change criteria.* The Newton-Raphson method only converges when the iteration starts close enough to the final solution. Therefore, the each of the solution variables is bounded by a maximum change criteria to ensure the convergence.
- *CFL condition.* Except simulations with an unconditionally stable time step, the time step is restricted by the Courant-Friedrichs-Lewy (CFL) criterion to guarantee the convergence of the full system.

2-3-4 Linear Solver

Consider a coupled linear system

$$A\mathbf{u} = \mathbf{f}, \tag{2.14}$$

where $A \in \mathbb{R}^{n \times n}$ and $\mathbf{u}, \mathbf{f} \in \mathbb{R}^n$.

To solve this system, INTERSECT uses FGMRES, a Krylov subspace method, preconditioned by CPR.
FGMRES, Flexible Generalized Minimal RESidual, is a variant of the right-preconditioned GMRES method (Appendix B-3) that allows changes in the preconditioning at every step. For this scheme, any iterative method can be used as a preconditioner, even the standard GMRES method itself.

The algorithm is presented in Algorithm 1.
It should be noted that vector $\bar{\mathbf{w}}_i$ on line 4 is allowed to change at every step:

$$\bar{\mathbf{w}}_i = M_i^{-1}\mathbf{q}_i,$$

where M_i is the preconditioner and $\mathbf{q}_i$ is the ith Arnoldi vector.
Subsequently, vectors $\bar{\mathbf{w}}_i$ are saved to be used in updating the solution, $\mathbf{v}_m$, on line 15. Obviously, when $M_i = M$ for $i = 1, \ldots, m$, the new method is mathematically equivalent to right-preconditioned GMRES.

A short introduction to the Krylov subspace methods is provided in Appendix B. For the basic properties of FGMRES we refer to [32].

Algorithm 1: FGMRES

```
Data: Initial guess $v_0$.
1  Setup $(m+1) \times m$ zero-matrix $\tilde{H}_m$
2  Compute $\mathbf{r}_0 = \mathbf{f} - A\mathbf{v}_0$ and $\mathbf{q}_1 = \frac{\mathbf{v}_0}{||\mathbf{v}_0||}$.
3  for $i$ = 1,2, ..., $m$ do
4      $\bar{\mathbf{w}}_i = M_i^{-1}\mathbf{q}_i$
5      $\mathbf{w} = A\bar{\mathbf{w}}_i$
6      for $j$ = 1, ..., $i$ do
7          $h_{j,i} = (\mathbf{w}, \mathbf{q}_j)$
8          $\mathbf{w} = \mathbf{w} - h_{j,i}\mathbf{q}_j$
9      end
10     $h_{i+1,i} = ||\mathbf{w}||_2$
11     if $h_{i+1,i} = 0$ then $m = i$ and break
12     $\mathbf{q}_{i+1} = \frac{1}{h_{i+1,i}}\mathbf{w}$
13 end
14 Define $\bar{W}_m = \left[ \mathbf{w}_1 \mid \mathbf{w}_2 \mid \dots \mid \mathbf{w}_m \right]$.
15 Compute $\mathbf{v}_m = \mathbf{v}_0 + \bar{W}_m\mathbf{y}_m$ where $\mathbf{y}_m$ minimizes $\mathbf{y}_m = ||\beta\mathbf{e}_1 - \tilde{H}_m\mathbf{y}||_2$.
16 if convergence then stop
17 $\mathbf{v}_0 = \mathbf{v}_m$ and go to line 2.
```

In line 15 vector $\mathbf{y}_m$ is found by solving $(i+1)\times i$ matrix least square problem with Hessenberg structure. The solution is found by means of QR decomposition (see [39]). Due to the Hessenberg structure this is done at a cost of $\mathcal{O}(i^2)$ flops.

2-4 CPR

The reservoir part of (2.13) can be written as follows:

$$A\mathbf{x} = \begin{bmatrix} A_{pp} & A_{ps} \\ A_{sp} & A_{ss} \end{bmatrix} \begin{bmatrix} \mathbf{x}_p \\ \mathbf{x}_s \end{bmatrix} = \begin{bmatrix} \mathbf{b}_p \\ \mathbf{b}_s \end{bmatrix} = \mathbf{b},$$

where
A is the matrix A_{rr} from Equation (2.13),
A_{pp} represents the pressure block coefficients,
A_{ss} corresponds to the non-pressure block coefficients (i.e. saturation),
A_{ps} and A_{sp} represent the respective coupling coefficients.

The pressure block A_{pp} has an elliptic (or parabolic) structure, while A_{ss} holds coefficients of a set of hyperbolic problems [38].

The CPR preconditioner is based on the idea that coupled system solutions are mainly determined by the solution of their elliptic (or parabolic) components [38]. For this reason, the procedure begins by extracting and solving pressure subsystems, forming the first stage of the CPR algorithm. Residuals associated with the pressure solution are corrected with an additional preconditioning step that is applied to the full system, the second stage. In other words, the CPR preconditioner solves the coupled linear system as follows:

1. Obtain the decoupled system: $A^*\mathbf{x} = \mathbf{b}^*$.
2. Use the first stage preconditioner, preferably an iterative method, to solve the pressure system: $A^*_{pp}\boldsymbol{\delta}_p = \mathbf{r}_p$.
3. Compute new residual: $\hat{\mathbf{r}} = \mathbf{r} - A^* \begin{bmatrix} \boldsymbol{\delta}_p \\ \mathbf{0} \end{bmatrix}$.
4. Apply the second stage precondtioner and correct: $\boldsymbol{\delta} = M^{-1}\hat{\mathbf{r}} + \begin{bmatrix} \boldsymbol{\delta}_p \\ \mathbf{0} \end{bmatrix}$.

Here, $\mathbf{r} = \begin{bmatrix} \mathbf{r}_p & | & \mathbf{r}_s \end{bmatrix}^T$ and $\boldsymbol{\delta} = \begin{bmatrix} \boldsymbol{\delta}_p & | & \boldsymbol{\delta}_s \end{bmatrix}^T$ are the residual vector corresponding to the decoupled system and correction obtained after the two stages, respectively.

The decoupling procedure and second stage preconditioner are presented in Section 2-4-1 and 2-4-2, respectively. The first stage preconditioner is the AMG method, which is described in Chapter 3.

2-4-1 Decoupling

To separate the pressure part from the remaining saturation part, A_{ps} has to be transformed to a (nearly) zero matrix. This can be accomplished by taking the Schur complement (see [19]) as described below.

Definition 2.1. Let $A \in \mathbb{R}^{m\times m}$ and $\mathbf{1} = \begin{bmatrix} 1 & 1 & \dots & 1 \end{bmatrix} \in \mathbb{R}^{1\times m}$. Then $\operatorname{colsum}(A)$ is defined as an $m \times m$ diagonal matrix with diagonal equal to $\mathbf{1}A$.

Define matrix Q by

$$Q = \operatorname{colsum}(A_{ps}) \operatorname{colsum}(A_{ss})^{-1} \approx A_{ps}A_{ss}^{-1}.$$

Using Q, the original system can be converted to the system

$$A^*\mathbf{x} = \begin{bmatrix} A^*_{pp} & A^*_{ps} \\ A_{sp} & A_{ss} \end{bmatrix} \begin{bmatrix} \mathbf{x}_p \\ \mathbf{x}_s \end{bmatrix} = \begin{bmatrix} \mathbf{b}^*_p \\ \mathbf{b}_s \end{bmatrix} = \mathbf{b}^*, \tag{2.15}$$

where

$$
\begin{aligned}
A^*_{pp} &= A_{pp} - QA_{sp}, \\
A^*_{ps} &= A_{ps} - QA_{ss}, \\
\mathbf{b}^*_p &= \mathbf{b}_p - Q\mathbf{b}_s.
\end{aligned}
$$

Obviously, A^*_{ps} is a nearly zero matrix. Therefore, the coupling between pressure and non-pressure blocks is weak.

2-4-2 Second Stage: ILU

Consider a system of linear equations

$$A\mathbf{u} = \mathbf{f}.$$

The LU decomposition is a direct method that factors matrix A as the product of a lower triangular matrix L and an upper triangular matrix U, $A = LU$. The linear system $LU\mathbf{u} = \mathbf{f}$ is easily solved: the solution of the lower triangular system $L\mathbf{y} = \mathbf{f}$ is found, followed by the solution of the upper triangular part $U\mathbf{u} = \mathbf{y}$. A detailed description of the LU decomposition can be found in [41].

Due to the fill in, the LU decomposition for large, sparse matrices is expensive. For $A \in \mathbb{R}^{n\times n}$, it requires approximately $\mathcal{O}(n^2)$ flops, which is not competitive with iterative solvers [20]. For this reason, the LU algorithm is frequently used in incomplete form as a preconditioner for an iterative solver. The ILU method requires the residual matrix $R = A - LU$ to satisfy some constraints, such as having a certain sparsity pattern.

Let $A \in \mathbb{R}^{n\times n}$ be a general sparse matrix with elements a_{ij} and P a zero pattern set, such that

$$P \subset \{(i,j) : i \neq j, 1 \leq i,j \leq n\}.$$

For a general static pattern, an Incomplete LU factorization can be computed as follows.

Algorithm 2: General static pattern ILU.

```
Data: $A, P$
1 foreach $(i,j) \in P$ do $a_{ij} = 0$;
2 for $k = 1, \ldots, n-1$ do
3     for $i = k+1, \ldots, n$ and if $(i,k) \notin P$ do
4         $a_{ik} \leftarrow a_{ik}/a_{kk}$
5         for $j = k+1, \ldots, n$ and for $(i,j) \notin P$ do
6             $a_{ij} \leftarrow a_{ij} - a_{ik}a_{kj}$
7         end
8     end
9 end
```

The ILU(0) factorization sets the zero pattern P to be precisely the zero pattern of A. For a detailed description of ILU(0), see [31]. INTERSECT contains block-wise ILU(0) factorization. That is, instead of treating A point-wise, ILU(0) is used in a block-wise approach.

The main advantages of the ILU methods are the simplicity and robustness for a broad class of problems. Regarding the use of ILU within INTERSECT, its primary disadvantage is that there is still no highly scalable parallel ILU(0) algorithm, [13]. The concept of scalability is explained in Chapter 4.

2-5 Parallel Computing

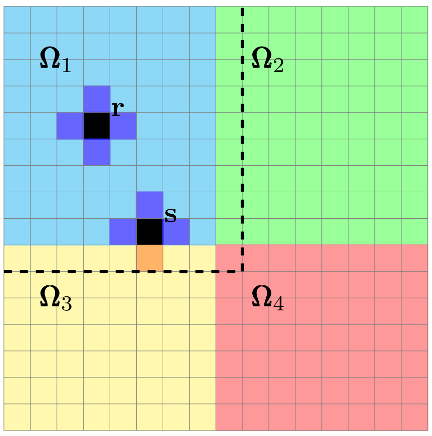


Figure 2.2: Domain partitioned into four subdomains. Blue, green, yellow and red cells are assigned to subdomain $\Omega_1, \Omega_2, \Omega_3$ and Ω_4, respectively. Black cells r and s and their dark blue neighbours are internal nodes of the subdomain Ω_1. The orange cell is assigned to Ω_3. The dashed line separates the halo cells of the subdomain Ω_1 from the domains they are assigned to.

The number of reservoir cells within INTERSECT is generally extremely large. Therefore, it is natural to parallelize the simulation. For parallel computing, the overall grid is partitioned into subdomains, one on each processor. Figure 2.2 illustrates a partitioning into four subdomains.
In this example, subdomain Ω_i $(i = 1, \dots, 4)$ is assigned to processor i which performs all the computations concerning the cells comprising this particular domain.
Furthermore, since updating a node requires the values of its neighbours, processor i maintains copies of the cells on other processors to which it is connected. These cells are called halo cells. The use of halo cells, minimizes the frequency of information that has to be send between processors.
Figure 2.2 highlights two internal cells of the subdomain Ω_1, cell r and cell s. It also shows the connections of r and s: seven dark blue nodes and one orange node. The orange cell is a halo cell on processor 1. That is, processor 1 only keeps a copy of the orange cell and updates its value after it is computed by processor 3.

The matrix obtained from the overall grid is partitioned in a similar way. More precisely, if a given cell is assigned to a particular processor then the entire row corresponding to that cell is also stored on the same processor. This is shown in Figure 2.3.
The figure schematically represents a linear system $A\mathbf{u} = \mathbf{f}$. The neighbours of r and s are denoted by x. The remaining values in the two rows are equal to zero and are not shown in

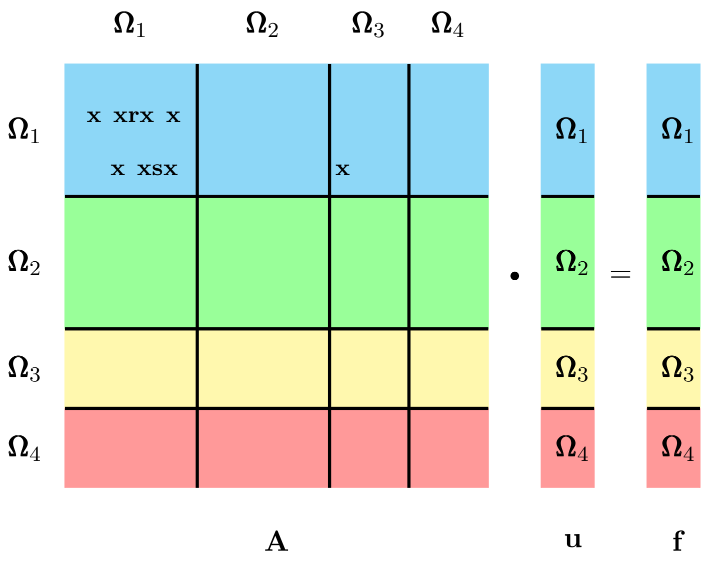


Figure 2.3: Schematic illustration of a general linear system $A\mathbf{u} = \mathbf{f}$ corresponding to the domain shown in Figure 2.2. The neighbouring cells of p and s are denoted by x.

the Figure 2.3. It should be noted that the value of the halo cell is stored into the sub-block matrix $A_{[\Omega_1\Omega_3]}$, while other neighbouring nodes belong to the sub-matrix $A_{[\Omega_1\Omega_1]}$.

Analogously, the solution and right-hand-side vector are partitioned into the subdomains and maintain the copies of the halo cells. For the sake of simplicity, Figure 2.3 does not show the halo cells contained by the vectors. However, an extensive version of vector $\mathbf{u}$ is illustrated in Figure 2.4, that demonstrates the principles behind the calculation of a dot product. In the figure, the inner product $\mathbf{w} \cdot \mathbf{u}$ is equal to a. The value of a is obtained by communicating a_i's between the processors, where a_i $(i = 1, \ldots, 4)$ may be seen as a "local" inner product calculated by processor i. Obviously, if no halo cells are present, then $a = \sum_{i=1}^{4} a_i$.

The described storage structure makes a matrix-vector remarkably straightforward. Interestingly, this operation requires less communication than the calculation of an inner product and therefore is cheaper. Of course, the row vector of matrix A corresponding to cell s can be seen as vector $\mathbf{w}$ in Figure 2.4. Due to the data structure of A, it is known a priori that only one a_i is unequal to zero. In our example, a_i is zero for $i = 2, \ldots, 4$, so to calculate the value of a we only need processor 1, because $a = a_1$. Therefore, for parallel simulations the calculation of a matrix-vector product is cheaper than the calculation of an inner product due to the matrix storage structure.

For more information on parallel computing see [35] and [16].

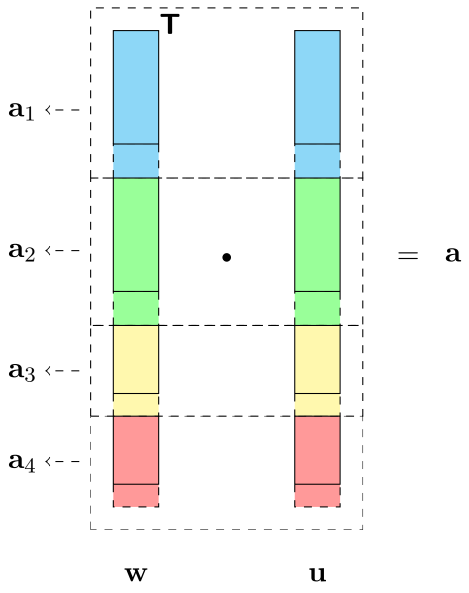


Figure 2.4: Schematic illustration of an inner product $\mathbf{w} \cdot \mathbf{u}$ calculation. Colored boxes with partially dashed border lines represent the halo cells. Blue, green, yellow and red parts of the vectors are owned by processor 1, 2, 3 and 4 respectively.

Chapter 3

AMG

3-1 Classical AMG

Multigrid methods form an essential step towards solving systems of linear equations resulting from the discretization of PDEs in a fast and efficient manner. Typically, these methods are applied to PDEs of elliptic nature.
Multigrid methods are called optimal, because they can solve a problem with N unknowns with only $\mathcal{O}(N)$ work. Since it is possible to allocate this work on parallel computers, in general large systems are solved readily by means of the multigrid techniques, [14].

A discretized PDE can be written as a linear system

$$A\mathbf{u} = \mathbf{f}, \tag{3.1}$$

where $A \in \mathbb{R}^{n\times n}$ is the discretization matrix, $\mathbf{f} \in \mathbb{R}^n$ is the right-hand side and $\mathbf{u} \in \mathbb{R}^n$ is the solution vector. The multigrid methods solve system (3.1) by employing two complementary processes: smoothing and defect correction. The smoothing involves the application of a smoother S_h, generally a basic iterative method (BIM) such as Jacobi and Gauss-Seidel. BIMs efficiently reduce the high frequency components of the iteration error, but their convergence is hindered by the low frequency errors. If however a problem is transfered to a coarser grid, the previously low frequency errors can be damped by a BIM effectively, because they turn into high frequency errors after coarsening. This error elimination technique is called defect correction. The use of BIMs is further described in Section 3-1-2.

There are two main types of multigrid methods: geometric and algebraic. Geometric mutligrid determines the various multigrid components based on the geometry of the problem and is explained in detail in [6]. Algebraic multigrid method (AMG) can be constructed without the explicit knowledge of the stencil. This approach uses only the underlying matrices. Therefore, AMG can be readily applied to structured and unstructured problems. This flexibility comes with a price: the approach requires a relatively expensive setup phase in addition to the solve phase, consisting of the smoothing and defect correction processes.

AMG was introduced in the 80s in [30, 3, 4, 5]. Since then, many AMG versions have been developed. In this section we focus on the classical Ruge-Stüben AMG.
First of all, we explain in Section 3-1-1 what the term 'grid' represents in the context of AMG. After that, we describe the smoothing process (Section 3-1-2). The discussion is revolved around the Jacobi and Gauss-Seidel methods. The fundamental concepts of dependence and algebraic smoothness are introduced in Section 3-1-3 and 3-1-4, respectively. Subsequently, the heuristic criteria, required for the coarsening procedure, are presented. The heuristics are stated in Section 3-1-5, while their application is visualized in Section 3-1-6. The transfer and coarse grid operators are described in Section 3-1-7 and Section 3-1-8, respectively. Since at this stage all crucial components of AMG are already introduced, we provide the AMG setup and solve algorithms in Section 3-1-9. The description of the method is concluded by a note on its convergence and costs (Section 3-1-10 and Section 3-1-11).

3-1-1 Undirected Adjacency Graph

In the geometric case, the unknown variables u_i have known spatial locations on a fine grid and a subset of these unknown variables is used on a coarse grid. In the algebraic case, the grid is generally not known, but by analogy with geometric multigrid, a subset of the variables u_i serves as the coarse grid unknowns. Clearly, it is useful to identify the grid with the indices of unknown quantities.
For illustrative purposes, we construct an adjacency graph corresponding with matrix A. The vertices of the undirected adjacency graph are associated with the grid points. Letting a_{ij} be the entries of A, an edge between the ith and jth vertices is drawn, when $a_{ij} \neq 0$ or $a_{ji} \neq 0$ with $i \neq j$ (see Figure 3.1). For the sake of simplicity, loops corresponding to the case $i = j$ are not included in the adjacency graph in Figure 3.1. obviously, the grid is based solely on the matrix A.

$$A = \begin{pmatrix} X & X & & & & X \\ X & X & X & & X & \\ & X & X & X & & \\ & & X & X & X & X \\ & X & & X & X & X \\ X & & & X & X & X \end{pmatrix}$$

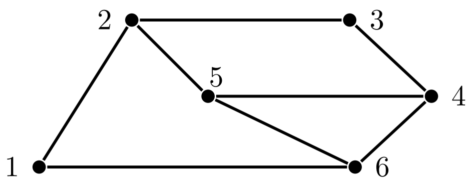


Figure 3.1: The sparsity structure of A, where X represents a non-zero entry, and the corresponding undirected adjacency graph.

3-1-2 Smoothing: Basic Iterative Methods

Suppose that system (3.1) has a unique solution $\mathbf{u}$. Then, by $\mathbf{v}$ we denote the iterative approximation to $\mathbf{u}$. The accuracy of $\mathbf{v}$ can be established using any of the standard vector norms for the error

$$\mathbf{e} = \mathbf{u} - \mathbf{v}.$$

Unfortunately, for the most problems the error is just as inaccessible as the exact solution itself. For this reason, the quality of the approximation is usually presented by the residual, given by

$$\mathbf{r} = \mathbf{f} - A\mathbf{v}.$$

It follows that

$$A\mathbf{e} = \mathbf{r}. \tag{3.2}$$

Equation (3.2) is known as the residual equation.

In Section B we introduced a non-singular matrix M and matrix $N = M - A$ and explained that at step $k+1$ of an iterative method the numerical solution is computed as follows:

$$\mathbf{v}^{k+1} = \mathbf{v}^k + M^{-1}\mathbf{r}^k. \tag{3.3}$$

It should be noted that $\mathbf{e}^{k+1} = \left(I - M^{-1}A\right)\mathbf{e}^k$, while $\mathbf{r}^{k+1} = \left(I - AM^{-1}\right)\mathbf{r}^k$. This implies that $\mathbf{r}^{k+1}$ and $\mathbf{e}^{k+1}$ satisfy the residual equation provided $\mathbf{r}^k$ and $\mathbf{e}^k$ do so.
The iteration matrix of a BIM is defined as

$$B = I - M^{-1}A.$$

To introduce specific basic iterative methods, we denote the matrix A as

$$A = D - L - U,$$

where D is the diagonal matrix, $-L$ is the strictly lower triangular part of A and $-U$ is the strictly upper triangular part of A. In addition, we define matrix L_d and U_d:

$$L_d = D^{-1}L,$$
$$U_d = D^{-1}U.$$

Although there exist various basic iterative methods, we focus on Gauss-Seidel, Jacobi and damped Jacobi methods.

Jacobi

The Jacobi method is one of the simplest iterative schemes. It sets $M_{JAC} = D$. Therefore, N_{JAC} becomes equal to $L + U$. With other words,

$$\mathbf{v}^{k+1} = D^{-1}\left((L+U)\,\mathbf{v}^k + \mathbf{f}\right).$$

In component form, we can write the Jacobi iteration as follows:

$$v_i^{k+1} = \left(f_i - \sum_{j=1, j\neq i}^{n} A_{ij}v_j^k\right)/A_{ii} \qquad \forall i = 1, \ldots, n.$$

It can be easily shown that matrix B_{JAC} is equal to $L_d + U_d$, see [41].

A variation of the Jacobi method can be obtained using the damping parameter $\omega \in \mathbb{R}$:

$$\mathbf{v}^{k+1} = (1-\omega)\mathbf{v}^k + \omega\bar{\mathbf{v}}^{k+1},$$

where $\bar{\mathbf{v}}^{k+1}$ is the original Jacobi iterand at step $k+1$. Obviously, $\omega = 1$ yields the original Jacobi scheme.

Gauss-Seidel

The Gauss-Seidel method is constructed by setting

$$M_{GS} = D - L,$$
$$N_{GS} = U.$$

Thus, $\mathbf{v}^{k+1} = (D-L)^{-1}\left(U\mathbf{v}^k + \mathbf{f}\right)$. This leads to the following equation in component form:

$$v_i^{k+1} = \left(f_i - \sum_{j=1}^{i-1} A_{ij}v_j^{k+1} - \sum_{j=i+1}^{n} A_{ij}v_j^k\right)/A_{ii} \quad \forall i = 1,\dots,n.$$

For the Gauss-Seidel method, $B_{GS} = (I - L_d)^{-1}U_d$, [41].
As shown above, the (damped) Jacobi method calculates all components of the new iteration a priori, while the Gauss-Seidel method uses the components as soon as they become available. This is the main difference between the two methods.

According to the calculation (B.1) and the definition of B, the basic iterative methods may be presented in the form

$$\mathbf{v}^1 = B\mathbf{v}^0 + M^{-1}\mathbf{f}. \tag{3.4}$$

Moreover, BIM's are designed so that the exact solution is a fixed point of the iteration, [6]:

$$\mathbf{u} = B\mathbf{u} + M^{-1}\mathbf{f}. \tag{3.5}$$

Subtracting Equation (3.4) from Equation (3.5), we obtain

$$\mathbf{e}^1 = B\mathbf{e}^0.$$

Repeating this argument m times, provides us with the error in the mth approximation:

$$\mathbf{e}^m = B^m\mathbf{e}^0. \tag{3.6}$$

To proceed further, the definitions of the matrix norm and the spectral radius are required.

Definition 3.1. The matrix norm induced by the vector norm $||.||_p$ with $1 \le p < \infty$ is given by

$$||A||_p = \sup_{\mathbf{x}\neq 0} \frac{||A\mathbf{x}||_p}{||\mathbf{x}||_p}.$$

Definition 3.2. The spectral radius of matrix A is defined by

$$\rho(A) = \{\max_{i=1,\dots,n} |\lambda_i| : \lambda_i \text{ is an eigenvalue of A}\}. \tag{3.7}$$

For a particular vector norm and its corresponding matrix norm, it is not difficult to see that due to the consistency condition the error after m iterations is bounded by

$$||\mathbf{e}^m|| \le ||B||^m||\mathbf{e}^0||.$$

Thus, if $||B|| < 1$, the iteration will converge. In [21] it is demonstrated that

$$\lim_{m\to\infty} B^m = 0 \quad \text{if and only if} \quad \rho(B) < 1.$$

It follows that the iteration associated with matrix B converges for all initial guess if and only if $\rho(B) < 1$. Further discussion of the convergence of basic iterative methods is held using a straightforward, but illustrative example.

Example 3.1. Consider the damped Jacobi iteration applied to the two-point boundary problem that describes the steady-state temperature distribution in a long uniform rod:

$$-u''(x) = f(x), \qquad 0 < x < 1,$$

with homogeneous Dirichlet boundary conditions. The domain is partitioned into n subintervals by by introducing the grid points $x_j = jh$ with $h = \frac{1}{n}$. The components of the approximation vector $\mathbf{v} = (v_1, v_2, \dots, v_{n-1})^T$ satisfy the linear system of $n-1$ equations:

$$\frac{-v_{j-1} + 2v_j - v_{j+1}}{h^2} = f(x_j), \quad 1 \le j \le n-1,$$
$$v_0 = v_n = 0.$$

Thus, in the matrix form we can write $A\mathbf{v} = \mathbf{f}$, where $\mathbf{f} = (f_1, f_2, \dots, f_{n-1})^T$ and

$$A = \frac{1}{h^2}\begin{bmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & & \\ & & & & & -1 \\ & & & & -1 & 2 \end{bmatrix}.$$

The iteration matrix of the damped Jacobi method satisfies $B_\omega = (1-\omega)I + \omega B_{JAC}$. Therefore, we find

$$B_\omega = I - \frac{\omega}{2}A.$$

It follows that the eigenvalues of B_ω and A are related by $\lambda(B_\omega) = 1 - \frac{\omega}{2}\lambda(A)$. The eigenvalues of A are given by

$$\lambda_l(A) = 4\sin^2\left(\frac{l\pi}{2n}\right), \quad 1 \le l \le n-1.$$

It is useful to calculate the corresponding eigenvectors. We denote the jth component of the lth eigenvector $\mathbf{w}_l$ by $w_{l,j}$, where

$$w_{l,j} = \sin\left(\frac{jl\pi}{2n}\right), \quad 1 \le l \le n-1, \quad 0 \le j \le n.$$

When the wavenumber l is in the range $1 \le l \le \frac{n}{2}$, these Fourier modes are called low-frequency or smooth modes, since the corresponding sine waves are long and smooth. Analogously, the modes in the upper half of the spectrum, with $\frac{n}{2} \le l \le n-1$, are known as high-frequency or oscillatory waves.

We conclude that the eigenvalues of B_ω are

$$\lambda_l(B_\omega) = 1 - 2\omega \sin^2\left(\frac{l\pi}{2n}\right),$$

whilst the eigenvectors of B_ω are equal to those of A. Thus, the iteration converges when $0 < \omega \leq 1$.

Using the eigenvectors of the iteration matrix, it is possible to compute the expansion for the error after m iterations. The error in an initial guess of the Jacobi method can be presented as

$$\mathbf{e}^0 = \sum_{l=1}^{n-1} c_l \mathbf{w}_l.$$

Substituting this into Equation (3.6), we obtain

$$\mathbf{e}^m = \sum_{l=1}^{n-1} c_l B_\omega^m \mathbf{w}_l = \sum_{l=1}^{n-1} c_l \lambda_k^m(B_\omega)\mathbf{w}_l. \tag{3.8}$$

The last step follows from the fact that $\mathbf{w}_l$ is an eigenvector of the iteration matrix. Equation (3.8) shows that after m iterations the lth mode of the original error is reduced by a factor $\lambda_k^m(B_\omega)$. Therefore, we continue with the further analysis of the eigenvalues. For $0 < \omega \leq 1$ the following holds:

$$\lambda_1 = 1 - 2\omega \sin^2\left(\frac{\pi}{2n}\right) = 1 - 2\omega \sin^2\left(\frac{\pi h}{2}\right) \approx 1 - \frac{\omega \pi^2 h^2}{2}.$$

From this follows that λ_1 associated with the smoothest mode, also called the near null space of A, will always stay close to 1. This implies that no value of ω can effectively diminish the smooth components of the error. Furthermore, the reduction of the grid spacing $h = \frac{1}{n}$ brings λ_1 only closer to 1. Nevertheless, in [6] it is pointed out that the value of ω is $\frac{2}{3}$ is optimal for the damping of smooth components. To be more precise, for $\frac{n}{2} \geq l \geq n-1$, $|\lambda_l| < \frac{1}{3}$ when $\omega = \frac{2}{3}$.

Although Example 3.1 illustrates only the inability of the damped Jacobi method to reduce the smooth components of the error, the same conclusion holds for other basic iterative methods.

3-1-3 Algebraic Smoothness

Before introducing the concept of algebraic smoothness, we state several important definitions needed in this section.

Definition 3.3. The matrix A is irreducible if and only if there is no matrix P such that PAP^T is a block upper triangular matrix.

Definition 3.4. The matrix A with entries a_{ij} is irreducibly diagonal dominant if and only if A is irreducible and

$$|a_{ii}| \geq \sum_{j=1, j\neq i}^{n} |a_{ij}| \quad \forall i = 1, \ldots, n \tag{3.9}$$

with strict inequality for at least one i.

Definition 3.5. The matrix A is an M-matrix if and only if

1. the diagonal entries of A are positive;
2. the non-zero off-diagonal entries of A are negative;
3. A is non-singular;
4. $A^{-1} \geq 0$.

An equivalent, but more useful in practice definition of the M-matrix property stated [41] is provided below.

Definition 3.6. The matrix A is an M-matrix if it satisfies the following properties:

1. the diagonal entries of A are positive;
2. the non-zero off-diagonal entries of A are negative;
3. A is irreducibly diagonally dominant.

The algebraically smooth error is defined loosely as any error that is not reduced effectively by smoothing. To provide a deeper insight into this definition we focus again on the damped Jacobi method. We assume that A of the linear system (3.1) is an M-matrix.
The damped Jacobi method approximates the solution as follows:

$$\mathbf{v}^{k+1} = \mathbf{v}^k + \omega D^{-1}(\mathbf{f} - A\mathbf{v}^k),$$

where $\mathbf{v}$ is the iterative approximation of the exact solution and D is the diagonal of A.
To ensure the convergence, the damping factor ω is set to $\alpha||D^{-1/2}AD^{1/2}||^{-1}$ for $\alpha \in (0, 2)$.
The error propagation for this iteration is

$$\mathbf{e}^{k+1} = (I - \omega D^{-1}A)\mathbf{e}^k. \tag{3.10}$$

At the point when the convergence of the Jacobi method starts to decrease significantly, the error is defined as algebraically smooth. This implies that the size of $\mathbf{e}^{k+1}$ is insufficiently less than the size of $\mathbf{e}^k$. A natural measure of size in this case is the A-norm:

$$||\mathbf{e}||_A = (A\mathbf{e}, \mathbf{e})^{1/2}.$$

According to Equation (3.10), algebraically smooth error satisfies

$$||\mathbf{e}||_A \approx ||(I - \omega D^{-1}A)\mathbf{e}||_A. \tag{3.11}$$

Using the definition of the A-norm, it can be easily shown that

$$||(I - D^{-1}A)\mathbf{e}||_A^2 = ||\mathbf{e}||_A^2 - 2\omega||A\mathbf{e}||_{D^{-1}}^2 + \omega^2||D^{-1}A\mathbf{e}||_A^2. \tag{3.12}$$

Combining Equation (3.2), (3.11) and (3.12) yields

$$||\mathbf{r}||_{D^{-1}} \ll ||\mathbf{e}||_A.$$

In the component form this expression is equivalent to

$$\sum_{i=1}^{n} \frac{r_i^2}{a_{ii}} \ll \sum_{i=1}^{n} r_i e_i.$$

Therefore, at least on average, $|r_i| \ll a_{ii}|e_i|$ for the algebraically smooth error. We can write this condition as

$$A\mathbf{e} \approx \mathbf{0},$$

what is equivalent to saying that smooth error has relatively small residuals.

Example 3.2. The following differential equation is examined in [37]:

$$-(au_x)_x - (bu_y)_y + cu_{xy} = f(x, y),$$

with homogeneous Dirichlet boundary conditions. The domain is two dimensional unit square. The problem is discretized using a uniform grid and finite difference approximations. The locally constant coefficients a, b and c are defined in the unit square as follows:

a = 1
b = 1000
c = 0
	a = 1
	b = 1
	c = 2

a = 1
b = 1
c = 0
	a = 1000
	b = 1
	c = 0

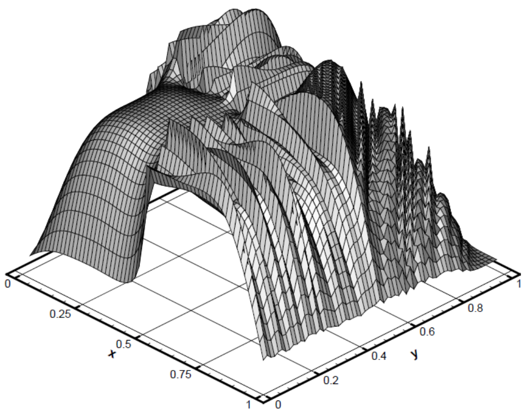


Figure 3.2: Error that is smooth in the algebraic sense, but is geometrically oscillatory (source: [37]).

The source function is equal to zero and the initial guess is chosen randomly. After having applied several smoothing steps, the iteration stalls and the algebraically smooth error on the finest level looks as shown in Figure 3.2. According to our definition this error is algebraically smooth, but is geometrically oscillatory on three of the four quadrants.

3-1-4 Influence and Dependence

Since AMG was originally developed for M-matrices, the ith equation is associated with the ith unknown due to the diagonal dominance. That is, the ith equation is used to determine the value of u_i. Obviously, all the equations are needed to determine any variable precisely. However, not all the variables in an equation are evenly important. If in the ith equation the coefficient a_{ij} is large relative to the other coefficients, then a small change in u_j will have more effect on u_i than a small change in other variables. This leads us to the following definition.

Definition 3.7. The variable u_i (corresponding to grid point i) strongly depends on the variable u_j (corresponding to point j) if

$$-a_{ij} \geq \theta \max_{k \neq i} \{-a_{ik}\}, \tag{3.13}$$

where the strength threshold θ satisfies $0 < \theta \leq 1$.

Remark 2. It is important to understand that the use of the above definition is not limited to M-matrices. If positive off-diagonal entries are present, they are considered to be weak connections.

The choice of the strength threshold is important, since θ can influence the stencil size and the convergence of the algorithm significantly, [44]. Generally, this parameter is set to 0.25, [1].

Having introduced the concept of dependence, we provide the definition of influence.

Definition 3.8. If the variable u_i strongly depends on the variable u_j, then the variable u_j strongly influences the variable u_i.

From these two definitions follows that the variable u_j strongly influences u_i, if the coefficient a_{ij} is comparable in magnitude to the largest off-diagonal element in the ith equation.

Definition 3.9. If the variable u_i strongly influences/depends on variable u_j, then u_i and u_j are strongly connected.

Within the AMG algorithm, the strength of connection between the variables is usually stored in the strength-of-connection matrix, which is usually denoted by S. As will be shown in following sections, the coarsening and interpolation processes are based on the fundamental concepts of algebraic smoothness and influence.

3-1-5 Coarsening Heuristics

The coarsening process describes the construction of a partitioning of the indices $\{1, 2, \ldots, n\} = C \cup F$ into coarse grid and fine grid variables, also known as C- and F-points. Here, a variable is called a fine grid variable, if it is not present on the coarse grid. In contrast, a C-point exists on the fine and coarse grid.
The classical (also called Ruge-Stüben or simply RS) coarsening process is based on the following two heuristic criteria:

H 3.1. For each F-point i, every point j that strongly influences i either should be a C-point or should strongly depend on at least one C-point that strongly influences i.

H 3.2. The set of coarse points C should be a maximal independent subset of all points, i.e. no two C-points are connected to each other, and no other C-point can be added without losing the independence.

Criterion **H 3.1** is designed to control the quality of interpolation. We will explore this heuristics in more detail in Section 3-1-7, where the interpolation formula is introduced.

Criterion **H 3.2** aims to control the multigrid efficiency, which is defined by the computational costs per cycle and convergence factor. If the coarse grid is a large fraction of the fine grid, the interpolation of smooth errors is generally very accurate and therefore leads to a faster convergence. However, the costs of doing a V-cycle increase drastically. Criterion **H 3.2** bounds the size of the coarse grid by prohibiting the strong dependence of C-points on one another and therefore moving the coarse grid elements farther apart. The heuristic controls the convergence by requiring C to be a maximal subset.

Generally, it is not possible to fulfill both criteria simultaneously. Since criterion **H 3.1** is important for the interpolation quality, **H 3.1** is enforced, while **H 3.2** is used as a guideline.

3-1-6 Coarsening

Although the coarsening procedure is based solely on the strength-of-connection operator S, it is difficult to get an intuitive understanding of this procedure from a purely matrix point-of-view. Therefore, we provide an explanation which uses the adjacency graph corresponding to S.

Before the coarsening (or coloring) algorithm is described, we make two definitions.

Definition 3.10. Define S_i as the set of points that strongly influence i.

Definition 3.11. Define S_i^T as the set of points that strongly depend on i.

The coarse grid selection algorithm consists of two passes. The first pass creates a set of C-points that have good approximation properties and tend to satisfy heuristic **H 3.2**. Once the initial coarse grid points have been assigned according to the first pass, the second pass changes some F-points to C-points to enforce **H 3.1**.

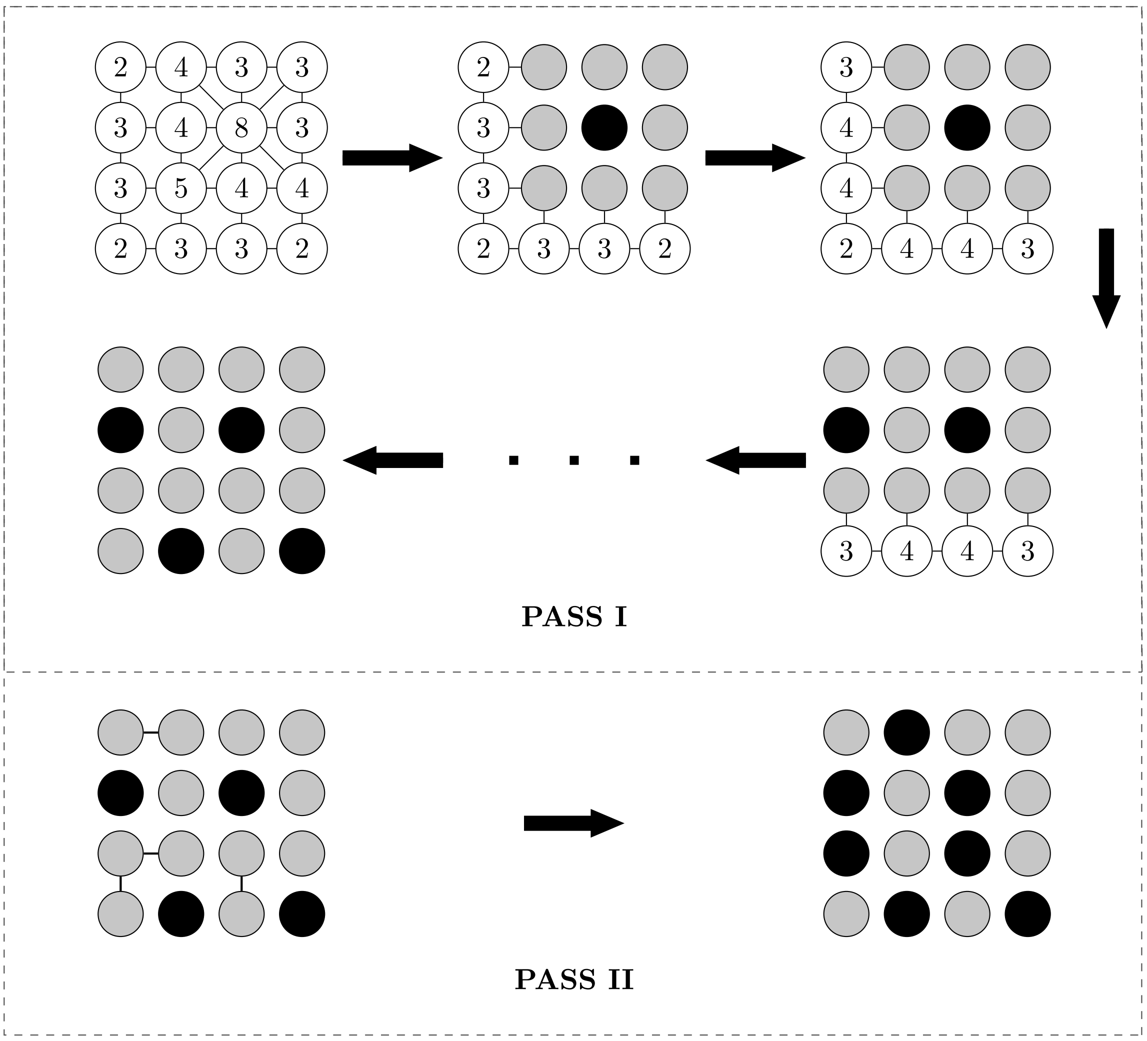


Figure 3.3: PASS I and II of the coloring process. Black points represent C-points, gray points denote F-points, and white points denote unassigned points. The numbers correspond to the values of measure λ. In PASS II solid lines represent strong F-F connections that do not satisfy heuristic **H 3.1**.

The process begins by assigning to each point i a measure λ_i, that represents its potential to become a coarse grid point. The simplest approach to make this assignment is to define the measure as the number of points strongly influenced by i. This number is the cardinality of S_i^T. When the measures are assigned, we select the point with maximum λ_i value to be the first C-point.

There are several points that strongly depend on the first C-point. Hence, the interpolation formula for each of these point should include this first C-point. For this reason, the points that the first coarse grid point strongly influences, should become F-points. The whole set S_i^T can be added to the set of F-points, without any violations of criterion **H 3.1** or potential violations of **H 3.2**. Since the points strongly depended on these new F-points can be useful for an accurate interpolation, it is logical to consider them as potential C-points. Therefore, for each new F-point j in S_i^T, the measure λ_k of each unassigned point k on which j strongly depends, is incremented. That is, the measures of unassigned points k in S_j are incremented by 1 for each fine grid point j. This process continues until all points have become either fine or coarse grid points.

Since the first pass does not guarantee to fulfill criterion **H 3.1**, it is followed by the second pass. This pass examines all strong connections between fine grid points for common coarse neighbours. If **H 3.1** is not satisfied, new points are added to the set C. It should be noted that there are many problems that do not require the second pass, such as a five- and nine-point Laplacian stencil on a uniform grid. Figure 3.3 illustrates a general example where both passes are needed.

The coarsening process depends on the order in which the points are scanned while seeking for the next point with maximal measure. For example, in many cases there are several grid points with the maximal value of λ at the start, and any of them can be selected as the first C-point. Obviously, also during the coloring process, there can arise several points with the same maximal measure. Thus, there are many possible coarsenings. The heuristics **H 3.1** and **H 3.2** ensure that all the possible coarse grids will provide an adequate representation of smooth error components and keep the size of the coarse grid relatively small.

Example 3.3. Consider an anisotropic Poisson problem

$$-\epsilon u_{xx} - u_{yy} = f,$$

where ϵ is very small. The grid is uniform. This problem shows strong dependence in the y-direction and little dependence in the x-direction. The coloring produced in this case is shown in Figure 3.4.

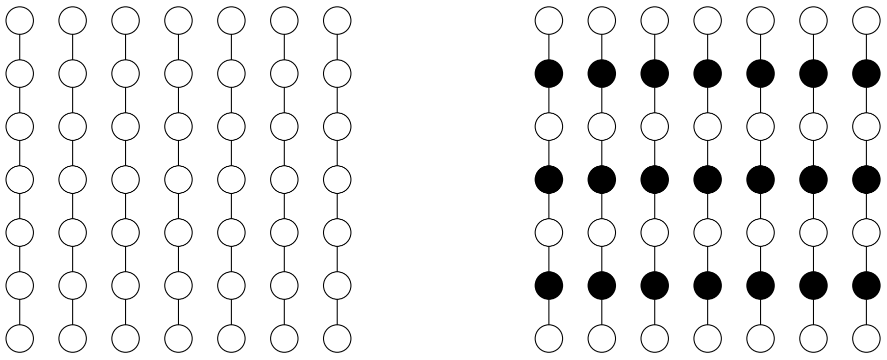


Figure 3.4: Semicoarsening in the y-direction produced for an anisotropic Poisson problem (Example 3.3). On the left, the original grid is shown. Solid lines denote strong dependences. On the right, the final coarsening is illustrated.

The coloring scheme generates a semicoarsened grid, because there is no coarsening in the x-direction. Thus, AMG performs coarsening only in the direction of strong dependence.

Although for Example 3.2 the main idea behind the semicoarsening stays the same, it obtains a more complicated form (see Figure 3.5). In this case there is strong dependence in the

y-direction ($b = 1000$) in the upper left quadrant. Therefore, AMG preliminary coarsens that part of the grid in the y-direction. Analogously, in the lower right quadrant where the dependence is strong in the x-direction ($a = 1000$), the grid is coarsened in the x-direction. In the upper right quadrant, where $a = b = 1$ and $c = 2$ the coarsening is performed in the northwest direction. Finally, in the lower left quadrant the coarsening is standard, because there is no anisotropy in that region.

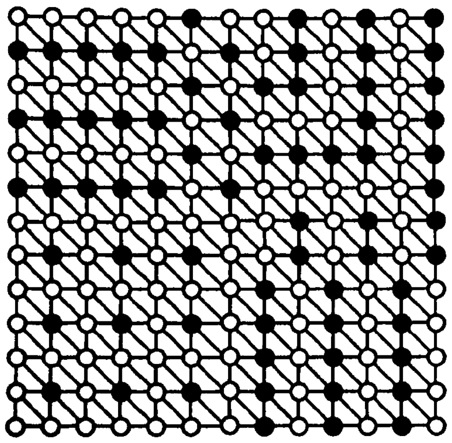


Figure 3.5: Semicoarsening of Example 3.2 (source: [37]).

3-1-7 Interpolation

Suppose that $\{e_i : i \in C\}$, is a set of values on the coarse grid corresponding to a smooth error. This set has to be interpolated to the fine grid. When the grid is unstructured or does not exist, it can be challenging to find an accurate interpolation operator. The goal of this section is to provide a suitable approach for the grid transfer.

If a fine grid point i strongly depends on a coarse grid point j, then the value e_j contributes considerably to the value of e_i in the ith fine grid equation. Therefore, the value e_j in the coarse grid equation is a reasonable candidate for the interpolation formula in order to accurately approximate e_i on the fine grid.
To strengthen this idea, we recall from Section 3-1-3 that for a symmetric M-matrix A the following inequality holds:

$$||\mathbf{r}||_{D^{-1}} \ll ||\mathbf{e}||_A.$$

Using this expression and the Cauchy-Schwarz inequality, we have

$$||\mathbf{e}||_A^2 = (A\mathbf{e}, \mathbf{e}) = (D^{-1/2}A\mathbf{e}, D^{1/2}\mathbf{e}) \leq ||D^{-1/2}A\mathbf{e}||\,||D^{1/2}\mathbf{e}|| = ||\mathbf{r}||_{D^{-1}}\,||\mathbf{e}||_D \ll ||\mathbf{e}||_A\,||\mathbf{e}||_D.$$

Thus, assuming that A has a zero row sum

$$\begin{aligned}||\mathbf{e}||_A = \sum_{ij} a_{ij}e_ie_j &= \frac{1}{2}\sum_{ij}(-a_{ij})(e_i - e_j)^2 + \sum_{ij} a_{ij}e_i^2 \\ &= \frac{1}{2}\sum_i \left(\sum_{j\neq i} |a_{ij}|(e_i - e_j)^2 + a_{ii}(e_i - e_j)^2 \right) \\ &= \sum_i \sum_{j\neq i} |a_{ij}|(e_i - e_j)^2 \ll ||\mathbf{e}||_D = \sum_i a_{ii}e_i^2.\end{aligned}$$

From this follows, that for most i holds

$$\sum_{j\neq i} \left(\frac{|a_{ij}|}{a_{ii}}\right)\left(\frac{e_i - e_j}{e_i}\right)^2 \ll 1, \quad 1 \leq i \leq n. \tag{3.14}$$

The left side of Equation (3.14) is a sum of products of two nonnegative components. Since the products are very small, one or both of the terms in each product must be small. However, if e_j strongly influences e_i, off-diagonal element $-a_{ij}$ can be comparable to the diagonal element a_{ii}. This means that for these e_j's holds $e_j \approx e_i$, because $e_j - e_i$ is small. With other words, smooth error varies slowly in the direction of strong dependence. We can conclude that the fine grid quantity u_i can be interpolated from the coarse grid quantity u_j if i strongly depends on j.

Remark 3. In order to make smooth error more oscillatory on a coarser grid, AMG performs coarsening in the direction of strong dependence. This is exactly what we observed in Example 3.3.

Remark 4. A slightly different manner to show that $e_j \approx e_i$ is provided in [14]. The alternative derivation is based on the fact that geometrically smooth functions are in the near null space of the fine grid operator.

To define the interpolation operator P, we introduce the neighbourhood of i, N_i: for each fine grid point i, N_i is the set of all points $j \neq i$ such that $a_{ij} \neq 0$. N_i contains three types of points:

- the neighbouring C-points that strongly influence i; these points form *the coarse interpolatory set* for i, denoted by C_i;
- the neighbouring F-points that strongly influence i; this set is denoted by D_i^s; and
- the neighbouring C- and F-points that do not strongly influence i; they form the set of *weakly connected neighbours*, D_i^w.

For the interpolation formula we require the ith component of $P\mathbf{e}$ to be defined as follows

$$(P\mathbf{e})_i = \begin{cases} e_i & \text{if } i \in C, \\ \sum_{j\in C_i} \omega_{ij}e_j & \text{if } i \in F, \end{cases} \tag{3.15}$$

where the interpolation weights, ω_{ij} have to be determined.

From Section 3-1-3, we know that smooth error has relatively small residuals: $A\mathbf{e} \approx \mathbf{0}$. In the component form, this condition is equivalent to:

$$a_{ii}e_i \approx -\sum_{j\in N_i} a_{ij}e_j.$$

The sum on the right hand side can be splitted into a sum over C_i, D_i^s and D_i^w:

$$a_{ii}e_i \approx -\sum_{j\in C_i} a_{ij}e_j - \sum_{j\in D_i^s} a_{ij}e_j - \sum_{j\in D_i^w} a_{ij}e_j. \tag{3.16}$$

From Equation (3.15) follows that to obtain the interpolation weights, we need to replace the e_j with $j \in D_i^s$ and $j \in D_i^w$ either by e_i or e_j with $j \in C_i$.
If the points are weakly connected to i we distribute e_j to the diagonal coefficients. In other words, if $j \in D_i^w$, we replace e_j by e_i. By doing so and redistributing the terms, we obtain:

$$\left(a_{ii} + \sum_{j\in D_i^w} a_{ij}\right) e_i \approx -\sum_{j\in C_i} a_{ij}e_j - \sum_{j\in D_i^s} a_{ij}e_j. \tag{3.17}$$

This approach is easily justified. Assume that the dependence has been underestimated: the points in D_i^s strongly influence e_i. Then $e_j \approx e_i$, because smooth error varies slowly in the direction of strong dependence. Therefore, the replacement is a logical step. Alternatively, we can assume that e_i has no strong dependence on the points of D_i^s. This means that the corresponding value of a_{ij} is relatively small. Thus, the possible error created by the distribution to the diagonal is insignificant.

The e_j's in the third sum of Equation (3.16) can also be distributed to the diagonal. However, experience has shown that it is better to distribute these terms to C_i [6]. The main idea behind this distribution is to approximate e_j with $j \in D_i^s$ with a weighted sum of e_k, where $k \in C_i$. This can be done by making the following approximation:

$$e_j \approx \frac{\sum_{k\in C_i} a_{jk}e_k}{\sum_{k\in C_i} a_{jk}}. \tag{3.18}$$

The choice of the numerator is based on the fact that the e_j strongly depends on the e_k's in proportion to a_{jk}. The denominator assures that constants are interpolated exactly by Equation (3.18). It should be noted that if there is strong F-F connection between points i and j, then there must exist at least one point common to their interpolatory sets.

Substituting Equation (3.18) into Equation (3.17) and interchanging sums in the last term, gives us the interpolation weights:

$$\omega_{ij} = -\frac{a_{ij} + \sum_{m\in D_i^s}\left(\frac{a_{im}a_{mj}}{\sum_{k\in C_i} a_{mk}}\right)}{a_{ii} + \sum_{n\in D_i^w} a_{in}}. \tag{3.19}$$

This completes the interpolation formula (3.15).

At this point we have all the information required for the motivation of criterion **H 3.1** from Section 3-1-5 that says that for each F-point i, every point j that strongly influences i either must be a C-point or must strongly depend on at least one C-point that strongly influences i. Approximation (3.18) is applicable to F-points strongly influencing the F-point i, so to achieve an accurate interpolation, these points should be represented in the interpolation formula. In contrast to coarse grid points, they contribute to the interpolation formula by distributing their values to points in C_i as illustrated in Equation (3.18). Obviously, Equation (3.18) will be more precise if there would be several points in C_i that strongly influence j. However, the necessary condition for the approximation is that there is at least one point in C_i on which j strongly depends. This requirement is enforced by **H 3.1**.

3-1-8 Galerkin Operators

Once the coarse grid $k+1$ is constructed and the interpolation operator is defined, the restriction operator R is easily obtained:

$$R^k = (P^k)^T. \tag{3.20}$$

The coarse grid operator is then constructed using Galerkin condition:

$$A^{k+1} = R^k A^k P^k. \tag{3.21}$$

This definition has certain pros and cons. On the one hand, it guarantees the convergence of AMG under the assumption that A^k is symmetric positive definite and the smoothing step is convergent. On the other hand, the Galerkin approach creates coarse grid operators with high density.

3-1-9 Algorithm

Assume that the superscript 0 and M correspond to the finest and coarsest level, respectively. We denote the grids by $\Omega^0 \supset \Omega^1 \supset \Omega^2 \supset \cdots \supset \Omega^M$. Furthermore, for every $m \in \{0, 1, \ldots, M-1\}$ grid Ω^m is partitioned into a set of fine points F^m and a set of coarse points C^m, such that $F^m \cap C^m = \{\emptyset\}$. To transfer grid $m+1$ to a finer grid m, the prolongation operator P^m from Equation (3.15) is used. The inverse transformation is obtained by applying the restriction operator R^m, which is defined by (3.20). The coarse grid operator A^{m+1} is obtained using the Galerkin condition (3.21). These multigrid components are constructed in the setup phase, as shown in Algorithm 3.

Algorithm 3: AMG setup phase.

```
Data: $m = 0$, $\Omega^m, A^m$
1 while $\Omega^0$ is not sufficiently small do
2     Split $\Omega^m$ into $C^m$ and $F^m$.
3     Set $\Omega^{m+1} = C^m$.
4     Define interpolation $P^m$.
5     Define restriction $R^m$.
6     Define coarse grid operator $A^m$.
7     Set up smoother $S^m$, if necessary.
8     Set $m = m + 1$.
9 end
```

When the setup phase is completed, the method proceeds with the solve phase which repeats a certain cycle until the termination criterion is reached. A possible cycle is shown in Algorithm 4. In the algorithm, the smooth error on level m is defined as $\mathbf{e}^m = \mathbf{u} - \mathbf{v}^m$, where $\mathbf{u}$ and $\mathbf{v}^m$ are the exact solution and its approximation on mth level, respectively. This error is eliminated by defect correction, which solves the residual equation on a coarser grid $m+1$ and interpolates the error back to grid m in order to improve the numerical solution.

Algorithm 4: AMG solve phase.

```
Data: $A^m, R^m, P^m, S^m, u^m, f^m$
while $m \leq M$ do
    if $m$=$M$ then
        Solve $A^M\mathbf{v}^M = \mathbf{f}^M$ directly.
    else
        Apply smoother $S^m$ $\mu_1$ times to $A^m\mathbf{v}^m = \mathbf{f}^m$.
           Perform defect correction:
              Set $\mathbf{r}^m = \mathbf{f}^m - A^m\mathbf{v}^m$.
              Set $\mathbf{r}^{m+1} = R^m\mathbf{r}^m$.
              Apply AMG solve phase to $A^{m+1}\mathbf{e}^{m+1} = \mathbf{r}^{m+1}$.
              Interpolate $\mathbf{e}^m = P^m\mathbf{e}^{m+1}$.
              Correct the solution: $\mathbf{v}^m \leftarrow \mathbf{v}^m + \mathbf{e}^m$.
        Apply smoother $S^m$ $\mu_2$ times to $A^m\mathbf{v}^m = \mathbf{f}^m$.
    end
end
```

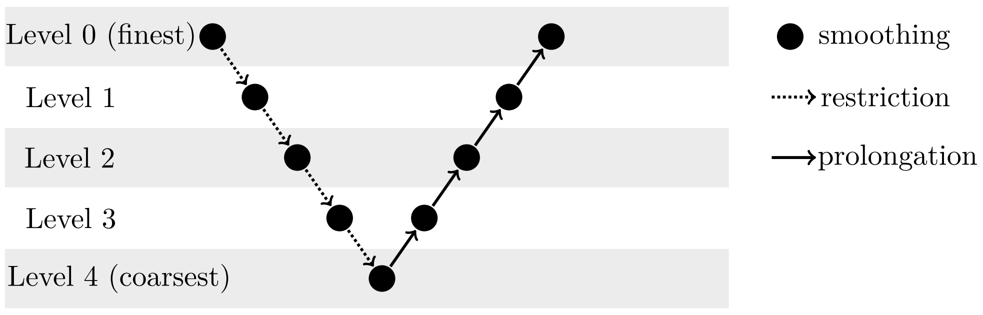


Figure 3.6: Multigrid V-cycle on five levels.

Depending on the order in which the grid levels are visited, it is possible to categorize the V-, W-cycle and full multigrid cycle (FMG). Algorithm 4 describes a $V(\mu_1, \mu_2)$-cycle. It starts at the finest grid level, where a presmoothing procedure, consisting of μ_1 steps, is performed. At that point, the residual is calculated and transfered to the next coarser level, where another μ_1 presmoothing steps take place. This process is repeated until the coarsest grid is reached. After that, the correction is interpolated back to a finer grid and μ_2 postsmoothing steps are performed. The cycle stops when the finest grid level is attained. For $M = 4$ a V-cycle is schematically illustrated in Figure 3.6.

Figure 3.7 shows a W-cycle and an FMG. Obviously, W-cycles weight coarse grids more heavily than V-cycle. Therefore, for complicated problems a W-cycle can be more efficient

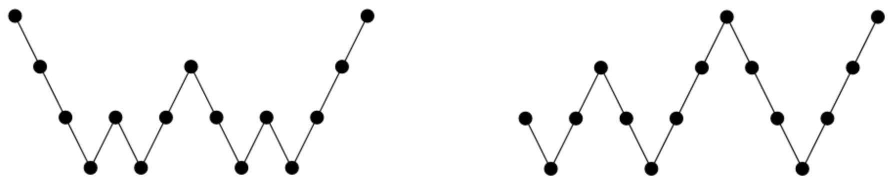


Figure 3.7: W- and FMG cycles for a four grid method.

and robust. The cost to find solution using an FMG cycle for a large problem is proportional to the number of grid elements on the finest grid. Thus, full multigrid cycles are asymptotically optimal.

3-1-10 Convergence

In [30] a description of the convergence of the V-cycle with respect to the A-norm is presented. The analysis is carried out under the assumption that the fine grid matrix is symmetric and positive definite (SPD).

Definition 3.12. A symmetric matrix $A \in \mathbb{R}^{n\times n}$ is positive definite if $\mathbf{x}^T A\mathbf{x} > 0$ for all non-zero vectors $\mathbf{x} \in \mathbb{R}^n$.

For the coarse level m, the error propagation operator of the defect correction is denoted by T^m. Using the notation introduced in Section 3-1-9 and line 7 to line 11 of Algorithm 4, we compute T^m. The approximation of the exact solution at iteration $k+1$ of the defect correction process is given by

$$\begin{aligned}\mathbf{v}_{k+1}^m &= \mathbf{v}_k^m + \mathbf{e}_k^m \\ &= \mathbf{v}_k^m + P^{m+1}\left(A^{m+1}\right)^{-1} R^{m+1}\left(\mathbf{f}^m - A^m\mathbf{v}_k^m\right) \\ &= \mathbf{v}_k^m + P^{m+1}\left(A^{m+1}\right)^{-1} R^{m+1}\mathbf{r}_k^m.\end{aligned}$$

Therefore, the error at step $k+1$ becomes

$$\begin{aligned}\mathbf{e}_{k+1}^m &= \mathbf{u}^m - \mathbf{v}_{k+1}^m \\ &= \mathbf{u}^m - \mathbf{v}_k^m - P^{m+1}\left(A^{m+1}\right)^{-1} R^{m+1}\mathbf{r}_k^m \\ &= \mathbf{e}_k^m - P^{m+1}\left(A^{m+1}\right)^{-1} R^{m+1}A^m\mathbf{e}_k^m \\ &= \left(I^m - P^{m+1}\left(A^{m+1}\right)^{-1} R^{m+1}A^m\right)\mathbf{e}_k^m.\end{aligned}$$

Thus, $T^m = I^m - P^{m+1}\left(A^{m+1}\right)^{-1} R^{m+1}A^m$.
In Section 3-1-2 it shown that the error propagation matrix corresponding to the smoother S^m is equal to $I^m - (M^m)^{-1}A^m$, where M is defined by the relaxation method, e.g. M is equal to the diagonal of A in case of Jacobi.

We denote the error propagator for AMG by E_{ALG}, where ALG refers to a given algorithm. For instance, for the two grid method with ν pre- and post-smoothing steps, we obtain

$$\begin{aligned} E_{TG} &= \left(S^0\right)^\nu T^0 \left(S^0\right)^\nu \\ &= \left(I^0 - \left(M^0\right)^{-T} A^0\right)^\nu \left(I^0 - P^1 \left(K^1\right)^{-1} R^1 A^0\right) \left(I^0 - \left(M^0\right)^{-1} A^0\right)^\nu, \end{aligned} \tag{3.22}$$

where K^1 represents the coarse grid matrix, i.e. $K^1 = A^1$.
The associated convergence factor $\rho\left(E_{ALG}\right)$ is defined as in Equation (3.7).

Theorem 3.1. *Define A, S^m, T^m and $\mathbf{e}^m$ as above. Let $A > 0$. Assume that the interpolation operators have full rank and that the restriction and coarse grid operators are defined as in Section 3-1-8. Moreover, suppose that, for all $\mathbf{e}^m$,*

$$||S^m \mathbf{e}^m||_A^2 \leq ||\mathbf{e}^m||_A^2 - \delta ||T^m \mathbf{e}^m||_A^2 \tag{3.23}$$

holds with some $\delta > 0$ independently of $\mathbf{e}^m$ and m. Then $\delta \leq 1$. In addition, provided that the coarsest grid equation is solved and that at least one smoothing step is performed after each defect correction step, the V-cycle to solve $A\mathbf{u} = \mathbf{f}$ has a convergence factor $\rho\left(E_{V(0,\nu)}\right)$ bounded above by $\sqrt{1-\delta}$.

Proof: see (Theorem 3.1, [30]).

Remark 5. The condition stated in Equation (3.23) means that error components $\mathbf{e}^m$ that cannot efficiently be reduced by defect correction, i.e. $||T^m \mathbf{e}^m||_A \approx ||\mathbf{e}^m||_A$, have to be effectively reducible by the smoothing process. At the same time, for components that can be efficiently reduced by the defect correction procedure, i.e. for which $||T^m \mathbf{e}^m||_A \ll ||\mathbf{e}^m||_A$, the smoothing process is allowed to be ineffective.

Theorem 3.2. *Consider the assumptions made in Theorem 3.1. If the condition stated in Equation (3.23), is replaced by*

$$||S^m \mathbf{e}^m||_A^2 \leq ||\mathbf{e}^m||_A^2 - \delta ||T^m S^m \mathbf{e}^m||_A^2,$$

the V-cycle convergence factor is bounded above by $\frac{1}{\sqrt{1+\delta}}$ if at least one smoothing step is performed before each defect correction step.

Proof: see (Remark 3.2, [30]).

In the context of AMG, convergence is not a suitable measure for the performance of the method, if numerical work is not taken into account. Since nothing definite is known a priori about the the size of the coarse grids and the dependence of the coarse grid variables on each other, it is very important to find the conditions that are not only beneficial in terms of convergence, but also allow to control the costs of the algorithm.

3-1-11 Costs of AMG

Since the ratio of C- and F-points is not known in advance, the costs of AMG cannot be predicted in terms of the floating-point operation counts and storage. Therefore, we introduce two simple tools to analyse the performance of the algorithm.

Definition 3.13. Geometric complexity is the total number of grid points, on all levels, divided by the number of grid points on the finest level.

Definition 3.14. Algebraic complexity is the total number of non-zero elements, in the fine grid discretization matrix A^0 and all the coarse grid operators, A^m with $m \in \{0, 1, \ldots, M\}$, divided by the number of non-zero entries in the fine grid operator.

Geometric complexity, also known as grid complexity, indicates the storage space required for the right hand side and approximation vectors.
Algebraic complexity provides an accurate approximation of the storage of all A^k operators. In addition, it gives a reasonable indication of the expense of the AMG V-cycle: the costs in solve phase of the algorithm are dominated by relaxation and the computation of the residual, which are directly proportional to the number of non-zero elements in the operator and, therefore, to the algebraic complexity.

3-2 Coarse Grid Selection

There are two fundamentally different types of coarsening schemes. The first type splits all points into two disjoint sets: C-points that will form the grid on the next level and F-points that will be interpolated by the C-points. The second type, known as coarsening by aggregation, accumulates aggregates, which are disjoint subsets of the variable set and correspond to exactly one coarse grid point.

3-2-1 Serial Coarsening Strategies

Classical Ruge-Stüben coarsening algorithm introduced in Section 3-1-6, belongs to the first type and is serial, [44]. Before describing any parallel approaches, we outline the structure of two other serial methods, because sequential schemes form the basis for the majority of parallel algorithms.

Aggressive Coarsening

For many PDEs, the application of classical AMG leads to surprisingly large Galerkin operators on the first coarse grid, but performs relatively efficiently on all the other levels. For example, in [37], where isotropic 7-point stencils on regular 3D meshes are concidered, it is shown that the Galerkin matrix on the first coarse grid is larger, than the fine grid matrix by the factor of 1.36. Since the first coarse level significantly contributes to the final complexity,

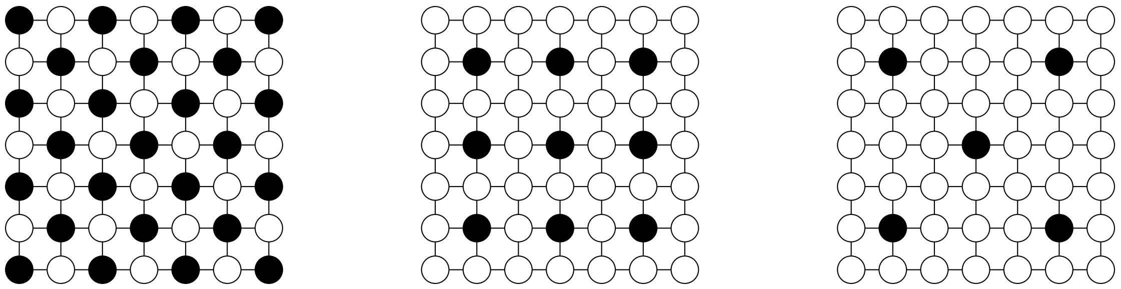


Figure 3.8: Aggressive coarsening applied to 5-point isotropic Laplacian: A2 coarsening (midden), A1 coarsening (right). On the left, the intermediate step is shown.

it is important to reduce the number of non-zero elements on this level. This can be done efficiently by using aggressive coarsening, which is based on the standard coarsening scheme.

Aggressive coarsening adapts the concept of strong connection by introducing long-range strong n-connection.

Definition 3.15. A variable i is strongly n-connected to a variable j along a path of length l, if there exists a sequence of variables $i_0 = i, i_1, \ldots, i_l = j$ with i_k strongly connected to i_{k+1} for $k = 0, 1, \ldots, l-1$.

Definition 3.16. A variable i is strongly n-connected with respect to (p, l) to a variable j, if at least p paths of lengths at most l exist such that i is strongly n-connected to j along each of these paths.

Analogously to the set S_i in Section 3-1-6, we define the set $S_i^{(p,l)}$:

Definition 3.17. $S_i^{(p,l)}$ is the set of points to which variable i is strongly n-connected with respect to (p, l).

With other words, for any p and l, aggressive coarsening can be applied by replacing S_i by $S_i^{(p,l)}$ in the standard coarsening scheme and omitting its second pass. However, the use of $S_i^{(p,l)}$ requires the storage of the complete connectivity information for each point i. To prevent this, we consider a different approach that also leads to aggressive coarsening. Essentially, aggressive coarsening can be applied by using the first pass of the coloring scheme twice. First, it is used exactly as described in Section 3-1-6. Then, the first pass is applied to $S_i^{(p,l)}$ which comprises only the resulting C-points.
It should be noted that aggressive coarsening is normally used only on the first coarse level. For this reason, the memory requirements for this strategy stay low.

Aggressive A2 coarsening, i.e. choosing $p = 2$, $l = 2$, and aggressive A1 coarsening ($p = 1$, $l = 2$) are the most common in practical use, [37]. Figure 3.8 illustrates A1 and A2 coarsening of 5-point isotropic Laplacian on a 7×7 grid.

Coarsening by Aggregation

Coarsening by aggregation is fundamentally different from the standard and aggressive coarsening. It is defined by building aggregates.

Definition 3.18. An aggregate consists of a root point i and its neighbourhood, i.e. all points j such that $|a_{ij}| > \theta\sqrt{|a_{ii}a_{jj}|}$.

Exactly one coarse grid variable is associated with each aggregate. The aggregation scheme consists of two passes. In the first pass, a root point is selected that is not adjacent to any aggregate. After that, the aggregate corresponding to the chosen root point is built. This procedure is repeated until all points are adjacent to an aggregate. The first pass is demonstrated in Figure 3.9. In the second pass, all unaggregated points are either used to create new aggregates or absorbed into already existing aggregates.

When this approach is applied, each F-variable is interpolated from exactly one C-variable (the root point). That is, although each fine grid point may have more than one connection to the set of coarse grid points, the sets of interpolatory variables are restricted to exactly one C-variable each. For more details see [28].

Coarsening by aggregation is generally fast and produces small complexities, [44]. However, the performance of the aggregation scheme is strongly depended on its second pass: creating too many aggregates can increase the complexity, while enlarging the existing aggregates can lead to slower convergence.

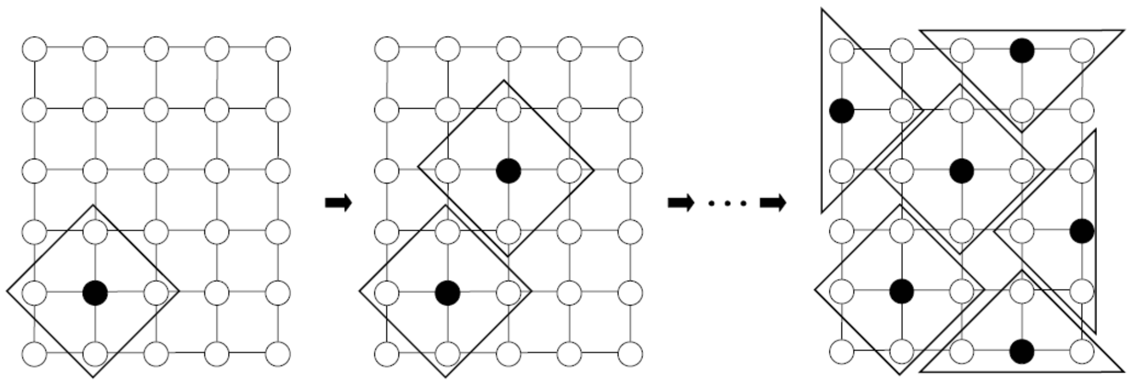


Figure 3.9: The first pass of the aggregation scheme applied to a 5-point Laplacian. Black points represent root points, boxes and triangles correspond with aggregates (source: [44]).

3-2-2 Parallel Coarsening Strategies

There exist various approaches to parallelize the coarse grid selection algorithms described previously. The simplest method, called RS0, starts by partitioning all variables in subdomains so that each processor corresponds to one subdomain. On each subdomain a serial

coarsening algorithm ia used, whereby the variables are treated independently of variables on other subdomains. With other words, the variables located on the processor boundaries are ignored.
Since this approach requires no communication between processors, it is highly efficient, but leads to poor convergence, [44]. Moreover, generally this approach violates heuristic **H 3.1** required for the standard and aggressive coarsening, and leads to large complexities if coarsening by aggregation is applied, [44].
A possible way to improve this strategy for standard and aggressive coarsening algorithm is by performing an extra pass only on the boundary variables which will ensure that condition **H 3.1** is satisfied by assigning additional coarse grid points. This algorithm is known as RS3. For coarsening by aggregation, we can start by generating aggregates on the boundaries to achieve better performance. Once there are no more unaggregated points adjacent to an aggregate on the processor boundaries, the method can proceed by choosing aggregates in the interior.

Cleary-Luby-Jones-Plassman

Cleary-Luby-Jones-Plassman (CLJP) coarsening begins by computing global measures as in classical approach, and then adding a random number between 0 and 1 to each measure. This procedure creates unique local maxima. The algorithm is demonstrated in Figure 3.10

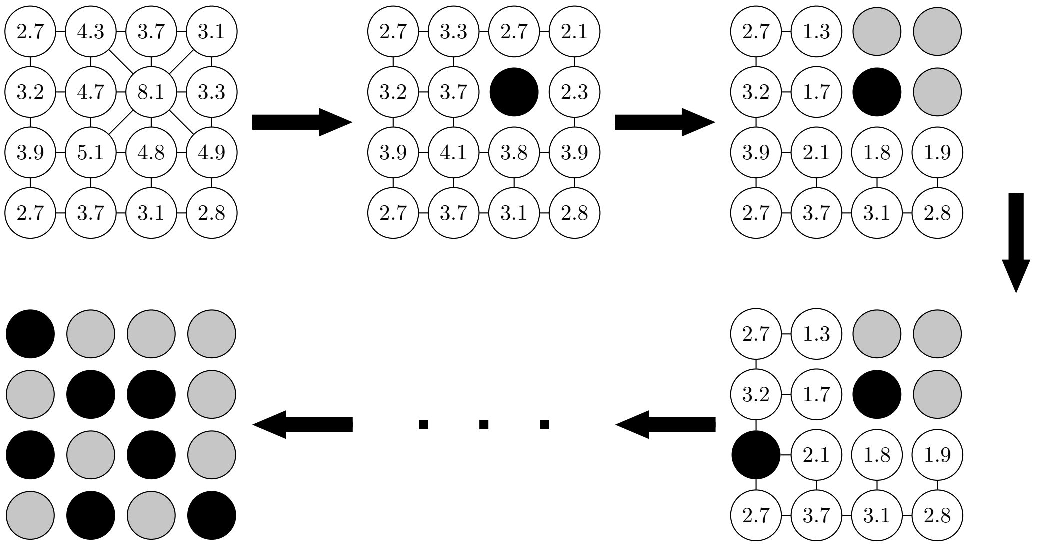


Figure 3.10: CLJP coarsening. Black points denote C-points, gray points are F-points, and white points are unassigned points. The numbers represent the sum of measure λ and a random number between 0 and 1.

Suppose that point i is a local maximum, then it is made a C-point and all connections to points j that are strongly connected to i, are removed. In addition, we subtract 1 from j's measure. This step increases the likelihood of the neighbours of i to become F-points, but does not make it an F-point instantly. After that, for each j, all points k that depend on j, are examined: if k also depends on i, we remove the edge connecting k and j, and decrement the measure of j. A point becomes an F-point, when its measure becomes smaller than 1.

One of the advantages of CLJP coarsening procedure is that it is completely independent of the number of processors, if the same set of global random numbers is used, [44]. Furthermore, not each processor has to contain a C-point. Therefore, coarsening does not slow down on coarser levels. The main disadvantages of this approach is that often it creates coarse grid point clusters and leads to high complexities (see [44] for an example).

Parallel Maximally Independent Set

The beginning of Parallel Maximally Independent Set (PMIS) algorithm is similar to that of CLJP: PMIS adds a random number between 0 and 1 to the global measures and choses the local maxima to be C-points. After that, all points that are influenced by coarse grid points are made F-points and are eliminated from the graph. This process continues till all variables are either F- or C-points.
To reduce the complexities, PMIS replaces criterion **H 3.1** by **H 3.3**.

H 3.3. For each F-point i there has to be at least one C-point j that strongly influences it.

Although the complexities of PMIS are significantly lower than those of CLJP, it is still independent of the number of processors, [44].

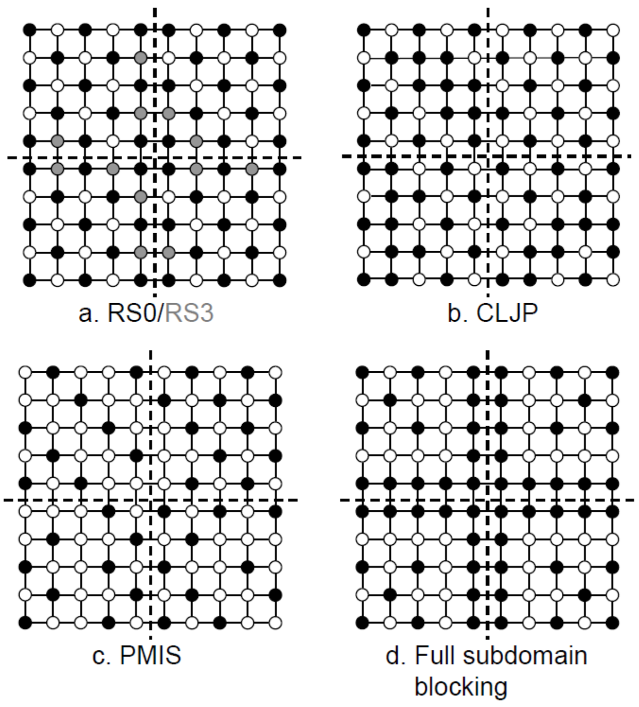


Figure 3.11: Parallel coarsening strategies applied to a 5-point Laplacian using 4 processors. White points represent F-points, black points are C-points and gray points are C-points obtained during special boundary treatments (source: [44]).

Subdomain Blocking

Subdomain blocking begins by coarsening the processor boundaries. Once the coarse grid points on the boundaries are selected, the method proceeds to coarsen inside the subdomains. For the interior of the subdomains, any serial approach can be used.
The subdomain blocking is called full, when all the boundary points are set to be C-points. This strategy produces too many coarse grid points on the boundaries, which may cause problems on the coarser grids and lead to high complexities. An alternative approach for the coarsening of the boundaries is to apply standard coarsening. This method is known as minimum subdomain blocking.

The difference in behaviour of the parallel coarsening strategies is illustrated in Figure 3.11. Undoubtedly, there exist other parallel strategies. For instance, in [23] classical coarsening is combined with CLJP to produce an efficient algorithm for structured problems. In fact, the majority of alternative parallel coarsening strategies is based on the algorithms described above. Since these approaches are not related to our project, we refer to [12] and [25] for their description.

3-3 AMG within INTERSECT

As explained in Chapter 2, within INTERSECT AMG is applied to the pressure equation in the first stage of the CPR preconditioner. In the second stage the complete system is solved. Therefore, the solution for cell pressure provided by AMG does not have to be very accurate and the multigrid method terminates after one V-cycle.
The coarse grids are constructed by the PMIS coarsening scheme described in Section 3-2-2. Since this parallel scheme is suitable for a simulation on one processor core, it is used for serial and parallel runs.
The number of levels comprising the V-cycle is established according to several criteria. Level $m+1$ becomes the coarsest level in the hierarchy of AMG if:

- the ratio of the number of variables on level $m+1$ to the number of variables on level m is smaller than a certain limit,
- level $m+1$ consists of one variable,
- level $m+1$ is the 50th level.

On each level except the coarsest level, one pre- and one post-smoothing step is performed. The Gauss-Seidel method discussed in Section 3-1-2, is used for the smoothing process.
On the coarsest level, the solution is found by FGMRES preconditioned by ILU(0).

3-3-1 Heuristics

Within INTERSECT, AMG is repeatedly used to find the solution of the pressure equation. Assuming that the time step is sufficiently small, the structure of the coarse grids and the strength of dependence between the coarse grid variables obtained from the pressure matrix should be preserved. For this reason, the frequency of the complete AMG setup can be reduced in order to minimize the computational time required for the multigrid method within the simulation. When the setup is partial, the coarse grids and the sparsity structure of the prolongation, restriction and coarse grid operators can be reused. Table 3-1 provides an overview of the preserved parameters per AMG level.

Level	Grid	P		R		RAP	
		Sparsity	Values	Sparsity	Values	Sparsity	Values
0	Yes	Yes	Yes	Yes	Yes	Yes	Yes
1	No	No	Yes	No	Yes	No	Yes
2	No	No	No	No	No	No	Yes
3	No	No	No	No	No	No	No

Table 3-1: Partial setup per AMG level. *Yes* means recompute, while *NO* means reuse.

Typically, INTERSECT performs the complete setup every five time steps. For the time step with the complete setup, coarse grids, P, R and RAP are recomputed for the first non-linear iteration. At subsequent Newton iterations, the setup is partial.

Chapter 4

Problem Description and Solution Strategies

4-1 Problem Description

As explained in Section 2-5, INTERSECT uses parallel computing for large problems. Here, we introduce some terminology, including the concept of strong scalability and Amdahl's Law, required to evaluate the quality of parallelisation. After that, we describe the parallel performance of AMG within INTERSECT.

4-1-1 Terminology

An algorithm's execution time τ is the wall clock time required by a K-processor system to execute an algorithm to solve a problem of size N. Independent of the definition of the problem size, τ is supposed to be directly proportional to N.
A fundamental quantity for assessing the performance of a parallel programme, which is based on the running time, is the computation rate, also known as the computation speed.

Definition 4.1. Computation rate $R(N, K)$ is the amount of computation per second that the programme performs:

$$R(N, K) = \frac{N}{\tau(N, K)}. \tag{4.1}$$

The metric that compares the computation speed before and after the parallelisation is called speedup.

Definition 4.2. Speedup is the ratio of the parallel algorithm's speed to the serial algorithm's speed:

$$\text{Speedup} = \frac{R(N, K)}{R(N, 1)}. \tag{4.2}$$

Substituting Equation 4.1 into Equation (4.3), speedup can be rewritten as

$$\text{Speedup} = \frac{\tau(N,1)}{\tau(N,K)}. \tag{4.3}$$

The scalability of a parallel algorithm quantifies the relation between the execution time and the number of processor cores on which the algorithm is executed. There are two types of scalability: strong and weak. Strong scalability measures the ratio of time to the number of cores assuming that the problem size is fixed, as the number of cores increases. By contrast, to compute weak scalability, the problem size is also increased in direct proportion to the number of cores.

If an algorithm is strongly scalable, then ideally it should require $1/\bar{K}$ of the serial execution time to compute the answer to the same problem on $\bar{K}$ processors. On the other hand, if an algorithm is weakly scalable, then its running time should ideally be preserved when a problem of size $\bar{K} \times \bar{N}$ is executed on $\bar{K}$ cores, where $\bar{N}$ denotes the size of the problem executed in serial.

Definition 4.3. The fraction of the algorithm's total running time that must be performed on a single processor is called the sequential fraction.

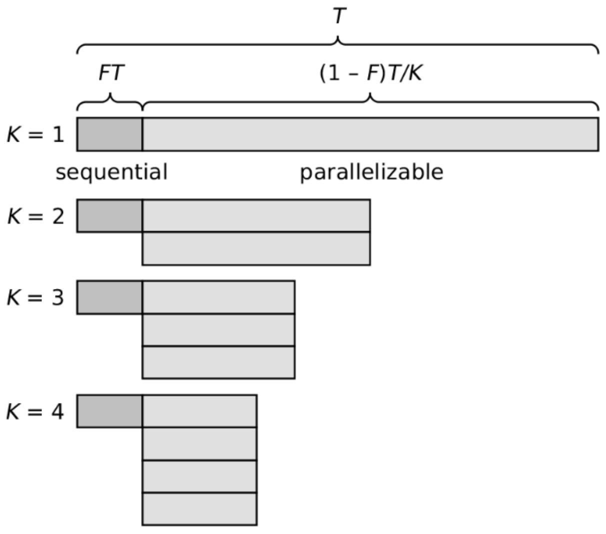


Figure 4.1: Strong scaling with a sequential fraction (source: [26]).

We denote the sequential fraction by F. Let T be the serial running time of the algorithm. Then the sequential portion of the algorithm on one core takes time FT, whilst the parallelizable portion requires time $(1-F)T$. When several cores are used, the parallelizable portion experiences an ideal speedup and its time becomes $(1-F)T$ divided by the number of cores, but the sequential portion still takes time FT (see Figure 4.1). This leads us to Amdahl's Law, stated in Theorem 4.1.

Theorem 4.1. *The total running time of an algorithm executed in parallel is equal to*

$$T(N,K) = FT(N,1) + \frac{1-F}{K}T(N,1).$$

The direct consequence of Amdahl's Law is that the overall speedup of a parallelisation is limited by the sequential portion of the algorithm, i.e. serial computation seriously hinders strong scalability.

4-1-2 Strong Scalability of AMG within INTERSECT

Although the greater part of the AMG algorithm, comprised of matrix and vector operations, can be parallelized in a straightforward way, it certainly requires communication and data exchange among processors. In addition, the parallelisation of the smoothing and coarsening processes is challenging [44].

As explained in Chapter 2, AMG forms a part of the linear solver. Therefore, the linear solver time is partially consumed by the multigrid method. Figure 4.2 illustrates the total execution time of INTERSECT and the running time of the linear solver on an increasing number of processor cores, for a problem of approximately 4×10^6 variables.

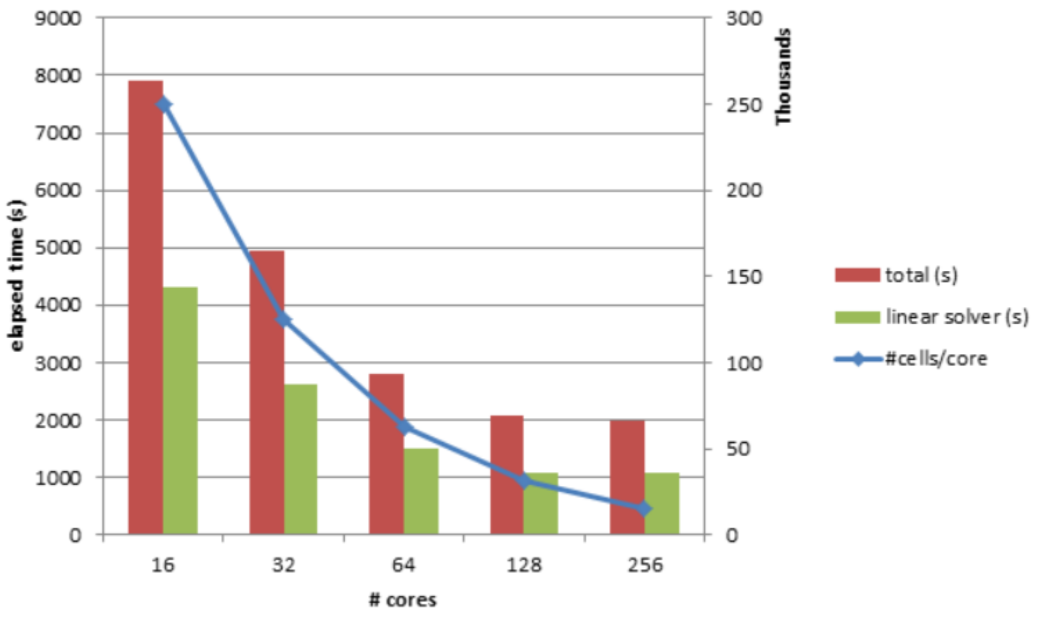


Figure 4.2: Strong scalability of INTERSECT for a case with 4×10^6 variables.

The total and linear solver time are efficiently reduced when the number of cores gradually rises from 16 to 128. However, the timings corresponding to 256 processors are approximately equal to those on 128 processor cores. This indicates the loss of strong scalability. Experiments have shown that a substantial amount of time within the linear solver is required for the communication within AMG. Obviously, for a more efficient functioning of the simulator in parallel, the data communication within the multigrid method should be reduced.

Remark 6. AMG remains weakly scalable, independent of the number of cores.

4-2 Solution Strategies

As explained in Section 3-1-11, the costs of AMG are strongly related to the algebraic complexity. The number of nonzero elements affects the number of operations per cycle. A denser operator requires more operations and, hence, a denser communication pattern. At the same time, large number of variables may entail the exchange of larger sets of data, which also requires communication between processors. Thus, the appropriate minimization of the operator and grid complexities is crucial for the reduction of communication.

Lower complexities inevitably lead to the loss of accuracy within the AMG hierarchy and accordingly slower convergence of the multigrid method. Since INTERSECT uses only one AMG V-cycle per linear iteration, the sacrifice of the convergence speed should lead to a poorer approximation of the cell pressure. Although this certainly has a negative influence on the second stage of the CPR preconitioner, ILU(0) may sufficiently improve the solution and, therefore, prevent the number of linear iterations from rising significantly. For this reason, the overall effect of the reduction of complexities should be positive with regard to the linear solver and total execution time.

In the remaining part of this section, we will discuss two solution strategies used to resolve the communication issue. Although both approaches decrease the number of nonzero entries within the hierarchy of AMG, the theory behind them is entirely different. The first solution strategy is aggressive coarsening. As explained in Section 3-2-1, aggressive coarsening is a substitute for the standard coarsening, because it modifies the coarsening scheme. By contrast, the second solution strategy, the non-Galerkin method, decreases the density of the existing coarse level operators and should be seen as an extension of the AMG algorithm.

4-2-1 Aggressive Coarsening

Aggressive coarsening was introduced in 1996 in [27]. In [37], it is suggested to apply aggressive coarsening to the finer levels in order to reduce the complexities. On the one hand, the application of aggressive coarsening drastically reduces the setup and solution costs, the complexity of the operators and the memory requirement. On the other hand, it decreases the efficiency of the smoothing procedure and encumbers the interpolation, because the prolongation operator deals with longer distances between coarse grid variables. According to [37], the benefits of the aggressive coarsening strategy certainly outweigh its disadvantages, at least when applied to Ruge-Stüben AMG. The theory behind the classical AMG with aggressive coarsening is presented in Section 3-2-1.

In [43] it is pointed out that aggressive coarsening can successfully be used with any coarsening algorithm. Similarly to the application of aggressive coarsening on classical Ruge-Stüben AMG, in order to construct sparser coarse grid operators than those generated by PMIS, the coarsening scheme used within INTERSECT, it is necessary to replace the concept of strong connection (Definition 3.9) by that of strong n-connection (Definition 3.15).
The most efficient manner to implement aggressive coarsening is by applying the PMIS algorithm twice. Firstly, PMIS is applied to the strength-of-connection matrix that should be seen as S_i from Definition 3.10. This produces a set of coarse points C, which is generally larger than desired. Secondly, PMIS is applied to matrix $S_i^{(p,l)}$ (Definition 3.17), restricted to

the variables belonging to C. For $p = 1$ and $l = 2$, an example of aggressive coarsening with PMIS is provided in Figure 4.3.

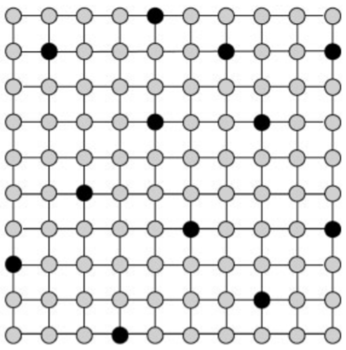


Figure 4.3: Example of aggressive coarsening with PMIS applied to a 10×10 5-point 2D Laplace operator (source: [43]). Black points denote coarse variables, while gray point indicate fine variables.

In the upcoming discussion, we use the notation introduced in Section 3-1-7.
Clearly, the interpolation formula, shown in Equation (3.19), fails whenever criterion **H 3.1** is violated by aggressive coarsening. The failure is caused by the fact that if C_i is empty, we divide by zero. To provide an alternative expression for the interpolation weights, that does not lead to a slower convergence, aggressive coarsening uses long-range interpolation, such as multipass interpolation [44].
The first pass of the miltipass interpolation is based on the following formula:

$$\omega_{ij} = -\left(\frac{\sum_{l \in N_i} a_{il}}{\sum_{k \in C_i} a_{ik}}\right) \frac{a_{ij}}{a_{ii}}. \tag{4.4}$$

More precisely, Equation (4.4) is applied to all F-points that are influenced by C-points (the length of connection path is equal to one). In the second pass, multipass interpolaion evaluates weights using the standard approach for variables which are influenced by those variables for which interpolation weights have already been obtained and which are connected by a path of length one. The second pass is repeated until weights are prescribed to to all remaining points.

To apply aggressive coarsening, INTERSECT was interconnected with SAMG (Algebraic Multigrid Methods for Systems), commercial software developed at Fraunhofer Institute for Algorithms and Scientific Computing. More precisely, the complete first stage of the CPR preconditioner was replaced by the AMG cycle within SAMG.

SAMG

SAMG is designed for the efficient solution of scalar and coupled systems of elliptic PDEs. It is a FORTAN90 library with the realization based on a modular concept. All modules are separated from each other and each has a fixed task. That is, each of the essential components of an AMG algorithm, such as smoothing, coarsening and interpolation, corresponds to a separate module. The central modules, in turn, consist of several child modules. For example,

the coarsening module contains a module for sorting and splitting procedures.
SAMG can be plugged into existing simulation codes and should be considered as a black box. Since no explicit information on the geometry is not required by AMG, only the fine grid operator and the corresponding right-hand side have to be passed to the solver. In addition, SAMG provides a rich environment allowing for many different approaches. For instance, it provides an option for design of heuristics. If a series of matrix equations with similar matrices have to be solved, the software can be advised to partially or completely reuse the setup.

The matrix data used within SAMG is the so-called modified Compressed Sparse Row (CSR) format. When the data is transformed into the standard CSR format, every matrix A is stored by means of three vectors, $\mathbf{ia} \in \mathbb{R}^{1\times n_v+1}, \mathbf{ja} \in \mathbb{R}^{1\times n_A}$ and $\mathbf{a} \in \mathbb{R}^{1\times n_A}$. Here, n_v and n_A denote the number of variables and the number of non-zero matrix entries, respectively. Vectors $\mathbf{a}$ contains the entries of A, whereas $\mathbf{ja}$ consists of the corresponding column numbers. The non-zero entries of the first row are stored first, followed by the second row, etc. Although the entries can be stored in any order within the row, typically the entry from a column with the lowest index comes first, then that from a column with the second lowest index, and so on. The third vector, $\mathbf{ia}$ is comprised of the cumulative sum of the number of entries per row. However, the $\mathbf{ia}(1)$ is always is equal to 0. Thus, the number of non-zero entries in row i is computed as $\mathbf{ia}(i+1) - \mathbf{ia}(i)$.
When the modified CSR format is used, then diagonal entries are stored first within a row. The order of the off-diagonal entries is not specified.

Example 4.1. Let matrix A be given by

$$A = \begin{bmatrix} 4 & 1 & 0 \\ 0 & 3 & 0 \\ 0 & 4 & 8 \end{bmatrix}.$$

Then the corresponding modified CSR vectors are

$$\begin{aligned} \mathbf{ia} &= [0, 2, 3, 5], \\ \mathbf{ja} &= [0, 1, 1, 2, 1], \\ \mathbf{a} &= [4, 1, 3, 8, 4]. \end{aligned}$$

As suggested in Section 3-1-9, all AMG approaches within SAMG are split into two phases, a setup phase and a solution phase. Obviously, the coarsening process is a part of the setup phase. According to [10], the typical ratio of coarse grid variables selected by SAMG to the total number of variables varies between 0.25 and 0.5 per level, i.e. the ratio of the cardinality of set C to the cardinality of set $C \cup F$ is equal to 0.25 - 0.5. Usually, this results in Galerkin operators which have more entries for the first coarse level than the finest level matrices. To reduce the number of non-zero entries of the coarse grid matrices, SAMG provides several options for aggressive coarsening. The user can choose between $A1$ and $A2$ coarsening strategies and select the levels on which aggressive coarsening should be applied.
In the second phase of AMG, SAMG performs standard cycles of V-, W- or FMG-type. The most important degrees of freedom for the solution phase are the choice of

- the type of cycling,
- the smoother,
- the solver on the coarsest level,
- the accelerator.

All smoothers and accelerators provided by SAMG are listed in [10].

SAMGp is the parallel version of SAMG. SAMGp uses subdomain blocking (see Section 3-2-2) as a parallel coarsening scheme, but allows to accelerate the coarsening by various aggressive coarsening strategies. It should be noted that all accelerated AMG approaches comprising SAMGp are scalable if applied to proper classes of applications [10]. The user's manual of SAMG/SAMGp and information on how to purchase the software can be found at [34].

In the frame of this project, both serial and parallel versions of SAMG are used. For serial simulations PMIS is replaced by the classical coarsening, while for parallel computations subdomain blocking is applied. Both classical coarsening and subdomain blocking are used with aggressive A1 coarsening on the first and second coarse levels. On the remaining levels the coarsening is standard. Moreover, on the coarsest grid SAMG applies Gauss elimination, instead of ILU(0)-preconditioned FGMRES. The other settings are equivalent to the standard INTERSECT settings, described in Section 3-3. The coupling of INTERSECT and SAMG is tested with and without heuristics.

4-2-2 Non-Galerkin method

Non-Galerkin method has been successfully applied in settings where geometric information is used to aid the multigrid algorithms in constructing the sparsity patterns and choosing matrix coefficients [2, 42]. We focus on the non-Galerkin method described in [15]. This purely algebraic approach is based on the traditional AMG techniques.

As problem size increases, the number of levels in the AMG hierarchy grows and denser coarse grid operators are generated [15]. This leads to denser communication patterns than existed on the fine level, because the processors that were not coupled on the fine level become coupled. In [18] it is shown that in parallel the time spent on some coarse levels can actually be larger than the time required for the fine level due to high density of the coarse grid operators.
The increase in density is caused by the standard Galerkin operator RAP with $R = P^T$. The non-Galerkin algorithm replaces RAP with a sparser coarse grid matrix, which aims to improve parallel scalability and maintain the convergence rate of AMG. The hierarchy of the non-Galerkin coarse grid operators is schematically shown in Figure 4.4.

$$\begin{tikzcd}[row sep=small]
\text{Level 0 (finest)} & A = A^0 \arrow[d, dotted] \\
 & A_G^1 = R^0 A^0 P^0 \arrow[d, dotted] \\
\text{Level 1} & A_{NG}^1 \arrow[d, dotted] \\
 & A_G^2 = R^1 A_{NG}^1 P^1 \arrow[d, dotted] \\
\text{Level 2} & A_{NG}^2
\end{tikzcd}$$

Figure 4.4: The hierarchy of the non-Galerkin coarse grid operators on three levels. A_G^m and A_{NG}^m denote the Galerkin and non-Galerkin coarse grid operators on level m $(m = 1, 2)$, respectively.

The algorithm consists of two phases. In the first phase the sparsity pattern of the non-Galerkin coarse grid operator is selected. While preserving the row sum, the second phase removes the entries in RAP that lie outside the non-Galerkin sparsity pattern.

Mathematical Motivation

We provide the mathematical motivation for the non-Galerkin approach under the assumption that AMG is applied to an SPD matrix.

In this section the following notation is used:

A_G is the Galerkin RAP operator,

A_{NG} is the sparser approximation of A_G obtained by the non-Galerkin algorithm,

E_G is the Galerkin two-grid error propagator,

E_{NG} is the non-Galerkin two-grid error propagator.

The general form of the two-grid error propagator is presented in Equation (3.22). For the Galerkin and non-Galerkin methods, the coarse grid matrix K^1 is equal to A_G^1 and A_{NG}^1, respectively. In the remaining part of this chapter, the level number is omitted in order to simplify the notation.

It is not difficult to see that E_G and E_{NG} can be rewritten as

$$E_G = I - B_G^{-1} A,$$
$$E_{NG} = I - B_{NG}^{-1} A,$$

where B_G and B_{NG} are the corresponding preconditioners for A.
The associated convergence factors are given by

$$\rho(E_G) = \max\left(\lambda_{\max}\left(B_G^{-1}A\right) - 1, 1 - \lambda_{\min}\left(B_G^{-1}A\right)\right),$$
$$\rho(E_{NG}) = \max\left(\lambda_{\max}\left(B_{NG}^{-1}A\right) - 1, 1 - \lambda_{\min}\left(B_{NG}^{-1}A\right)\right).$$

The result below provides an expression for the non-Galerkin two-grid convergence rate in terms of the two-grid convergence factor of the standard Galerkin method.

Theorem 4.2. *Define E_G, E_{NG}, B_G and B_{NG} as above. In addition, let*

$$\theta = ||I - A_{NG}A_G^{-1}||_2. \tag{4.5}$$

If $\theta < 1$ and both A_G and A_{NG} are SPD, then

$$\rho(E_{NG}) \leq \max\left(\lambda_{\max}\left(B_G^{-1}A\right)\frac{1}{1-\theta} - 1, 1 - \lambda_{\min}\left(B_G^{-1}A\right)\frac{1}{1+\theta}\right).$$

Proof: see (Theorem 2.1, [15]).

Theorem 4.2 shows that if A_G and A_{NG} are SPD, and

$$||I - A_{NG}A_G^{-1}||_2$$

is minimized, then the quality of convergence is preserved by the non-Galerkin method. With other words, the algorithm should construct an operator A_{NG} with a certain sparsity pattern (or non-zero pattern) so that θ is small.

To further motivate the algorithm heuristics, we define a matrix sparsity pattern as a set of tuples $\{(i, j)\}$. Denoting the elements of an $n \times n$ matrix Z by z_{ij}, the space of operators with a given non-zero pattern N is defined as

$$\boldsymbol{\mathcal{N}}_N := \{Z \in \mathbb{R}^{n\times n} : z_{ij} \neq 0 \text{ only if } (i, j) \in N\}.$$

The sparsity patterns of A_G and A_{NG} are represented by $\mathcal{N}_G$ and $\mathcal{N}_{NG}$, respectively. The algorithm decreases the density of the Galerkin matrix, if $\mathcal{N}_{NG} \subseteq \mathcal{N}_G$. Thus, our goal is to find $A_{NG} \in \boldsymbol{\mathcal{N}}_{NG}$ for which $\mathcal{N}_{NG} \subseteq \mathcal{N}_G$ and θ is small. We begin by considering $A_G \in \boldsymbol{\mathcal{N}}_G$, $A_{NG} \in \boldsymbol{\mathcal{N}}_{NG}$ and $E \in \boldsymbol{\mathcal{N}}_G$ such that

$$A_{NG} = A_G + E. \tag{4.6}$$

That is, the non-Galerkin operator is as a perturbation of the Galerkin operator.

Remark 7. Since A_{NG} and A_G are assumed to be SPD matrices, their eigenvalues are non-negative. From Equation 4.6 follows that E is symmetric. Therefore, the eigenvalues of E are real.

Substituting Equation (4.6) into Equation (4.5) yields

$$\theta = ||I - A_{NG}A_G^{-1}||_2 = ||EA_G^{-1}||_2. \tag{4.7}$$

Next, we show that under certain assumptions, the impact of the mid-range and high frequency modes on θ is minimal.
For the mid-range and high frequency modes, $\mathbf{x}$,

$$A_G\mathbf{x} = \lambda\mathbf{x}, \text{ such that } ||\mathbf{x}||_2 = 1,$$

we assume that $\lambda \in [\frac{\rho(A_G)}{10}, \rho(A_G)]$ [15].
Furthermore, for the non-Galerkin algorithm, we enforce the following row-wise heuristics:

$$||\mathbf{e}_i||_1 \leq \gamma||\mathbf{a}_i^G||_1, \tag{4.8}$$

where,
$||\mathbf{e}_i||_1 = \sum_j |e_{ij}|$ with e_{ij} representing the entries of E,
$||\mathbf{a}_i^G||_1 = \sum_j |a_{ij}^G|$ with a_{ij}^G representing the entries of A_G,
$\gamma \in [0, 1]$.

Condition stated in Equation (4.8), implies that

$$\max_i ||\mathbf{e}_i||_1 \leq \gamma \max_i ||\mathbf{a}_i^G||_1. \tag{4.9}$$

To continue our discussion, we need Theorem 4.3 (the Gershgorin theorem).

Theorem 4.3. *Let z_{ij} be elements of $Z \in \mathbb{R}^{n\times n}$. If λ_Z is an eigenvalue of Z, then λ_Z is located in one of the n closed disks in the complex plane that centered in z_{ii} and have as radius*

$$r_i = \sum_{j=1,j\neq i}^{n} |z_{ij}|,$$

i.e.,

$$\exists i \text{ such that } |z_{ii} - \lambda_Z| \leq r_i = \sum_{j=1,j\neq i}^{n} |z_{ij}|.$$

Proof: see [7].

Remark 8. From Theorem 4.3 follows that

$$\rho(Z) \leq \max_i \sum_{j=1,j\neq i}^{n} |z_{ij}| = \max_i ||z_i||_1.$$

Using Remark 8, we obtain

$$\rho(E) \leq \max_i ||\mathbf{e}_i||_1 \leq \gamma \max_i ||\mathbf{a}_i^G||_1.$$

According to [15], $\max_i ||\mathbf{a}_i^G||_1 = k\rho(A_G)$ with k typically in the vicinity of 2 or 3. Thus, the following inequality holds:

$$\rho(E) \leq k\gamma\rho(A_G).$$

From [41] follows that $||E||_2 = \sqrt{\rho\left(E^T E\right)}$. Since E is symmetric, $||E||_2 = \rho(E)$. Therefore, we have

$$||E||_2 \leq k\gamma\rho(A_G).$$

Using Definition 3.1 with the mode $\mathbf{x}$ in question, we obtain the following expression:

$$\theta = ||EA_G^{-1}||_2 = \sup_{||\mathbf{x}||_2=1} ||EA_G^{-1}\mathbf{x}||_2 = \frac{1}{\lambda} \sup_{||\mathbf{x}||_2=1} ||E\mathbf{x}||_2 = \frac{1}{\lambda}||E||_2 \leq \frac{k\gamma\rho(A_G)}{\lambda}. \tag{4.10}$$

Equation (4.10) implies that for mid-range or high frequency modes, the impact on θ is minimal when γ of the order 0.01.

It is not difficult to see that for the low frequency mode $\mathbf{x}$ such that $A\mathbf{x} \approx \mathbf{0}$ and $||\mathbf{x}||_2 = 1$, θ is small if $E\mathbf{x} \approx \mathbf{0}$. Thus, for a set of vectors H, that represents the near null space of the fine grid operator A, θ is close to zero when

$$EH \approx \mathbf{0} \Leftrightarrow A_G H \approx A_{NG} H. \tag{4.11}$$

To minimize the value of θ, the non-Galerkin algorithm enforces

$$A_G H = A_{NG} H. \tag{4.12}$$

In Chapter 3, it was stated that geometrically smooth functions are in the near null space of A. Therefore, H can be assumed to be constant. In fact, $H = \mathbf{1}$ is the most common assumption [15].[1] This choice implies that $A_G\mathbf{1} = A_{NG}\mathbf{1}$. With other words, the non-Galerkin method should preserve the row sums of A.

From the above discussion follows that the non-Galerkin algorithm attempts to minimize θ and, therefore, preserve the convergence of the standard version of AMG, by enforcing two heuristics:

- Equation (4.8) which is required for the mid-range or high frequency modes of the fine grid operator,
- Equation (4.12) which is necessary for the low frequency modes.

These heuristics form the basis of the non-Galerkin approach.

Algorithm

The non-Galerkin method consists of two complementary parts: Compute sparsity algorithm and Lumping algorithm. The non-zero pattern N_{NG} for A_{NG} is found by the Compute sparsity algorithm. It utilizes a drop tolerance in order to guaranty satisfaction of Equation (4.8). The second part of the non-Galerkin method, the Lumping process, performs the elimination of entries in A_G based on the sparsity pattern N_{NG}. It targets the heuristic stated in Equation (4.12) for the near null space of A.

[1] A more general version of the non-Galerkin algorithm, i.e. the near null space contains more than constant vector, is outside the scope of this thesis. This generalization and the corresponding algorithm are discussed in [15].

Compute sparsity

To compute the sparsity pattern, two processes are combined. One of the processes initializes the minimal sparsity pattern, while the other process targets the heuristic for mid-range and high frequency eigenmodes. The scheme can be found in Algorithm 5.
The minimal sparsity pattern is created using the prolongation operator P and the fine grid discretization matrix A. It is defines as

$$\bar{N}_{NG} = \{(i,j) \text{ such that } \left(P_I^T AP + P^T AP_I\right)_{ij} \neq 0\}, \tag{4.13}$$

where P_I is the injection operator between the coarse and fine grids. A very straightforward example of an injection matrix is presented in Example 4.2.

Example 4.2. Let the fine grid be comprised of three variables. AMG stores this components in a vector: $\begin{bmatrix}1 & 2 & 3\end{bmatrix}^T$. Suppose that on the coarse grid, the third variable is removed. Thus, on the coarse level, we have $\begin{bmatrix}1 & 2\end{bmatrix}^T$. The injection matrix P_I corresponding to this hierarchy is shown in Equation (4.14).

$$P_I \begin{bmatrix}1\\2\end{bmatrix} = \begin{bmatrix}1\\2\\0\end{bmatrix} \text{ with } P_I = \begin{bmatrix}1 & 0\\0 & 1\\0 & 0\end{bmatrix}. \tag{4.14}$$

The minimal sparsity pattern in Equation (4.13) preserves the important entries of A_G, independent of their magnitude [15]. If it was based on the on the classical strength-of-connection operator the entries with small magnitude would be removed.

$\bar{N}_{NG}$ is improved by the non-Galerkin method in order to target the heuristic. Let the set of neighbours of i in N_{NG} be given by

$$N_{NGi} = \{j \text{ such that } (i,j) \in N_{NG}\}.$$

After the pattern N_{NG} is initialized as the pattern of A_G, the algorithm removes the entries from N_{NG} operating row by row. It start with entries with the smallest magnitude and proceeds until further elimination would violate

$$2 \sum_{j \neq N_{NGi}} |a_{ij}^G| \leq \gamma \sum_j |a_{ij}^G|. \tag{4.15}$$

Equation (4.15) represents the heuristic from Equation (4.8). The factor of 2 compensates for the maximum change made to A_G when dropping an entry and lumping its value to the allowed neighbours [15].

Algorithm 5: Compute sparsity.

```
Data: $A_G, P, P_I$
1  $N_{NG} \leftarrow \emptyset$
2  for $(i,j)$ such that $a_{ij}^G \neq 0$ do
3      $N_{NG} \leftarrow N_{NG} \cup \{(i,j)\}$
4  end
5  for $i$ to $nrows(A_G)$ do
6      Initialize set $K$: $K_m$ is index of $m$th smallest magnitude off-diagonal nonzero in row $i$
7      for $m = 1$ to $|K|$ do
8          $N_{NGi} \leftarrow N_{NGi} \backslash K_m$
9          if $2\sum_{j \neq N_{NGi}} |a_{ij}^G| \leq \gamma \sum_j |a_{ij}^G|$ then
10             continue
11         else
12             $N_{NG} \leftarrow N_{NG} \cup K_m$
13             break
14         end
15     end
16 end
17 for $(i,j)$ such that $\left(P_I^T AP + P^T AP_I\right)_{ij} \neq 0$ do
18     $N_{NG} \leftarrow N_{NG} \cup \{(i,j)\}$
19 end
20 return $N_{NG}$
```

Lumping

Let the non-Galerkin operator be comprised of entries a_{ij}^{NG}. The lumping algorithm begins by initializing A_{NG} as a copy of A_G. After that, each entry a_{ij}^{NG} that is not in N_{NG} is removed from A_{NG}. A fraction of the value of a_{ij}^{NG} is added to each of j's strongly connected neighbours in row i. This is done to preserve the row sum of the Galerkin matrix as required by the heuristic in Equation (4.12). The lumping procedure uses as input the strength-of-connection matrix, S, the Galerkin coarse grid operator, A_G and the non-Galerkin sparsity pattern N_{NG}. The neighbours of j in S are defined as

$$N_{Sj} = \{k \text{ such that } s_{jk} \neq 0\},$$

where s_{jk} is an element of S.
Subsequently, we find set U, which represents the strong connections of j shared by the nonzero pattern of row i. The neighbours of the eliminated a_{ij}^{NG}, to which its value should be lumped, are stored in U. If no strong neighbours are found, i.e. $U = \emptyset$, a_{ij}^{NG} is lumped to the diagonal.
Finally, the algorithm symmetrizes A_{NG}. Since this operation affects the row sums, it is

followed by a row preserving procedure. The scheme is shown in Algorithm 6.

Algorithm 6: Lumping.

```
Data: $A_G, S, N_{NG}$
$A_{NG} \leftarrow A_G$
for $i$ to $nrows(A_{NG})$ do
    for $j$ such that $a_{ij}^{NG} \neq 0$ do
        if $j \notin N_{NGi}$ then
            $U \leftarrow N_{Sj} \cap N_{NGi}$
            if $U = \emptyset$ then
                $a_{ii}^{NG} \leftarrow a_{ii}^{NG} + a_{ij}^{NG}$
            else
                $U \leftarrow U \backslash \{i\}$
                $\sigma = \sum_{k \in U} |s_{jk}|$
                for $k \in U$ do
                    $a_{ik}^{NG} \leftarrow a_{ik}^{NG} + (|s_{jk}|/\sigma) a_{ij}^{NG}$
                end
            end
            $a_{ik}^{NG} \leftarrow 0$
        end
    end
end
$A_{NG} \leftarrow 0.5 \left( A_{NG}^T + A_{NG} \right)$
for $i = 1$ to $nrows(A_{NG})$ do
    $a_{ii}^{NG} \leftarrow a_{ii}^{NG} + \sum_j a_{ij}^{G} - \sum_j a_{ij}^{NG}$
end
return $A_{NG}$
```

Non-Galerkin within INTERSECT

It Chapter 5, is shown that the operators within INTERSECT are generally not symmetric and, hence, not SPD. There is no previous related work that considers the application of the non-Galerkin algorithm to not SPD matrices. Despite this fact, the non-Galerkin method was implemented within INTERSECT. Since the simulator makes no assumptions about the symmetry of the coarse grid operators, the symmetrization step (line 19 to line 22) within the Lumping algorithm was not included in the code. Furthermore, based on some experiments, the value of γ in Algorithm 5 was set to 0.03.

Modified Non-Galerkin

The analysis of the coarse grid operators, provided in Chapter 5, reveals that the non-Galerkin matrices typically satisfy most of the M-matrix properties. More precisely, the non-Galerkin operators generally contain a number of positive off-diagonal entries, but frequently satisfy the remaining properties stated in Definition 3.5 and Definition 3.6.

Algorithm 7: Modified lumping, INTERSECT version.

```
Data: $A_G, S, N_{NG}$
$A_{NG} \leftarrow A_G$
for $i$ to $nrows(A_{NG})$ do
    for $j$ such that $a_{ij}^{NG} \neq 0$ do
        if $j \notin N_{NGi}$ then
            $U \leftarrow N_{Sj} \cap N_{NGi}$
            if $U = \emptyset$ then
                $a_{ii}^{NG} \leftarrow a_{ii}^{NG} + a_{ij}^{NG}$
            else
                $U \leftarrow U \backslash \{i\}$
                $\sigma = \sum_{k \in U} |s_{jk}|$
                for $k \in U$ do
                    $a_{ik}^{NG} \leftarrow a_{ik}^{NG} + (|s_{jk}|/\sigma) a_{ij}^{NG}$
                end
            end
            $a_{ik}^{NG} \leftarrow 0$
        end
    end
    for $l$ such that $l \neq i$ and $a_{il}^{NG} \neq 0$ do
        if $a_{il}^{NG} > 0$ then
            $a_{ii}^{NG} \leftarrow a_{ii}^{NG} + a_{il}^{NG}$
            $a_{il}^{NG} \leftarrow 0$
        end
    end
end
return $A_{NG}$
```

The importance of the M-matrix properties for the non-Galerkin algorithm is supported by several observations. Firstly, being an M-matrix guarantees the convergence of the basic iterative methods [41]. Secondly, AMG was originally designed for the M-matrices [6]. Therefore, the use of coarse grid operators with the M-matrix properties should be beneficial for the algorithm. Finally, M-matrices belong to the more general class of monotone matrices, i.e. matrices with nonnegative inverses. In [22], it is pointed out that the use of monotone matrices considerably improves the performance of the Multiscale Finite Volume method, which is closely related to two-grid AMG.

In the light of provided arguments, we remove the positive off-diagonal entries in order to enforce as many M-matrix properties as possible. To preserve the row sums, as required by the non-Galerkin method, the values of eliminated entries are lumped to the diagonal. We note that this process can change the effect of the non-Galerkin method on the high and mid-range frequency modes, because it increases the sparsity of the operators.
The elimination of the positive off-diagonal entries is implemented within INTERSECT as a part of the Lumping algorithm. The modified lumping procedure used within INTERSECT is presented in Algorithm 7.

Chapter 5

Results

This chapter consists of three parts: the description of the test cases, the results obtained with the aggressive coarsening and the results generated by the non-Galerkin method.
In Section 5-1 we discuss the test cases in terms of the dimensions, active cells, fluid model and time discretization. In addition, an in-depth analysis of two small and two large test cases is provided.

In Section 5-2 and Section 5-3, we investigate the results by studying:

- *Algebraic complexity:* see Definition 3.14.
- *Geometric complexity:* see Definition 3.13.
- *Time steps:* the total number of time steps required to complete the simulation.
- *Non-linear iterations:* the number of non-linear iterations in the simulation.
- *Linear iterations:* the number of iterations used to solve the linear systems generated by the non-linear solver.
- *Average number of linear iterations per time step:* the total number of linear iterations divided by the total number of time steps.
- *Linear solver time:* the amount of time needed to solve the linear systems.
- *Linear iteration time:* the linear solver time divided by the total number of linear iterations.
- *CPU time:* the overall computational time required to complete the simulation.

These parameters are discussed only if they are relevant to the simulation. For instance, the average number of linear iterations per time step is computed, only if the number of time steps required for the algorithms is different.

For the standard non-Galerkin approach without heuristics, we also provide a detailed description of the results obtained for two small cases.

Section 5-2 describes the performance of INTERSECT with an aggressive coarsening strategy. It includes serial and parallel results. Aggressive coarsening was implemented by coupling INTERSECT with SAMG. Since SAMG is considered as a black box, the discussion is restricted to the complexities and performance analysis of the method.
In Section 5-3, the influence of the non-Galerkin method on the simulator is studied. The performance statistics are provided only for sequential runs, because the method was not implemented in parallel.

5-1 Test Cases

Active cells	Dimensions	Fluid model	Implicitness	Number of phases	Number of active components
2,250	$15 \times 15 \times 10$	Black Oil Isothermal	Fully Implicit	4	4
576	$24 \times 1 \times 24$	Black Oil Isothermal	Fully Implicit	4	4
63	$1 \times 7 \times 9$	Black Oil Isothermal	Fully Implicit	3	3
1,639	$20 \times 15 \times 8$	Compositional Isothermal	AIM IMPES	3	9
2,528	$9 \times 9 \times 4$	Compositional Isothermal	AIM IMPES	3	10
1,000	$10 \times 10 \times 10$	Black Oil Isothermal	Fully Implicit	3	3
389,559	$154 \times 91 \times 35$	Compositional Isothermal	AIM IMPES	3	13
348,809	$238 \times 192 \times 114$	Black Oil Isothermal	Fully Implicit	3	3
348,811	$238 \times 192 \times 114$	Compositional Isothermal	AIM IMPES	3	8
1,722,780	$18 \times 1126 \times 85$	Compositional Thermal with steam permitted	Fully Implicit	3	3
164,944	not a 'box'	Compositional Thermal with steam permitted	Fully Implicit	3	3

Table 5-1: General properties of the test cases.

Table 5-1 presents eleven test cases. In row 1 to 6 we have small cases, in the remaining rows we have large cases. The name of a case in the upcoming discussion will be based on its number of active cells. This number is equal to the number of rows in the pressure matrix A^*_{pp} from Equation (2.15), because for each active cell there is exactly one pressure equation. The dimensions correspond to the total number of cells in the x-, y- and z-direction. Table 5-1 contains several examples where the number of active cells is lower than the number of cells suggested by the dimensions. In those cases, the domain includes a number of inactive cells. Furthermore, according to the table, the 2528-case has only 324 cells. This is caused by local grid refinement, which is shown in Figure 5.1.

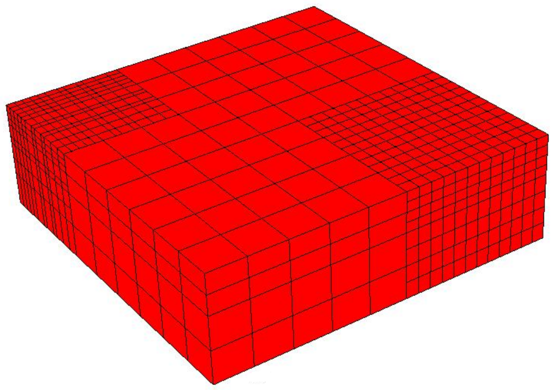


Figure 5.1: Local grid refinement of the 2528-case.

The difference between the black oil and compositional models is described in Section 2-2-2. We further distinguish between isothermal and thermal models with steam allowed. For the black oil cases, the number of active components is equal to the number of phases, because each component can exist only in one phase. If the compositional model is used, the number of active components can exceed the number of phases.
The fully implicit and AIM IMPES time discretization are defined in Section 2-2-2.

5-1-1 Mathematical Analysis

To offer a broad overview of the mathematical properties of the cases, two black oil and two compositional cases of different size categories have been selected, i.e. 2250-, 1639-, 389559- and 348809-case. The coarse level operators of the small cases will be used for an in-depth matrix analysis later in this chapter. The fine grid matrices of the four selected cases are studied in term of symmetry, sparsity and M-matrix properties.

Symmetry

The pressure matrices of the selected cases are symmetric in structure and non-symmetric in value. Thus, they do not satisfy the requirements for being SPD matrices. Nevertheless, all eigenvalues of the 2250-case matrix are positive.

Sparsity

All operators are remarkably sparse. The sparsity patterns are illustrated in Figure 5.2.

2250-case: The matrix contains only 14,460 non-zero entries. This implies that only 0.29% of all entries is not equal to zero. The sparsity pattern, shown in Figure 5.2a, reveals that the model includes only the fluxes between the neighbouring cells.

1639-case: Approximately 0.36% of the entries are non-zero entries.

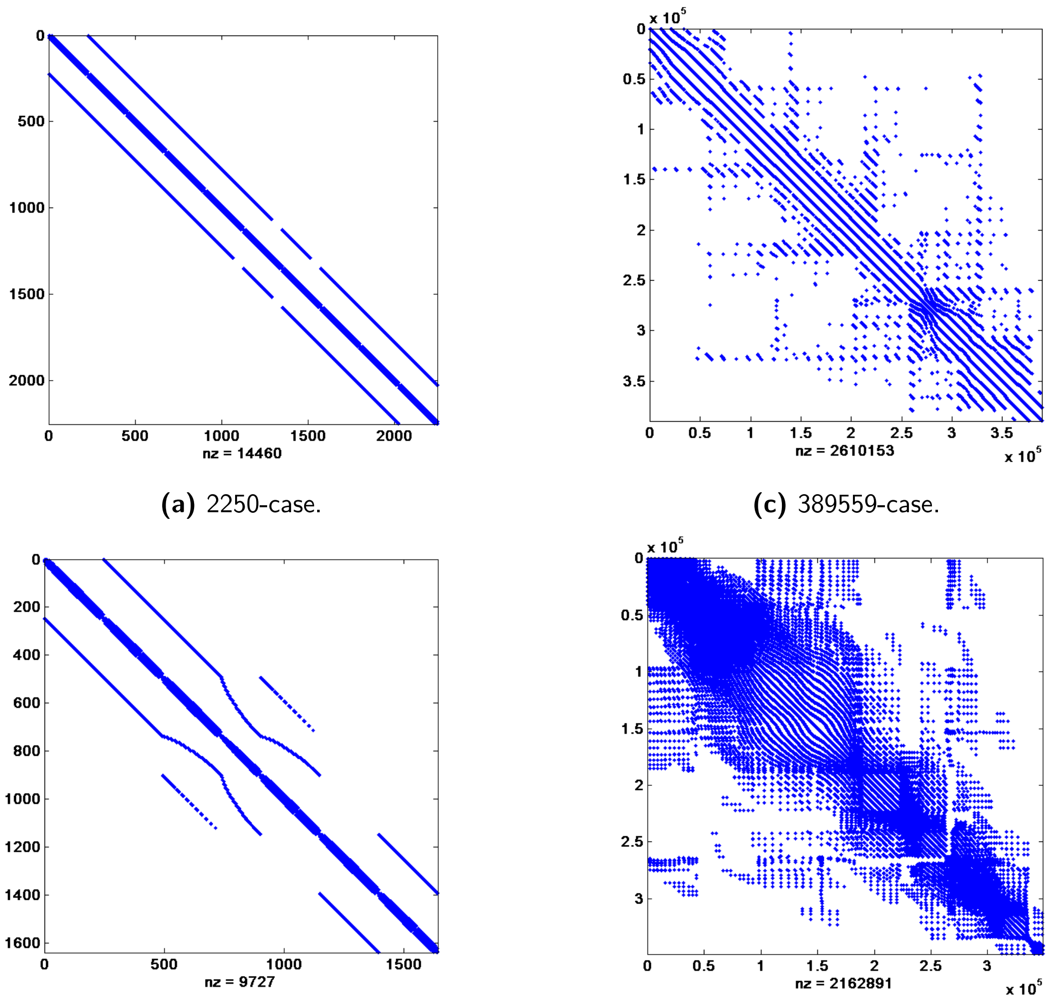
389559-case: From more than $1.5 \cdot 10^{11}$ entries of A^*_{pp} only 0.0017% are not equal to zero.

348809-case: The matrix contains 2,162,891 non-zero entries. This corresponds with 0.0018% of the total number of entries.

M-matrix Properties

According to Definition 3.5 and Definition 3.6, the pressure matrices of the 2250-, 1639- and 389559-case are M-matrices.

The off-diagonal entries of the pressure matrix of the 348809-case are negative, whereas the main diagonal is positive. Furthermore, A^*_{pp} is irreducible, but is not diagonally dominant. Therefore, it does not satisfy all requirements of Definition 3.6. Since the inverse matrix of the pressure matrix cannot be calculated due to the size of the problem, it is not possible to determine whether the matrix is an M-matrix.



(a) 2250-case. **(c)** 389559-case.

(b) 1639-case. **(d)** 348809-case.

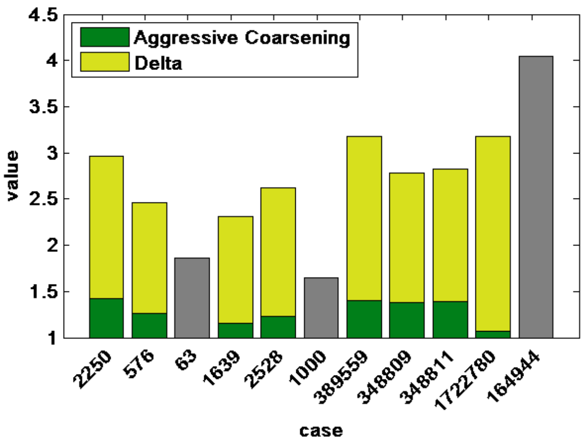
Figure 5.2: Sparsity patterns of the selected cases.

5-2 Aggressive Coarsening

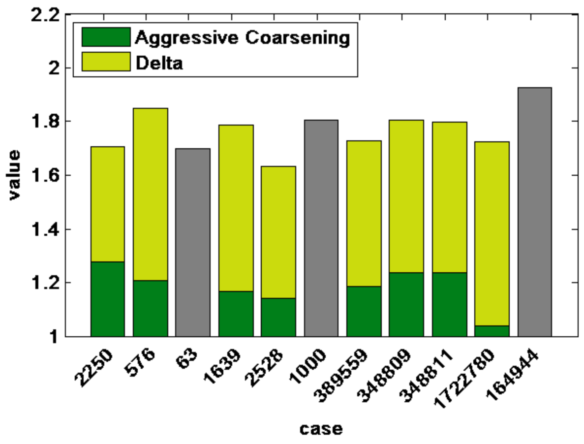
5-2-1 Serial: Aggressive Coarsening without Heuristics

In this section, we compare the results obtained with the aggressive coarsening strategy without heuristics with the results generated by INTERSECT with similar settings.

Algebraic and Geometric Complexity



(a) Algebraic.



(b) Geometric.

Figure 5.3: The decrease in algebraic and geometric complexities due to the use of aggressive coarsening. Delta shows the reduction. The total height of a bar denotes the complexity obtained with the Galerkin method, whereas the green part represents the complexity obtained with aggressive coarsening. The complexity of the cases corresponding to the gray bars is equal to one, when aggressive coarsening is applied. The height of the gray bars indicates the complexity with the Galerkin method.

The algebraic complexity is strongly reduced by aggressive coarsening (see Figure 5.3a). The gray bars correspond to the cases for which the algebraic complexity with aggressive coarsening is equal to one. According to Definition 3.14, this implies that no multigrid hierarchy is constructed by the aggressive coarsening approach. With other words, the coarse grid solver is applied to the fine grid problem.
For the remaining cases the algebraic complexity is less than 1.5. Thus, the total number of non-zero entries on the coarse levels is at least a factor two lower than the number of non-zero entries on the fine level. This can be explained by looking at the multigrid hierarchy. The aggressive coarsening method does not only increase the sparsity of the coarse grid operators, it also decreases the number of coarse levels. For example, for the 389559-case 12 coarse grid levels are constructed by INTERSECT, this number is reduced to 3 by aggressive coarsening. This is shown in Figure C.1. The described reduction leads to an extremely low number of non-zero entries, and therefore to significantly low algebraic complexities.

Figure 5.3b demonstrates the geometric complexity. Analogous to the algebraic complexity, the gray bars correspond to the cases for which the geometric complexity is equal to one. As before, this is caused by the fact that no coarse grids are constructed.

The geometric complexity of the other cases is reduced from 1.6-1.8 to 1.3 or lower values. Although the aggressive coarsening strategy efficiently reduces the number of coarse grid variables, the strong decrease in the geometric complexity is mainly caused by the low number of coarse grid levels.

Remark 9. In this chapter, the algebraic and geometric complexities are based on the data after the first linear iteration of the simulation. It has been observed that in terms of complexities the performance of the AMG method stays almost unchanged during the whole simulation, independent of the applied strategy.

Simulation Results

Small cases

For all small cases except the 63-case, the number of time steps and the number of non-linear iterations does not change when standard coarsening is replaced by aggressive coarsening. For the 63-case, the number of time steps increases from 29 to 31, whereas the number of linear iterations changes from 100 to 105.

Remark 10. For the small cases, the overall CPU time is extremely low and, therefore, cannot be used as an appropriate tool to measure the performance of the algorithms. For this reason, in this chapter the time of the small cases is not taken into account.

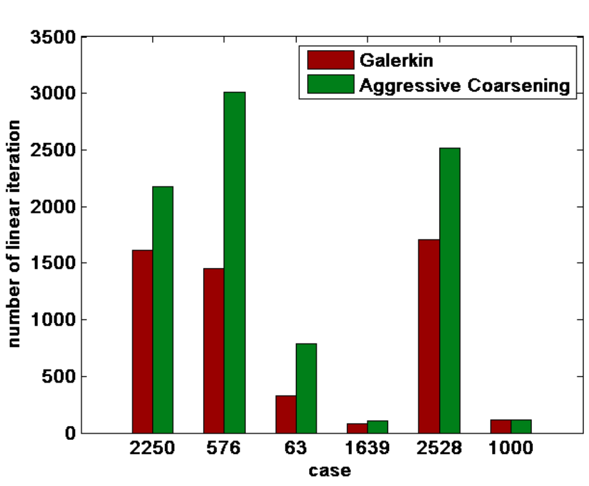
Figure 5.4a illustrates the total number of the linear iterations. It reveals that for the majority of the small cases aggressive coarsening substantially increases the number of linear iterations. Although the increase in the number of linear iterations is comparatively limited for the 1639-case, it adds more than 30% to the number of linear iterations generated with the standard Galerkin method. As shown in the figure, for the black oil 1000-case the number of iterations remains unchanged. The increase in the number of linear iterations is a predictable result, because the system becomes less accurate when aggressive coarsening is applied.

Large cases

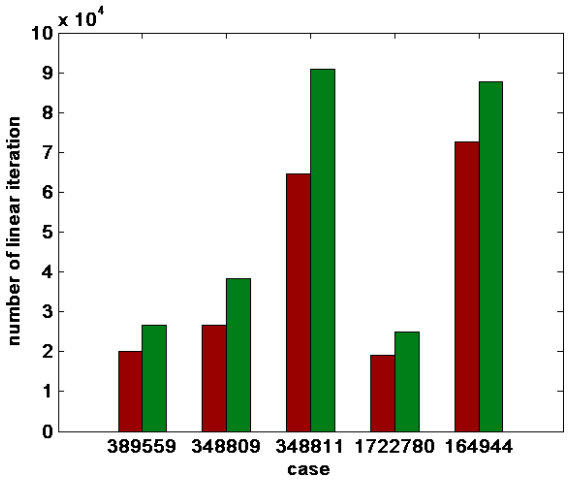
Contrary to the small cases, aggressive coarsening has an impact on the number of time steps for the large cases. The results are summarized in Table C-1. The maximum decrease of 2% is for the 348809-case, whilst the maximum increase of 21% is for the 1722780-case.

The non-linear statistics are provided in Table C-2. The table reveals that for the 1722780-case the increase is maximal (46%). The maximum decrease of 0.2% corresponds to the 164944-case.

The result for the 1722780-case match the expectations for the aggressive coarsening strategy. Since the system becomes less accurate, the non-linear solver requires more iterations to converge. In addition, the time step is frequently reduced to improve the convergence. It is more difficult to interpret a slight decrease in the number of time steps or in the number of non-linear iterations. The change in the number of time steps, which obviously leads to a different number of non-linear iterations, can be caused by one of the time step selection criteria.



(a) Small cases.



(b) Large cases.

Figure 5.4: Number of linear iterations with the Galerkin and aggressive coarsening method.

Figure 5.4b illustrates the number of linear iterations. Analogously to the small cases, the use of aggressive coarsening leads to an increase in the number of linear iterations. The increase varies between 20% and 45%.

The total time spent in the linear solver is shown in Figure 5.5. Aggressive coarsening noticeably reduces the linear solver time for the 389559-, 348811- and 164944-case. Considering the increase in the number of linear iterations, this implies that the time of an iteration is reduced. For the 164944-case, the time decrease is a remarkable phenomenon, because no coarse grids are constructed at least at the beginning of the simulation. Furthermore, the increase in the linear solver time of the 348809-case is negligible. In contrast to the other cases, for the largest 1722780-case the use of aggressive coarsening causes significant growth of the linear solver time.

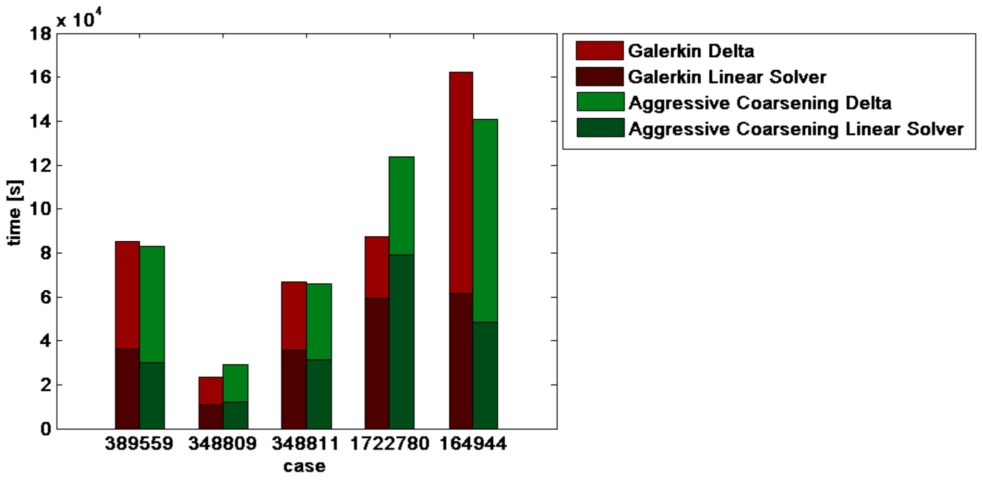


Figure 5.5: Total simulation time and linear solver time with Galerkin and aggressive coarsening methods. The total bar length is equal to the total simulation time, whereas the bottom part denotes the linear solver time.

In addition to the linear solver time, Figure 5.5 provides an overview of the total simulation time. For the 389559- and 348811-case the time gained by aggressive coarsening within the linear solver is reduced by the time required for the linearization. This can be seen as a direct consequence of the increase in the number of liner iterations.
For the 348809-case, the time rise caused by the linear solver is increased by the linearization time. Since the number of non-linear iterations for this case is lower when the Galerkin method is replaced by the aggressive coarsening strategy, this is an unexpected result.[1]
The difference in the total execution time of the 1722780-case originates from the increase in the linear solver time and the growth of the number of non-linear iterations.
For the 164944-case, the difference in the total time is greater than the difference in the linear solver time, because less non-linear iterations are required when aggressive coarsening is used.

5-2-2 Serial: Aggressive Coarsening with Heuristics

In this section, we describe the results of aggressive coarsening with the standard INTERSECT heuristics. There are two main reasons for including heuristics. Firstly, heuristics form a standard part of INTERSECT, because their use considerably decreases the computational time of the most cases. Secondly, the aggressive coarsening results can be optimized by a suitable choice of the frequency of a complete or partial setup.
Since the heuristics are included to reduce the computational time, the focus of this section is on the large cases (see Remark 10).

Obviously, the use of heuristics has no influence on the algebraic and geometric complexities after the first linear iteration. Due to the choice of the setup frequency within INTERSECT and Remark 9, we assume that the effect on the other liner iterations is negligible.

Simulation Results

Large cases

For the Galerkin and aggressive coarsening approach, there is no clear correlation between the use of heuristics and the number of steps or the number of non-linear iterations. However, with the standard INTERSECT heuristics, the aggressive coarsening strategy requires more non-linear iterations than the Galerkin method. The corresponding statistics are provided in Table C-1 and C-2.

The reduced update of the coarse grid matrices leads in general to an increase in the number of linear iterations. For the most cases the increase is smaller than 1%, whereas the maximum increase is approximately 42%. Only for the 1614944-case with the Galerkin coarse grid operators, the number of linear iterations is reduced when the heuristics are applied. The decrease is 1.4%.

[1]The simulation was executed two times. Unfortunately, the simulation results, including the number of non-linear iterations and linearization time, were different from each other. Both times the linearization time was larger than expected based on the number of non-linear iterations. Therefore, it is unlikely that the simulation was hindered by an external process, i.e. another process running simultaneously on the same node. Probably, the time increase was caused by a memory leak or another software bug. However, the simulation results of other cases were consistent.

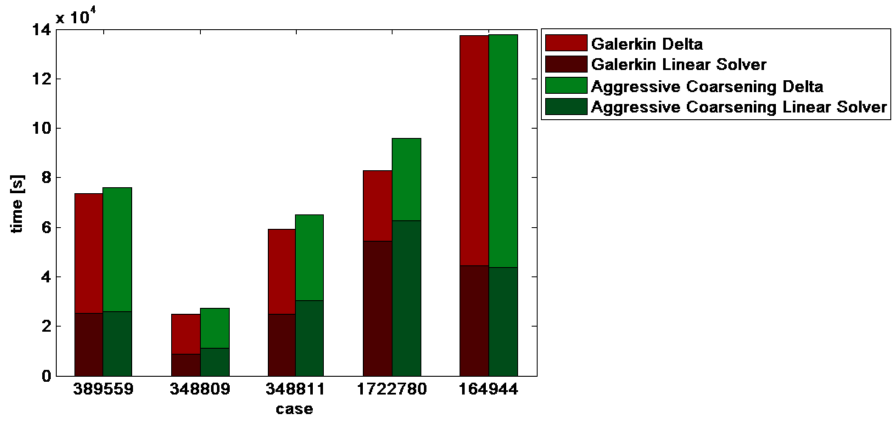


Figure 5.6: Total simulation time and linear solver time with Galerkin and aggressive coarsening methods with heuristics. The total bar length is equal to the total simulation time, whereas the bottom part denotes the linear solver time.

Despite the increase in the number of non-linear iterations in some cases, the use of heuristics reduces the total execution time. For both approaches, it substantially decreases the setup time of the linear solver and, therefore, the total simulation time. The timings are shown in Figure 5.6. Clearly, the Galerkin running time is lower than the aggressive coarsening time. The time difference originates from the linear solve and linearization stage. For the aggressive coarsening strategy, the linearization time is larger, because more non-linear iterations are required. Indubitably, the choice of heuristics for aggressive coarsening should be investigated more extensively.

5-2-3 Parallel: Aggressive Coarsening with Heuristics

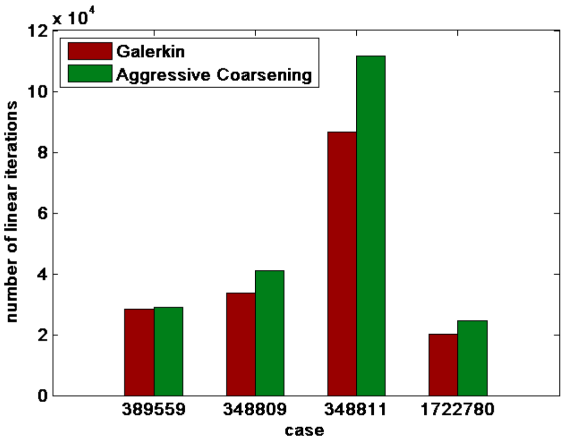
In this section, the parallel results of aggressive coarsening are compared with those of INTERSECT. The heuristics are included, because the reduction of setup frequency leads to a considerable time decrease for both strategies. The main part of the computations was performed on 32 processors. For the investigation of scalability additional simulations on 4, 8 and 16 processors were executed.
Since communication is the main issue of the parallel runs, the focus of this section is on the linear solver time, total simulation time and scalability of the large cases. Unfortunately, the application of aggressive coarsening in parallel consistently caused problems with the 1614944-case. The simulation was ‘freezing’ after several hours of computations for an unknown reason. Therefore, the case is not included in the upcoming discussion.

Simulation Results

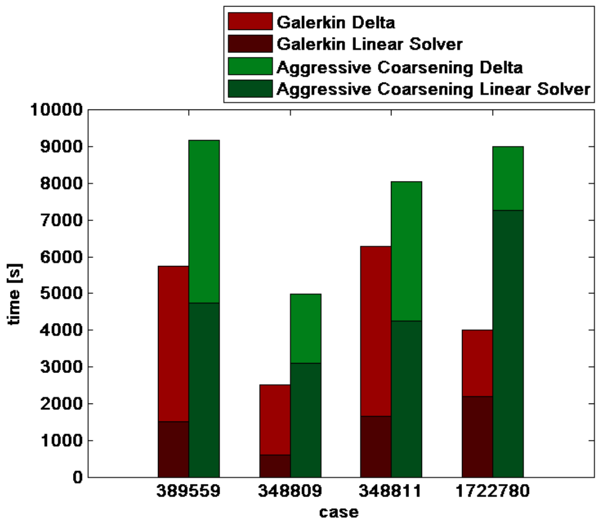
Large cases

The number of time steps and non-linear iterations of the parallel simulations can be found in Table C-1 and Table C-2, respectively. There is no clear correlation between the use of aggressive coarsening and the number of time steps or the number of non-linear iterations.

The number of linear iterations is shown in Figure 5.7a. Clearly, aggressive coarsening requires more linear iterations than standard coarsening. This implies that the convergence of the linear solver can be hindered to some degree by the aggressive coarsening strategy.



(a) Number of linear iterations.



(b) Total and linear solver time.

Figure 5.7: Number of linear iterations, and total and linear solver time for parallel simulations on 32 processors with the Galerkin and aggressive coarsening approach.

The linear solver and total time are demonstrated in Figure 5.7b. In parallel the linear solver time with aggressive coarsening is substantially higher than the Galerkin linear solver time. Therefore, the total execution time is significantly increased when standard coarsening is replaced by aggressive coarsening. The difference in the total time is additionally risen by the linearization time.

With regard to the standard Galerkin method, the performance of the aggressive coarsening strategy in parallel is obviously poorer than its performance in the serial runs. The time increase may be explained by the fact that the coarsest level systems constructed by aggressive coarsening are significantly larger than the Galerkin systems. Hence, due to low scalability of Gauss elimination [36], which is used by SAMG, the coarse grid solution can require more time.
Furthermore, additional time can originate from the application of the subdomain blocking scheme. It is not clear to what extent the behaviour of the subdomain blocking within SAMG is different from the PMIS coarsening within INTERSECT. Although it is not difficult to trace the PMIS steps, it is unfeasible to obtain more information about the coarsening within SAMG.
Another possible reason of the time growth is the fact that the heuristics are not optimized for aggressive coarsening. However, it is unlikely that the difference in the linear solver time shown in Figure 5.7b can be completely reduced by a decrease in the setup frequency.
Finally, the application of long range interpolation, which is assumed to be efficient for serial computations, leads to communication beyond neighbouring processors [12]. Clearly, this may reduce the effectiveness of the matrix storage structure and, therefore, increase the execution time.

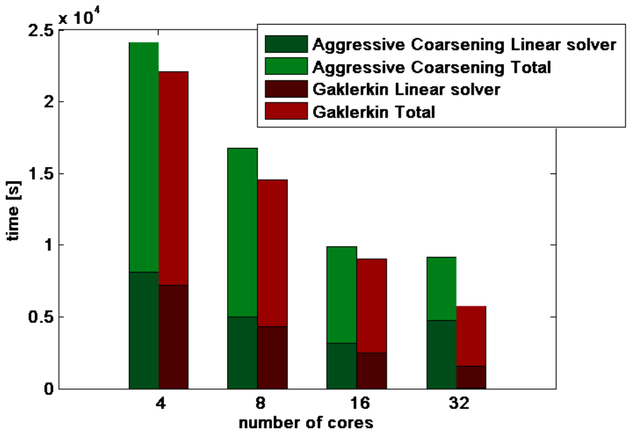


Figure 5.8: Scalability of the 389559-case with the standard Galerkin and aggressive coarsening strategy.

For the 389559-case, we also present the results for the strong scalability. The linear solver and total time of the simulations on 4, 8, 16 and 32 processors is provided in Figure 5.8. Independent of the number of cores, the liner solver and total time of standard coarsening is lower than the aggressive coarsening time.
In terms of scalability, the Galerkin results are very positive. Obviously, the number of cores is not large enough to illustrate the consequences of the communication issue.
The reduction of the linear solver and total time is clear when the number of processor cores is increased from 4 to 8 and subsequently to 16. However, when the number of processors is risen to 32, the linear solver time becomes higher than the linear solver time on 16 cores. This is probably caused by a communication issue. Despite the increase of time required for the solution of linear systems, the total running time of the simulation on 32 cores is slightly lower than that on 16 cores.

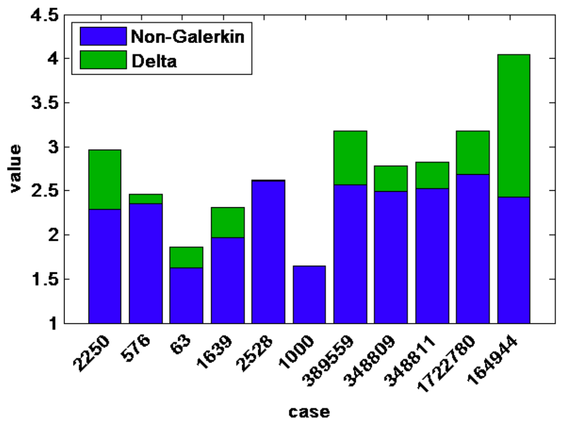
5-3 Non-Galerkin

5-3-1 Serial: Standard Non-Galerkin without Heuristics

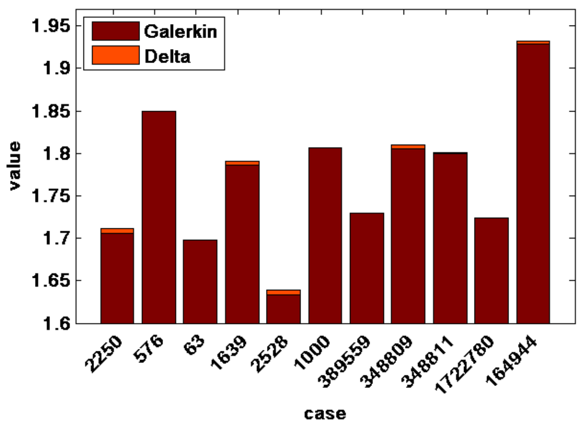
Here, we discuss the standard non-Galerkin method implemented without heuristics. We compare it to the Galerkin method within INTERSECT with equivalent settings.

Algebraic and Geometric Complexity

The effect of the non-Galerkin algorithm on the operator and grid complexity is demonstrated in Figure 5.9a and 5.9b, respectively.
In general, while maintaining the geometric complexity, the non-Galerkin algorithm significantly reduces the algebraic complexity. For example, the operator complexity of the 389559-case is reduced from 3.180 to 2.568, while the geometric complexity increases from 1.729 to 1.730.



(a) Algebraic.



(b) Geometric.

Figure 5.9: The influence of the non-Galerkin algorithm on the algebraic and geometric complexity with respect to the Galerkin method. Delta illustrates the difference in the complexities obtained with the Galerkin and non-Galerkin approach. In Figure (a) the decrease in the algebraic complexity is demonstrated. The total bar height represents the operator complexity when the Galerkin method is applied. The blue part of a bar shows the complexity obtained with the non-Galerkin approach. Figure (b) demonstrates the increase in the geometric complexity due to the non-Galerkin method. The total bar heights denotes the grid complexity with the non-Galerkin algorithm. The red part shows the complexity obtained with the Galerkin method.

For the compositional 164944-case, the decrease in the algebraic complexity is maximal: from 4.043 to 2.432. The reduction comes at the expense of a negligible increase of the grid complexity from 1.928 to 1.933.

Level	Galerkin		Non-Galerkin	
	# variables	# non-zero entries	# variables	# non-zero entries
0	389559	2645561	389559	2645561
1	192215	2771355	192215	2510780
2	61809	1761463	61868	1061441
3	20334	826104	20315	386902
4	6352	286620	6461	126501
5	2026	84174	2163	40513
6	752	25528	815	14064
7	304	7888	316	4565
8	119	2245	139	1760
9	55	743	61	558
10	22	158	27	173
11	10	50	16	60
12	4	8	11	33
13			8	20

Table 5-2: The number of variables and non-zero entries on each AMG level of the 389559-case obtained with the Galerkin and non-Galerkin method.

The algebraic complexity does not improve in the 1000-case. This can be explained by the fact that for this model the Galerkin approach requires an exceptionally low number of coarse levels and few non-zero off-diagonal entries. More precisely, the AMG method with Galerkin coarse grids constructs three coarse levels with no non-zero off-diagonal entries on the coarsest level. The application of the non-Galerkin method leads to the same AMG hierarchy. For this reason, the geometric complexity remains unchanged as well.

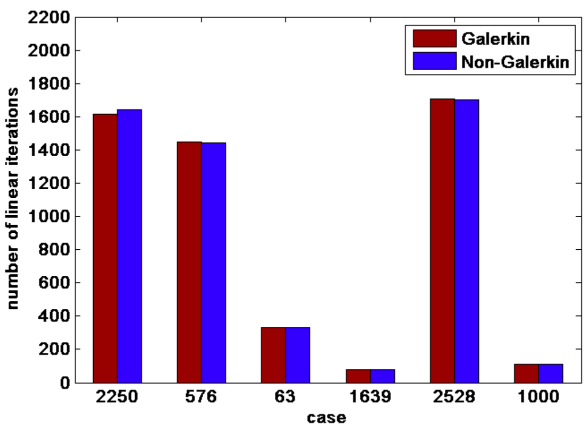
The in-depths analysis of the non-Galerkin coarse grids reveals that the increase in the geometric complexity originates from the coarsest levels, where the non-Galerkin algorithm preserves more variables than the Galerkin approach.
Moreover, the number of non-zero elements on the coarsest levels generated by non-Galerkin is usually higher than that of Galerkin. This implies that the decrease in the algebraic complexity, usually originating from the first three/four coarse levels, is slightly diminished by the coarser levels.

The number of points and non-zero entries for the 389559-case for each level of the V-cycle can be found in Table 5-2. We note that starting from level 4, the non-Galerkin algorithm repeatedly generates more variables than the Galerkin method. Furthermore, it requires one additional coarse level and produces more non-zero elements than Galerkin after level 9. This reinforces the observation that the coarsest levels slightly reduce the difference in the algebraic complexity arising from the preceding levels.

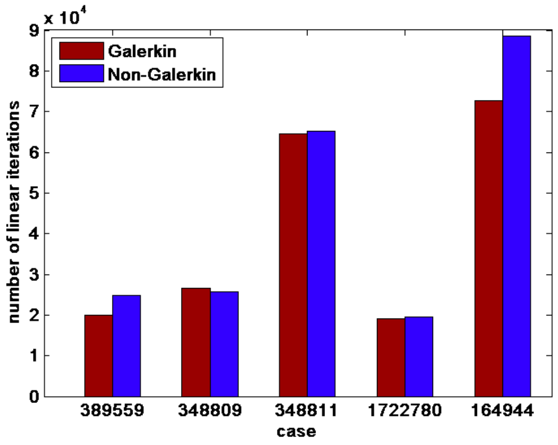
Simulation Results

Small cases

For the small cases, the number of time steps and non-linear iterations are not changed when the non-Galerkin method is applied instead of the standard Galerkin technique. The total number of linear iterations required for these cases is illustrated in Figure 5.10a. For one of the black oil cases, the 2250-case, the use of the non-Galerkin algorithm leads to a distinguishable, but not large increase in the number of linear iterations. Interestingly, for a more complicated compositional case with a greater number of active cells, the 2528-case, the number of linear iterations is slightly lower when non-Galerkin is used. In addition, the non-Galerkin method performs satisfactory for the 1000-case which contains a local grid refinement.



(a) Small cases.



(b) Large cases.

Figure 5.10: Number of linear iterations with the Galerkin and non-Galerkin coarse grids.

Large cases

For the large cases, the number of time steps differs for the Galerkin and non-Galerkin algorithms (see Table C-3). For the 389559-case, the increase due to the use of the non-Galerkin coarse grids is maximal, 12.6%. Contrary to the other large cases, the 348809-case requires a lower number of time steps with non-Galerkin.

The number of non-linear iterations of the simulations with the Galerkin and non-Galerkin algorithms can be found in Table C-4. According to the statistics, the non-Galerkin method increases the number of non-linear iterations for all the large cases except the 348809-case. The maximal increase is 28.8%.

The number of linear iterations is demonstrated in Figure 5.10b. For the 389559-case and 164944-case, this number increases considerably. We note that both cases are compositional. In addition, the 389559-case has many active components, while the 164944-case includes steam components.

The difference in the number of time steps and non-linear iterations implies that the linear solver faces a different number of problems, depending on the coarse grid operators. Therefore, assuming that the growth in the number of linear iterations is gradual, it can be useful to consider the number of linear iterations per time step. We present Figure 5.11 and Figure C.2a that show the typical behaviour of the linear solver in terms of iterations and the average number of linear iterations per time step.
Figure 5.11 demonstrates the cumulative number of linear iterations as a function of time for

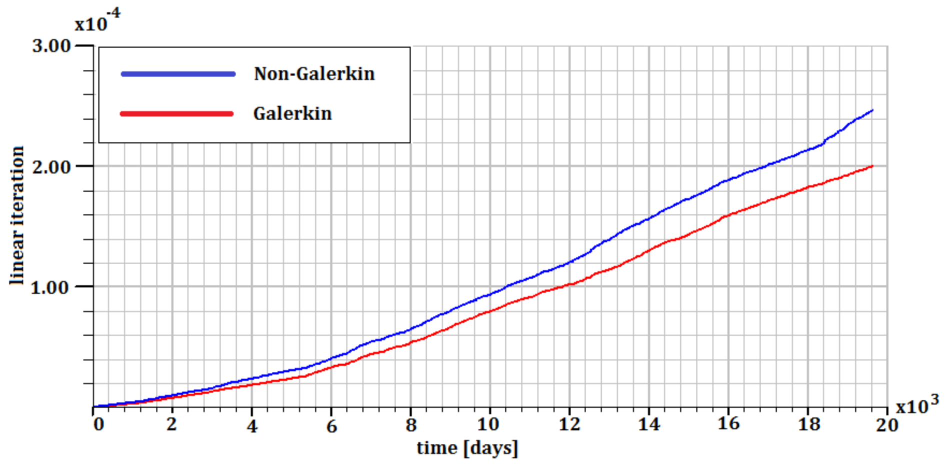


Figure 5.11: Cumulative number of linear iterations of the 389559-case with Galerkin and non-Galerkin methods.

the 389559-case. It verifies that the increase in the number of linear iterations is a global phenomenon. Thus, it is reasonable to look at the average number of linear iterations per time step, provided in Figure C.2a. The figure shows that while the application of non-Galerkin results in a distinguishable increase in the number of linear iterations per time step for the 389559-case, it hardly affects the other four cases. Thus, in terms of iterations the performance of the linear solver stays approximately unchanged when various Galerkin operators are used.

The average time of a linear iteration is illustrated in Figure C.2b. In comparison to the Galerkin method, the additional time of $1.05s$ and $1.30s$ is required per iteration of the 389559- and 1722780-cases, respectively. Although for the 348809- and 348811-cases, the additional time is lower, it corresponds with almost 50% of the linear iteration time when the Galerkin method is used. Since the non-Galerkin method reduces the density of the largest coarse grid matrices, the time increase is a direct result of the non-Galerkin setup.
From the increase in the number of linear iterations and the time per linear iteration, it follows that non-Galerkin raises the linear solver time. The linear solver data is included in Figure 5.12.

For all cases, the total simulation time is increased when the non-Galerkin method is applied. As mentioned above, the increase in time arises mainly from the setup phase, that consists of the construction of the injection matrix, the sparsity pattern calculation and the lumping procedure. In Figure 5.13 we summarize all sources for the increase in the linear solver time for the 389559-case.
In addition to the setup stage, the time increase is caused by the additional linearization steps. For the 389559-case, the linearization originates from the non-linear iterations and contributes 26% to the rise of the total execution time.
The injection matrix is a part of the input of the Compute Sparsity procedure. Contrary to the other input components, the injection matrix is not built during the standard AMG iteration with Galerkin coarse grid operators. The construction of this matrix requires less than 1% of the time difference between the Galerkin and non-Galerkin simulation for the 389559-case.

The sparsity pattern calculation corresponds with approximately 51% of the total time differ-

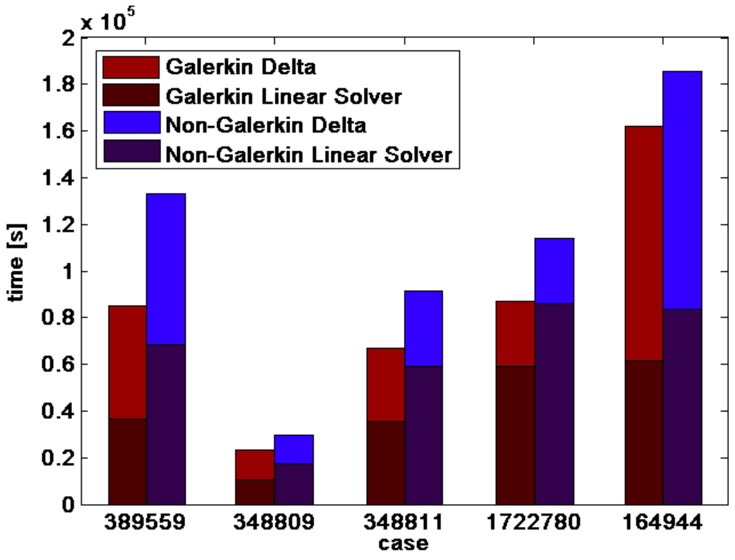


Figure 5.12: Total simulation time and linear solver time with Galerkin and non-Galerkin methods. The total bar length is equal to the total simulation time, whereas the bottom part denotes the linear solver time.

ence. As shown in Figure 5.13, we distinguish between the minimal pattern calculation and the remainder of the Compute Sparsity algorithm. The minimal pattern calculation requires two multiplications of three large sparse matrices. The matrix multiplication is an expensive procedure within INTERSECT, therefore, approximately 36% of the increase in the total time consists of the minimal pattern calculation.
The remaining 23% of the time difference is spent in the lumping algorithm, which combines the Galerkin coarse grid matrix with the non-Galerkin sparsity pattern, while maintaining the row sums.
It is interesting to note that for the 348809-case, the computation of the minimal pattern requires 54% of the time growth. In addition, a significant amount of time is spent on the lumping procedure and the remainder of the sparsity pattern calculation (see Figure C.3). As explained above, the non-Galerkin leads to a reduction in the number of non-linear iterations for this case, therefore, no extra linearization steps are performed.

Case Analysis

In this secion, we describe the results obtained for the 2250- and 1639-case in detail. The analysis is performed in terms of

- solution for the cell pressure,
- sparsity of the coarse grid operators,
- symmetry of the coarse grid operators,
- M-matrix properties of the coarse grid operators,

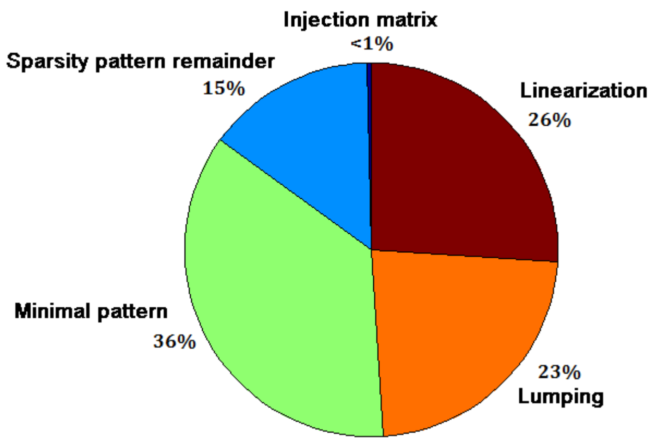


Figure 5.13: Sources of the total time growth due to the use of the non-Galerkin coarse grid operators, the 389559-case.

- spectrum of the preconditioned pressure matrix.

2250-case

Solution after 1 V-cycle

We compare the solution for the cell pressure with the non-Galerkin coarse grid operators to the solution provided by INTERSECT after one V-cycle. Figure 5.14 illustrates the solution after the first V-cycle within the first non-linear iteration. It follows from the figure, that the difference between the solutions is extremely subtle. The calculations show that the maximum norm of the error is equal to 2.9×10^{-6}.

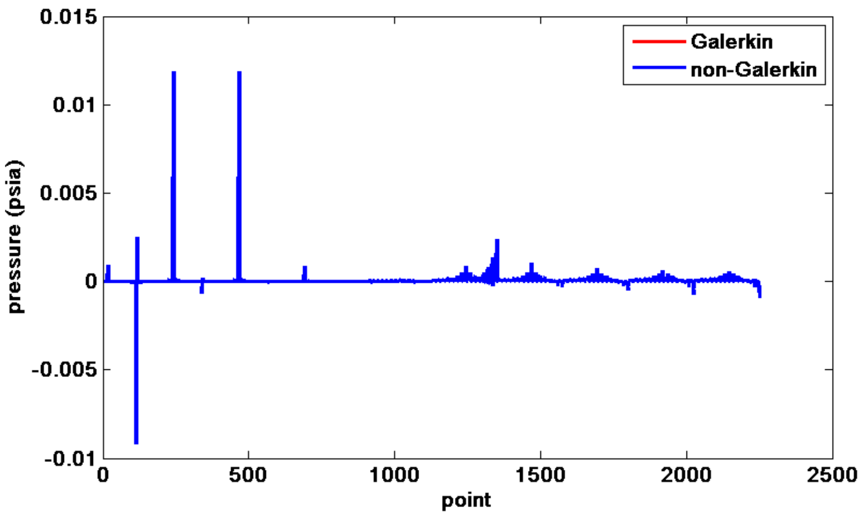
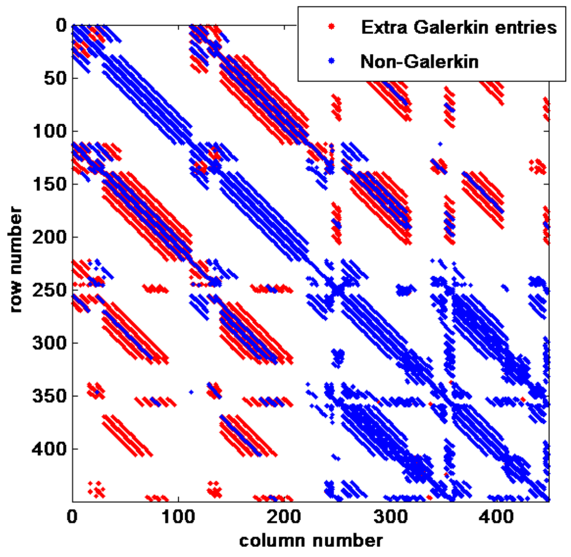
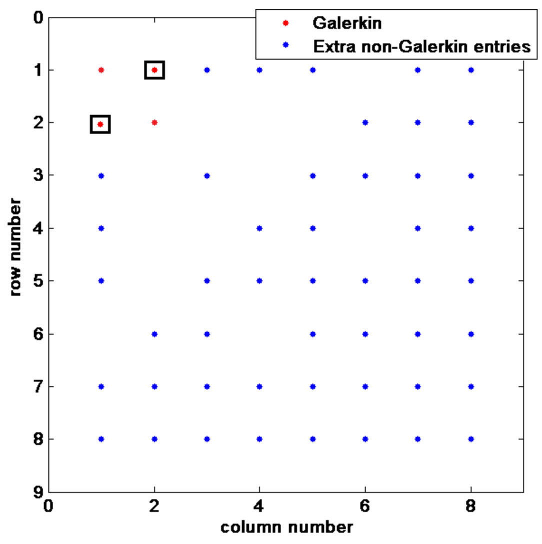


Figure 5.14: Solution for the cell pressure after one V-cycle.

Sparsity Six coarse grid levels are constructed by AMG with the Galerkin operators for the



(a) Second coarse level.



(b) Fifth coarse level.

Figure 5.15: Sparsity patterns. In Figure (a), the blue dots form sparsity pattern of the non-Galerkin operator, whereas the combination of the blue and red dots represents the sparsity of the Galerkin operator. In Figure (b), the red dots correspond to the Galerkin sparsity pattern, while the combination of the blue and red dots without squares shows the sparsity pattern of the non-Galerkin operator.

2250-case. Although the non-Galerkin method preserves this number, it increases the number of variables starting from the fourth coarse grid level. This is demonstrated in Figure C.4a. Certainly, this is a minor drawback, because the size of the problems on the coarsest levels is very low, i.e. there are less than thirty variables for both methods.
A direct consequence of the problem growth is the inability of non-Galerkin to increase the sparsity of the matrices on the coarsest levels. Therefore, just as in the 389559-case, the non-Galerkin performance in terms of matrix density is inconsistent. Nevertheless, the method efficiently increases the sparsity on the first three coarse levels, which are more relevant for the reduction of communication. For instance, on the third coarse level the density decrease is more than 53%. The number of non-zero elements per level is shown in Figure C.4b. Figure 5.15a and Figure 5.15b illustrate the sparsity patterns on the second and fifth coarse level, respectively. Clearly, the non-Galerkin method successfully increases the sparsity of the operator on the second coarse level, but fails to do so on the fifth coarse level.

Symmetry

The Galerkin coarse grid operators are symmetric in structure and non-symmetric in value for the whole hierarchy of the first V-cycle. Since the symmetrization step is not included in the INTERSECT implementation of the Lumping algorithm, the non-Galerkin matrices are not symmetric. More precisely, the non-Galerkin operators are non-symmetric in value, but are symmetric in structure on all levels, except the second and third coarse level.

M-matrix properties

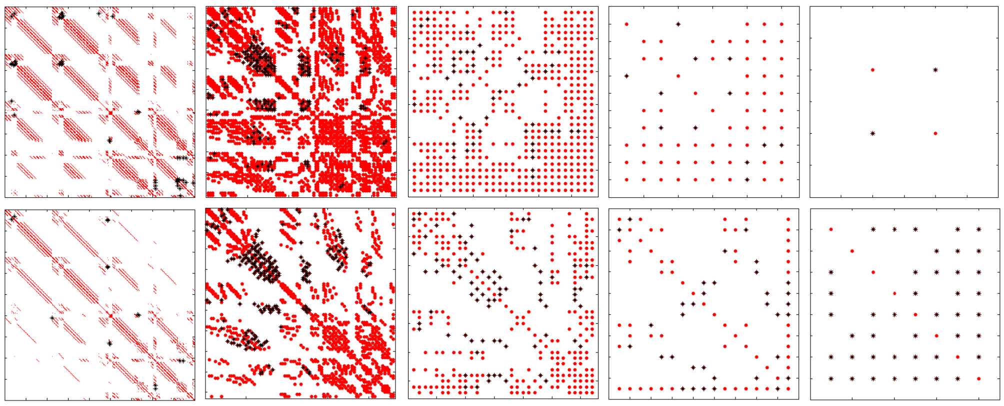
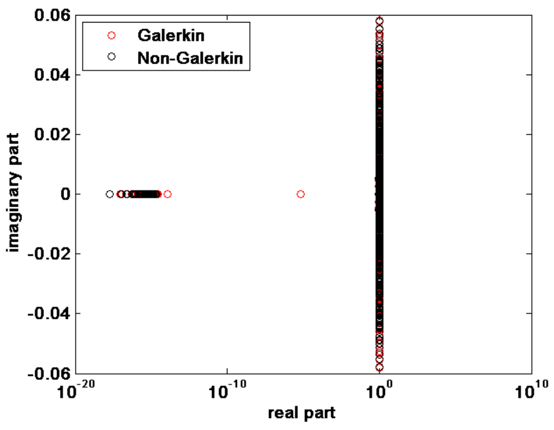
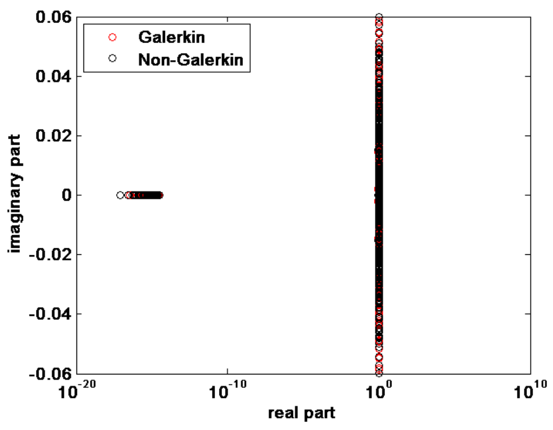


Figure 5.16: The sparsity patterns of the Galerkin and non-Galerkin operators. The negative and positive off-diagonal entries are shown in red and black, respectively. The first row is formed by the Galerkin operators starting from the second coarse level. The second row shows the non-Galerkin operators on the same levels.



(a) After the first linear iteration.



(b) After the tenth linear iteration.

Figure 5.17: The approximated spectrum of the pressure matrix preconditioned by one AMG V-cycle with the Galerkin and non-Galerkin coarse grid operators.

The non-Galerkin coarse grid operators preserve the M-matrix properties of the Galerkin operators. On the first coarse grid, the operators are irreducibly diagonally dominant, have positive diagonal entries and negative off-diagonal entries. Thus, they are M-matrices by Definition 3.5.

Starting from the second coarse level the operators contain positive off-diagonal entries and, therefore, do not satisfy the requirements for being an M-matrix. Although the non-Galerkin approach decreases the number of positive off-diagonal entires from 50 to 29 on the second coarse level compared to Galerkin, it repeatedly creates more positive off-diagonal entries on the remaining levels. The positions of the positive off-diagonal entries of Galerkin and non-Galerkin coarse grid operators are presented in Figure 5.16.

Spectrum

From Theorem B.5 and Section 2-3-4 it follows that the spectrum of the preconditioned pressure matrix is important for the convergence of FGMRES. Therefore, in this section we compare the eigenvalues of the pressure matrix preconditioned by one AMG V-cycle with Galerkin coarse grid operators to those preconditioned by one AMG V-cycle with non-Galerkin hierarchy. In Appendix B-4 it is explained that Ritz values are approximations of the desired eigenvalues. For this reason, we describe the spectrum of a preconditioned pressure matrix by calculating 1,000 Ritz values.

The eigenvalues of the original pressure matrix lie between 8.2 and 3.3×10^3. The approximated eigenvalues after the first and tenth linear iterations are illustrated in Figure 5.17a and Figure 5.17b, respectively.

Remark 11. Since we expect the original pressure matrix and the pressure matrix used in the tenth linear iteration to be very similar, we assume that the eigenvalues of the latter matrix are close to those of the original pressure matrix.

The figures demonstrate that the Galerkin and non-Galerkin methods affect the spectra of the pressure matrices in a similar manner. In Figure 5.17a, the majority of the real parts of the Galerkin eigenvalues (more than 87%) is clustered around one, while the remaining eigenvalues are close to zero. In Figure 5.17b, for only 110 Galerkin eigenvalues the real parts is clearly separated from one. Although the distribution of the non-Galerkin eigenvalues is slightly different, the above facts also hold for the non-Galerkin method.

It should be noted that the preconditioned pressure matrix is only a part of the system preconditioned by the CPR preconditioner, obtained after the application of the Schur complement. Therefore, the presence of the eigenvalues with the real part close to zero does not directly imply that the CPR preconditioner performs poorly for the 2250-case.

1639-case

Solution after 1 V-cycle

As in the 2250-case, we consider the cell pressure after the first linear iteration of the simulation. The solution for the cell pressure obtained with the non-Galerkin approach is remarkably similar to the solution provided by the Galerkin method (see Figure 5.18). The maximum norm of the error equals 3.0×10^{-13}.

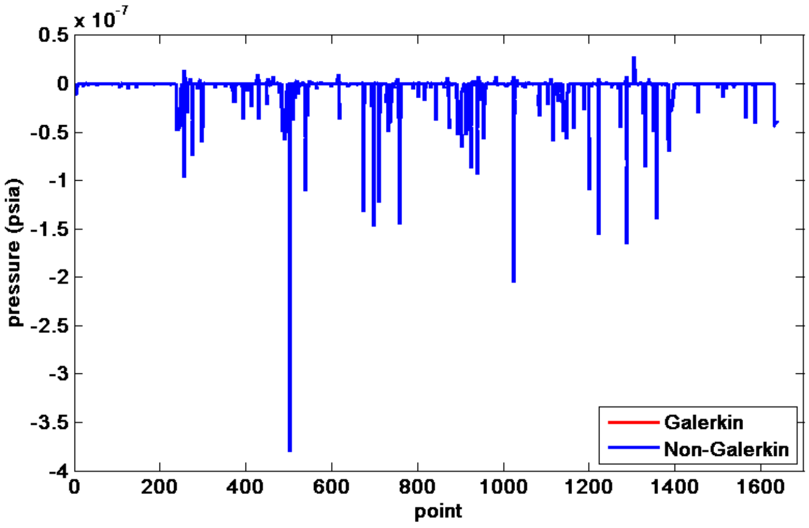


Figure 5.18: Solution for the cell pressure after one V-cycle.

Sparsity

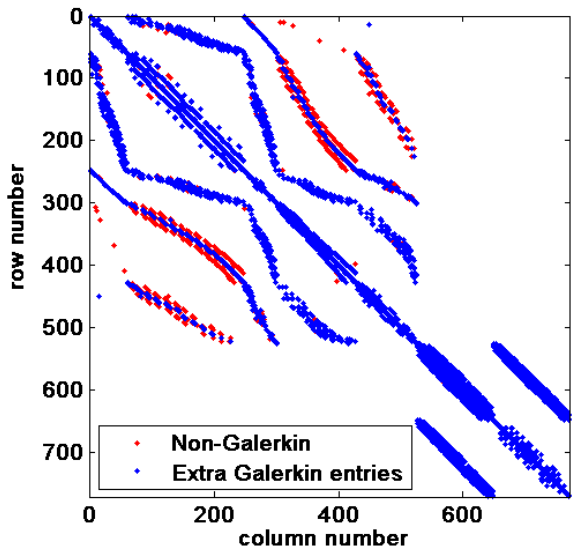
Regarding the sparsity and the number of levels, the performance of the non-Galerkin algorithm for this case is exceptionally high. The method constructs seven coarse levels, whereas the Galerkin hierarchy constists of eight coarse levels. Although, starting from level 3, the non-Galerkin algorithm preserves more variables, its number of non-zero elements is lower than that of the Galerkin algorithm. On the first and second coarse level, non-Galerkin reduces the number of non-zero entries by slightly less than 35% and 45%, respectively (see Figure 5.19a and Figure 5.19b). The number of variables and non-zero entries per level is provided in Figure C.5a and Figure C.5b, respectively.

Symmetry

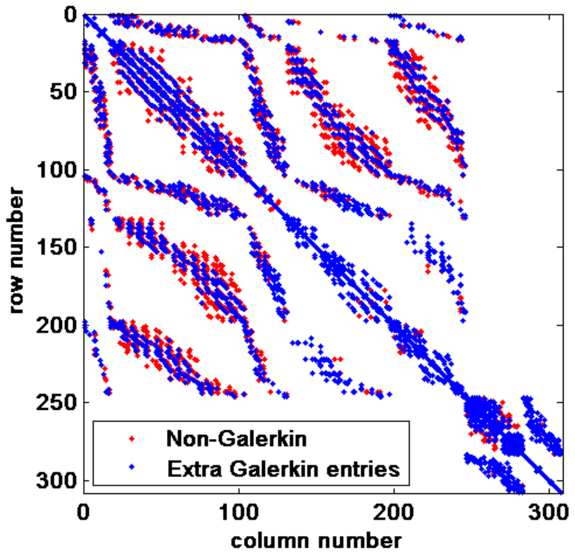
All Galerkin and non-Galerkin coarse grid operators are symmetric in structure and non-symmetric in value.

M-matrix properties

At the first coarse grid level, the Galerkin approach yields a matrix with a positive main diagonal. The inverse of this matrix is entrywise positive. Furthermore, the majority of the off-diagonal entries is negative: there are only four positive values. The non-Galerkin operator on the first coarse level contains negative off-diagonal entries, positive diagonal entries, and is irreducibly diagonally dominant. This implies that the non-Galerkin method transforms the Galerkin matrix into an M-matrix.



(a) First coarse level.



(b) Second coarse level.

Figure 5.19: Sparsity patterns. The blue dots form sparsity pattern of the non-Galerkin operator, whereas the blue and red dots form the sparsity of the Galerkin operator.

On the coarser grids the operators do not satisfy the requirements for being M-matrices, independent of the approach. First of all, they contain positive diagonal and off-diagonal entries. In addition, some entries of the inverse matrices are negative. Interestingly, on the second coarse level, the number of positive off-diagonal entries generated by non-Galerkin is lower than that produced by the Galerkin method. However, on the remaining levels the number of positive off-diagonal entries comprising the Galerkin matrices is higher.

Spectrum

For this case we present the results after the first linear iteration, as the distribution of the Ritz values after the tenth linear iteration is extremely similar to the distribution shown in Figure 5.20. The figure includes 1,000 approximated eigenvalues.

The eigenvalues of the original pressure matrix range between 8.5×10^5 and 2.6×10^7. When the Galerkin operators are applied, the number of the Ritz values, which are close to zero, is equal to 274. The non-Galerkin algorithm slightly shifts some of the approximated eigenvalues, but does not change the main properties of the distribution.

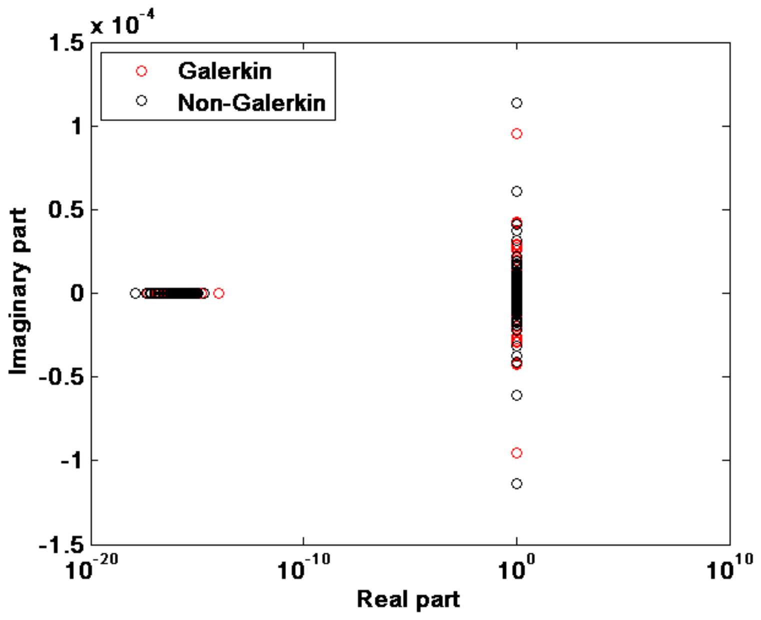


Figure 5.20: The approximated spectrum of the pressure matrix preconditioned by one AMG V-cycle with the Galerkin and non-Galerkin coarse grid operators after the first linear iteration.

5-3-2 Serial: Standard Non-Galerkin with Heuristics

In this section, the performance of the non-Galerkin method with the INTERSECT heuristics is described. The results are compared to those of the standard non-Galerkin algorithm without heuristics and Galerkin algorithm with heuristics. As in the section about aggressive coarsening, the linear solver and total time of the large cases form the main subject. In addition, the non-Galerkin performance with slightly different heuristics is discussed.

Simulation Results

Large cases

The number of time steps rises, when the heuristics are included in the non-Galerkin algorithm. This is shown in Table C-3. The increase is possibly caused by a slower convergence of the Newton-Raphson method.

The number of non-linear iterations is presented in Table C-4. According to the table, the Galerkin algorithm with heuristics requires a lower number of non-linear iterations than the non-Galerkin method with standard INTERSECT heuristics. At the same time, there is no clear correlation between the non-Galerkin versions in terms of non-linear iterations.

Furthermore, the number of linear iterations increases by $2\% - 28\%$, when the non-Galerkin algorithm with heuristics is applied instead of Galerkin with the same settings. However, the non-Galerkin algorithm with heuristics requires in general more linear iterations than the non-Galerkin method without heuristics.

The time required for the linear solver decreases for all cases when the frequency of the AMG setup is reduced. For the most cases, the extra linearization time due to the increase in the number of non-linear iterations is outweighed by the reduction of the linear solver time. However, for the 348809-case the increase in the number of non-linear iterations of the non-Galerkin algorithm prevents the decrease of the total simulation time. In fact, the heuristics have a negative effect on the non-Galerkin total time for this case.

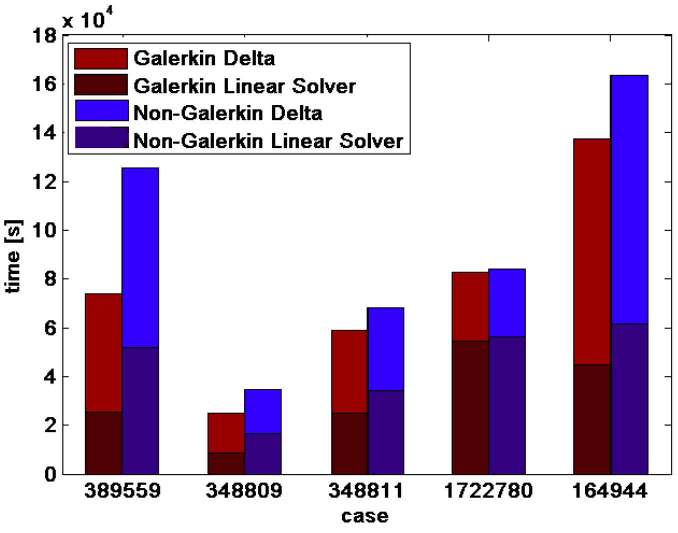


Figure 5.21: Total simulation time and linear solver time with Galerkin and non-Galerkin methods with heuristics. The total bar length is equal to the total simulation time, whereas the bottom part denotes the linear solver time.

The timings of the non-Galerkin and Gaerkin methods with heuristics are demonstrated in Figure 5.21. Obviously, the heuristics are more favourable for the Galerkin approach. Therefore, we made an attempt to improve the non-Galerkin time by performing the complete setup less frequently. Unfortunately, the effect was opposite: there was a clear-cut increase in the non-Galerkin linear solver and total simulation time compared to the non-Galerkin method with the standard INTERSECT heuristics.

5-3-3 Serial: Modified Non-Galerkin without Heuristics

Inspired by the matrix properties of the 2250-case coarse grid matrices in Section 5-3-1, a modified version of the non-Galerkin method was implemented. We compare its performance to INTERSECT and to the standard non-Galerkin method without heuristics.

Algebraic and Geometric Complexity

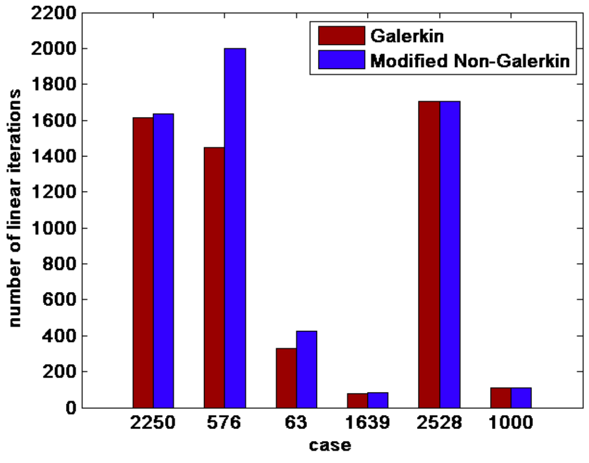
The difference in the complexities of the modified and standard non-Galerkin algorithm are very limited. For the small 576-, 63- and 1000-case, the complexities are unchanged, because the coarse grid operators of these cases contain no positive off-diagonal entries.
Generally, the replacement of the undesired off-diagonal values decreases the algebraic complexity by approximately 1.5%. This implies that the complexity is substantially improved in comparison with the Galerkin method. For the 1722780-case, the decrease is exceptionally large, almost 25%.

The effect of the modifications on the geometric complexity of standard non-Galerkin seems arbitrary. The geometric complexity can be slightly higher, lower or remain completely unchanged. The differences in the geometric complexity are not higher than 0.5%.

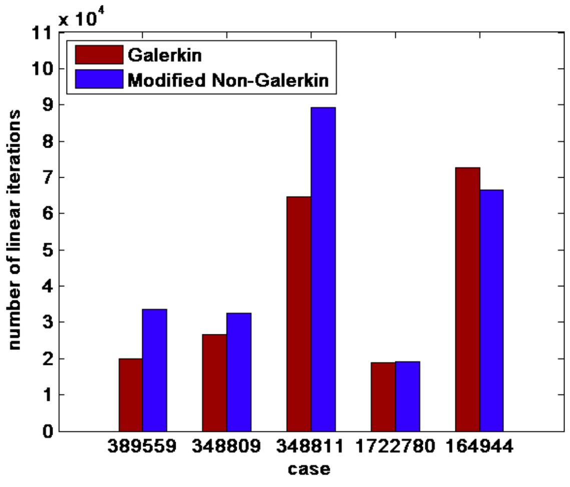
Simulation Results

Small cases

To describe the simulation statistics of the modified non-Galerkin method, we compare them to the results produced by the Galerkin algorithm. According to statistics, the modified method has no influence on the number of time steps and it changes the number of non-linear iterations only for the 63-case. The observed decrease is one iteration.



(a) Small cases.



(b) Large cases.

Figure 5.22: Number of linear iterations with the Galerkin and modified non-Galerkin coarse grids.

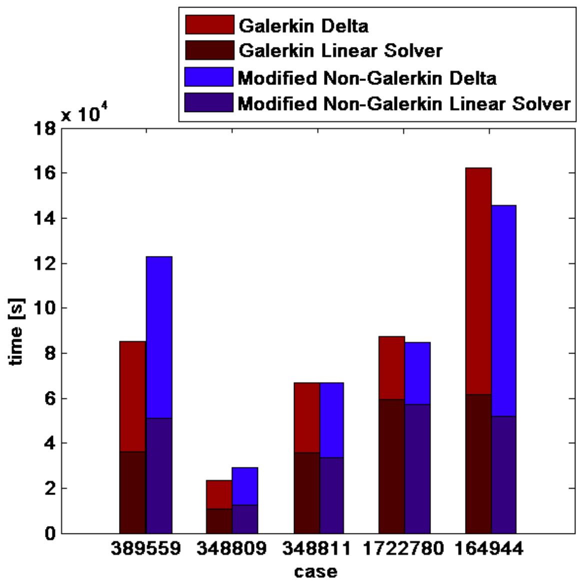
The number of linear iterations is illustrated in Figure 5.22a. There are no changes for the 1000-case and the increase for the 1639- and 2528-case is negligible. For the remaining three cases the difference is noticeable: the increase for the 576-case is 40%. From the previous results follows that for the 2250-, 576- and 63-case the modified version also significantly increases the number of linear iterations compared to the standard non-Galerkin approach.

Large cases

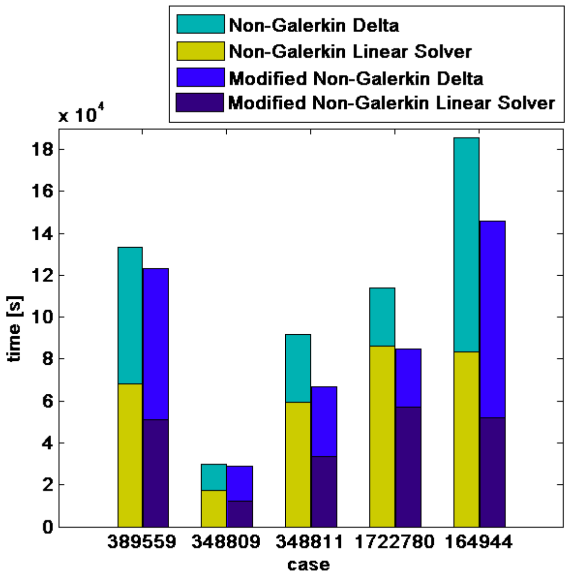
The number of time steps needed for the modified non-Galerkin method is presented in Table C-3. The maximal increase of approximately 35 % with respect to Galerkin approach is observed for the 389559-case. At the same time, the decrease for the 348809- and 164944-case is very limited (less than 1%).

In general, the new method requires more non-linear iterations than the Galerkin method. For the 359559-case the increase is approximately 43%. The 164944-case is an exception, because its number of non-linear iterations is reduced by the modified version. The statistics can be found in Table C-4.

The number of linear iterations increases for all cases except the 164944-case. However, the duration of a linear iteration is reduced when the Galerkin operators are replaced by the non-Galerkin operators with negative off-diagonal entries. Therefore, when the difference in the number of linear iterations is not too large, the linear solver time of the modified non-Galerkin is lower than the Galerkin linear solver time. This is shown in Figure 5.23a.



(a) Galerkin.



(b) Standard non-Galerkin.

Figure 5.23: Total simulation time and linear solver time of the modified non-Galerkin method compared to the Galerkin and standard non-Galerkin method. The total bar length is equal to the total simulation time, whereas the bottom part denotes the linear solver time.

The total simulation time is significantly reduced for the thermal 1722780- and 164944-case. In addition, a small decrease is observed for the 348811-case. On the other hand, the CPU time of the 389559- and 348809-case increases when the modified non-Galerkin algorithm is used instead of the Galerkin method. The increase originates from the linear solver and linearization time. Since the number of non-linear iterations increases considerably for the 389559-case, the contribution of the linearizaion time exceeds the difference caused by the linear solver.
Analogously, for the 348811-case, the time gained within the linear solver is reduced due to the increase in the number of non-linear iterations. Contrary to the other cases, for the 164944-case the linearization is profitable when the modified non-Galerkin algorithm is used. This is benefit is caused by the low number of non-linear iterations.

It is important to note that the modified non-Galerkin algorithm improves the results of the standard non-Galerkin method in terms of linear solver and total time. This is illustrated in Figure 5.23b.

Case Analysis

2250-case

Solution after 1 V-cycle

The modified non-Galerkin method approximates the Galerkin solution very accurately. The maximum norm of the error is equal to 3×10^{-6}.

Sparsity

For the 2250-case, seven coarse grid levels are constructed when the modified non-Galerkin method is applied. Recall that for the Galerkin and standard non-Galerkin method, only six coarse levels are generated. On the common levels, the number of variables of the standard and modified non-Galerkin approach coincide (see Figure C.4a). This implies that due to the coarsest level the geometric complexity of the modified system is slightly higher. However, regarding the algebraic complexity, the method is more favourable than the standard non-Galerkin. The number of non-zero entries per level is illustrated in Figure C.4b.

Symmetry

Although, the modified version of non-Galerkin produces non-symmetric operators on the first six coarse levels, the matrix corresponding to the coarsest grid is symmetric. By analogy with the standard non-Galerkin method, the operators obtained with the modified algorithm are non-symmetric in structure and value on the second and third coarse levels. The remaining operators are symmetric in structure.

M-matrix properties

All coarse grid operators generated by the modified non-Galerkin method satisfy the M-matrix properties. The robustness of this result is not straightforward, since the modified matrices can substantially differ from the non-Galerkin matrices after the multiplication with the restriction and interpolation operators.

Spectrum

The modified non-Galerkin algorithm hardly affects the spectrum of the preconditioned pressure matrix. With regard to the Galerkin method, it slightly shifts the Ritz values (see Figure C.6a).

1639-case

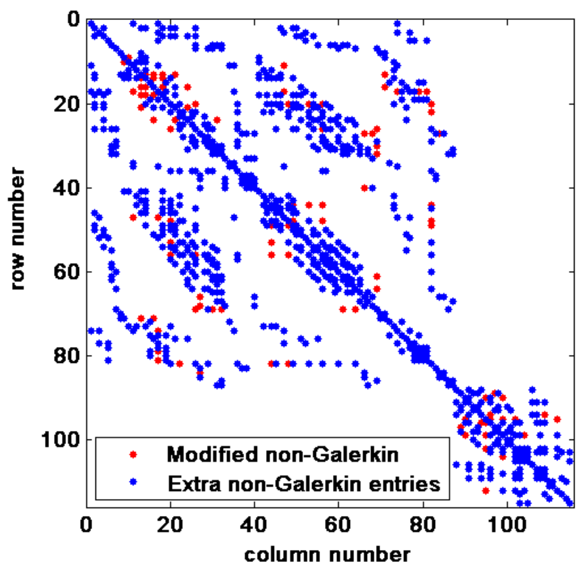
Solution after 1 V-cycle

The cell pressure solution is almost unchanged by the modified non-Galerkin method. The maximum norm of the error equals 3×10^{-13}.

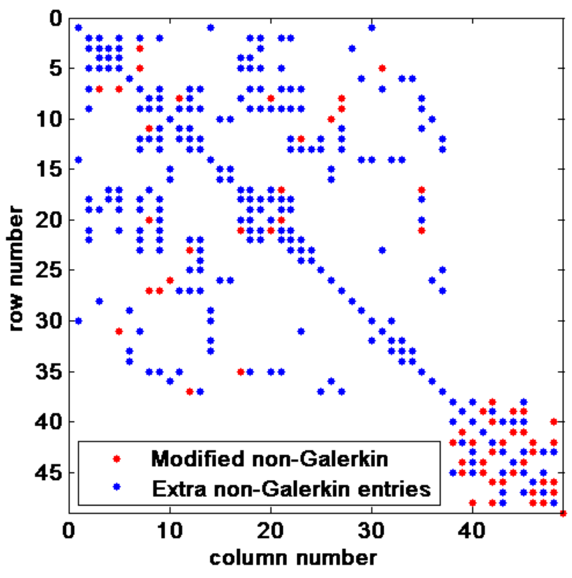
Sparsity

For the modified version of the non-Galerkin algorithm, seven coarse levels are constructed by the AMG setup. Starting from the second coarse level, the modified version gradually reduces the number of non-zero entries of the non-Galerkin method (see Figure C.5b). The effect on the sparsity pattern of the non-Galerkin matrices is shown in Figure 5.24.

Symmetry



(a) Third coarse level.



(b) Fourth coarse level

Figure 5.24: Sparsity patterns. The blue dots form sparsity pattern of the modified non-Galerkin operator, whereas the combination of the blue and red dots represents the sparsity of the standard non-Galerkin operator.

On the first, fifth and sixth coarse levels, the modified operators are symmetric in structure and non-symmetric in value. On the seventh level, the matrix is symmetric, because it is a diagonal matrix. The remaining matrices are non-symmetric in vaue and structure.

M-matrix properties

We already know that the non-Galerkin method transforms the Galerkin matrix into an M-matrix on the first grid, but fails to improve the properties of the matrices on coarser grids. As expected, the modified non-Galerkin method generates operators with positive diagonal and negative off-diagonal entries on all levels. In addition, the modified matrices have entrywise positive inverses. Thus, the operators of the modified algorithm are M-matrices.

Spectrum

The spectrum of the Galerkin operator is hardly affected by the modified non-Galerkin algorithm. The approximated eigenvalues are illustrated in Figure C.5b.

Chapter 6

Conclusions

Both aggressive coarsening and the non-Galerkin algorithm are successfully applied to the AMG algorithm in INTERSECT. Generally, aggressive coarsening shows high performance for serial runs, but is less effective when a parallel machine is used. The non-Galerkin method has not been tested for parallel simulations, but its serial results are very promising.
In the remaining part of this section, we draw further conclusions from the results, stated in Chapter 5, and present suggestions for future research.

6-1 Aggressive Coarsening

Typically, the application of aggressive coarsening leads to a slower convergence of the linear solver, compared to the standard approach. This holds for both serial and parallel simulations. For serial runs without heuristics, aggressive coarsening frequently reduces the time required for a linear iteration. Thus, under the condition that the number of linear iterations is not extremely high, the linear solver time is reduced.

Unfortunately, for parallel runs, the effect produced by aggressive coarsening on the linear iteration time is opposite. This can be caused by unsuitable heuristics. However, it has been noticed that the solver, obtained by the coupling of INTERSECT with SAMG, is poorly scalable. Hence, the time rise can also originate from a communication issue. We provide several plausible reasons for the increase in communication among the processor cores.
Firstly, the extensive data exchange can arise due to a low number of coarse levels, which usually leads to a large number of non-zero entries on the coarsest level. Since Gauss elimination is not highly scalable, this can cause a significant increase in the linear solver time.
Secondly, the behaviour of the subdomain blocking may substantially differ from that of the PMIS algorithm. If the coarsening scheme selects many variables on the subdomain boundaries, the exchange of large data sets is inevitable.
Thirdly, the interplay between INTERSECT and SAMG may negatively affect parallel simulations, if extremely large data sets have to be transfered.
Finally, aggressive coarsening requires the use of long range interpolation, which is assumed

to be efficient for serial computations, but leads to communication beyond neighbouring processors [12]. Obviously, this should reduce the effectiveness of the matrix storage structure used in SAMG and, therefore, increase the execution time.
Clearly, the increase in the linear solver time can result from the combination of some or all factors presented above.

In general, the rise of the total time due to aggressive coarsening is additionally driven by the time required for the linearization.

6-2 Non-Galerkin

The results, obtained with the non-Galerkin method for serial simulations, are promising for parallel runs. As shown in Chapter 5, the density of the coarse grid grid operators is effectively reduced by the algorithm. In addition, the number of linear iterations, that is necessary for the non-Galerkin approach, is very limited.

The increase in the total execution time, associated with the algorithm, is caused by the non-Galerkin setup phase and, to some extent, by the additional linearization time. The computation of the minimal sparsity pattern within the lumping process proves to be the most expansive component of the non-Galerkin setup. The high costs arise from one of the standard functions in INTERSECT, the multiplication of three sparse matrices of large size.

The modified non-Galerkin method improves the M-matrix properties of the non-Galerkin coarse grid operators. The investigation of the modified coarse level operators of two small cases reveals that the matrices satisfy all M-matrix properties. More importantly, the application of the algorithm decreases the linear solver and total time of the standard non-Galerkin algorithm. In terms of time, the performance of the modified non-Galerkin method is frequently higher than that of the standard Galerkin method without heuristics.

6-3 Future Research

Further research on both aggressive coarsening and non-Galerkin solution strategies is highly recommended.

The proper implementation of aggressive coarsening, as an optional part of the coarsening scheme, should allow an in-depth analysis of the algorithm's behaviour with regard to pressure equations. In addition, it should prevent the inconsistencies in the performance observed with the INTERSECT - SAMG solver.
Another interesting question, which has not been explored in this thesis, is the actual aggressiveness of the coarsening. From the results in Chapter 5 follows that the investigated approach eliminates the majority of the coarse grid levels, creating low complexities at the expense of the accuracy. The aggressiveness can be adapted by replacing A1 coarsening by a less aggressive version, such as A2 coarsening. In addition, the number of coarse levels on which aggressive coarsening is applied can be reduced to one.

To study the influence of the non-Galerkin algorithm on the communication pattern, the parallel version of the code should be created. However, some experiments can be performed with

the existing serial implementation. First of all, the current choice of γ in the Compute sparsity algorithm (Algorithm 5) is based on the experiments with one case. Therefore, additional investigation of the effect of this parameter on the performance of the non-Galerkin method is required. Furthermore, the obtained results reveal that on the coarsest levels the Galerkin algorithm can be more efficient than the non-Galerkin approach. Thus, the restriction of the number of levels, to which the non-Galerkin method is applied, may be used to optimize the structure of the AMG hierarchy.
Undoubtedly, the modified non-Galerkin algorithm requires further theoretical and experimental research. The fact that enforcing the absence of the positive off-diagonal entries in the coarse grid matrices seems to outweigh the principles of the construction of the non-Galerkin sparsity pattern, should be closely investigated.

In Chapter 5, we noted that the use of heuristics improves the performance of both solution strategies. However, the reduction of the linear solver and total execution time, associated with aggressive coarsening and the non-Galerkin method, was smaller than that for of the Galerkin approach. For this reason, we suggest further optimization of the heuristics for aggressive coarsening and the non-Galerkin algorithm.
Finally, future research can be conducted on the technique, that combines the non-Galerkin method with the aggressive coarsening strategy, in order to reduce the communication between processors.

Appendix A

Well Modeling

Within INTERSECT we distinguish between three types of wells: producer, injector and observer wells [35]. Each well communicates with the grid cells of the reservoir through at least one well-to-cell connection, also known as connecting reservoir cell. A well-to-cell connection represents the flow path between the well and a single reservoir cell.

Producer wells

Producer wells are used for producing oil to the surface. If the current reservoir conditions do not allow oil production, e.g. if the pressure in the connecting reservoir cells is below a certain limit, the producer wells should be closed.

Injector wells

The purpose of the injector wells is to inject fluids into the reservoir. The injection stream describes the composition of the fluid injected from the surface and should be defined for each injector well. Similarly to the producer wells, the injector wells are closed if no fluid can be injected at the current reservoir conditions.

Observer wells

Contrary to producer an injector wells, the observer wells do not exchange fluids with the reservoir. They are used to allow convenient access to particular properties inside the connecting reservoir grid cells.

Wells are modeled by means of segments. A segment is analogous to a grid cell, except that it corresponds to a part of wellbore rather than the reservoir. Throughout the segment, the properties and composition of the fluid mixture are uniform.
For more details on the well modeling we refer to [35].

Appendix B

Krylov Subspace Methods

In order to introduce Krylov subspace methods, we consider the main idea behind the basic iterative methods.

Equation (2.14) can be rewritten as $M\mathbf{u} = N\mathbf{u} + \mathbf{f}$, where M is a nonsingular matrix and $N = M - A$. Therefore, at step $k+1$ of a basic iterative method, the numerical approximation of $\mathbf{u}$, denoted by $\mathbf{v}$ is equal to

$$\mathbf{v}^{k+1} = M^{-1}\left(N\mathbf{v}^k + \mathbf{f}\right) = \left(I - M^{-1}A\right)\mathbf{v}^k + M^{-1}\mathbf{f} = \mathbf{v}^k + M^{-1}\mathbf{r}^k, \tag{B.1}$$

where $\mathbf{r}^k = \mathbf{f} - A\mathbf{v}^k$.

Developing the iterations, we find

$$\begin{aligned}
&\mathbf{v}^0,\\
&\mathbf{v}^1 = \mathbf{v}^0 + (M^{-1}\mathbf{r}^0),\\
&\mathbf{v}^2 = \mathbf{v}^1 + (M^{-1}\mathbf{r}^1)\\
&\quad = \mathbf{v}^0 + M^{-1}\mathbf{r}^0 + M^{-1}\left(\mathbf{f} - A\mathbf{v}^0 - AM^{-1}\mathbf{r}^0\right)\\
&\quad = \mathbf{v}^0 + 2M^{-1}\mathbf{r}^0 - M^{-1}AM^{-1}\mathbf{r}^0,\\
&\quad \vdots
\end{aligned}$$

It follows that

$$\mathbf{v}^i \in \mathbf{v}^0 + \text{span}\{M^{-1}\mathbf{r}^0, M^{-1}A\left(M^{-1}\mathbf{r}^0\right), \ldots, \left(M^{-1}A\right)^{i-1}(M^{-1}\mathbf{r}^0)\}.$$

Definition B.1. For A defined by Equation (2.14) and $\mathbf{v} \in \mathbb{R}^n$, a Krylov subspace of dimension m $(m < n)$ is defined as

$$\mathcal{K}_m(A, \mathbf{v}) = \text{span}\{\mathbf{v}, A\mathbf{v}, \ldots, A^{m-1}\mathbf{v}\}.$$

Thus, a $\mathbf{v}^i$ obtained by a basic iterative method is an element of $\mathbf{v}^0 + \mathcal{K}_i(M^{-1}A, M^{-1}\mathbf{r}^0)$. Unlike the basic iterative methods, Krylov methods do not have iteration matrix M. They approximate the solution by minimizing the residual over the corresponding Krylov subspace.

Since FGMRES utilizes the Arnoldi method, we introduce Arnoldi first and then describe the FGMRES algorithm.

B-1 Arnoldi

The Arnoldi method transforms matrix A to an upper Hessenberg matrix by orthogonal similarity transformations.

Definition B.2. Matrix $H \in \mathbb{C}^{n\times n}$ is an upper Hessenberg matrix if it has zero entries below the first subdiagonal:

$$H = \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} & \cdots & h_{1,n-3} & h_{1,n-2} & h_{1,n-1} & h_{1,n} \\ h_{2,1} & h_{2,2} & h_{2,3} & \cdots & h_{2,n-3} & h_{2,n-2} & h_{2,n-1} & h_{2,n} \\ 0 & h_{3,2} & h_{3,3} & \cdots & h_{3,n-3} & h_{3,n-2} & h_{3,n-1} & h_{3,n} \\ 0 & 0 & h_{4,3} & \cdots & h_{4,n-3} & h_{4,n-2} & h_{4,n-1} & h_{4,n} \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & h_{n-1,n-2} & h_{n-1,n-1} & h_{n-1,n} \\ 0 & 0 & 0 & \cdots & 0 & 0 & h_{n,n-1} & h_{n,n} \end{bmatrix}.$$

Definition B.3. If $X \in \mathbb{C}^{n\times n}$ is nonsingular, then the map $A \mapsto XAX^{-1}$ is called the similarity transformation of A.

Definition B.4. A similarity transformation is called orthogonal if $X^{-1} = X^*$.

Thus, to fully reduce matrix $A \in \mathbb{C}^{n\times n}$ to Hessenberg form by orthogonal similarity transformations, a matrix Q should be created such that

$$A = QHQ^*, \tag{B.2}$$

where H is a Hessenberg matrix and Q^* is the Hermitian conjugate of Q.

Clearly, Equation (B.2) is equivalent to

$$AQ = QH. \tag{B.3}$$

Since n is assumed to be large, computing fully reduction is generally out of question and only a part of the system $AQ = QH$ is considered. Let $m < n$ and Q_m represent the $n \times m$ matrix whose columns are equal to the first m columns of Q:

$$Q_m = \left[\ \mathbf{q}_1 \ \middle|\ \mathbf{q}_2 \ \middle|\ \dots \ \middle|\ \mathbf{q}_m \ \right]. \tag{B.4}$$

Then the new system can be stated as

$$AQ_m = Q_{m+1}\tilde{H}_m, \tag{B.5}$$

where the Hessenberg matrix $\tilde{H}_m$ is the $(m+1) \times m$ upper-left section of H. That is,

$$\tilde{H}_m = \begin{bmatrix} h_{1,1} & \cdots & h_{1,m} \\ h_{2,1} & \cdots & h_{2,m} \\ & \ddots & \vdots \\ & & h_{m+1,m} \end{bmatrix}.$$

Thus, the mth column of AQ_m can be written as

$$A\mathbf{q}_m = h_{1,m}\mathbf{q}_1 + \cdots + h_{m,m}\mathbf{q}_m + h_{m+1,m}\mathbf{q}_{m+1}. \tag{B.6}$$

Equation (B.6) is equivalent to

$$\mathbf{q}_{m+1} = \frac{A\mathbf{q}_m - \sum_{i=1}^{m} h_{i,m}\mathbf{q}_i}{h_{m+1,m}}. \tag{B.7}$$

The recursive computation of the columns of the matrix Q in this manner is known as Arnoldi iteration. This iteration is simply the modified Gram-Schmidt iteration (see [39]) that implements (B.7). The method is outlined in Algorithm 8.

Algorithm 8: Arnoldi

Data: $\mathbf{q}_1$ is an arbitrary n-vector such that $||\mathbf{q}_1||_2 = 1$.

```
1  for $i$ = 1,2, ..., $m$ do
2      $\mathbf{w} = A\mathbf{q}_i$
3      for $j$ = 1, ..., $i$ do
4          $h_{j,i} = (\mathbf{w}, \mathbf{q}_j)$
5          $\mathbf{w} = \mathbf{w} - h_{j,i}\mathbf{q}_j$
6      end
7      $h_{i+1,i} = ||\mathbf{w}||_2$
8      if $h_{i+1,i} = 0$ then break
9      $\mathbf{q}_{i+1} = \frac{1}{h_{i+1,i}}\mathbf{w}$
10 end
```

The Arnoldi process can also be seen as a computation of an orthogonal projections onto $\mathcal{K}_m(A, \mathbf{r}_0)$.

Definition B.5. A general projection method for solving the linear system $A\mathbf{u} = \mathbf{f}$ is a method that seeks an appropriate solution $\mathbf{v}$ from $\mathbf{v}_0 + \mathcal{K}$ of dimension m by imposing the following condition:

$$\mathbf{f} - A\mathbf{v} \perp \mathcal{L},$$

where $\mathcal{L}$ is a subspace of dimension m.

Regarding the projection onto the Krylov space, the most important properties of the Arnoldi process are stated below.

Proposition B.1. *Under the assumption that Algorithm 8 does not break down before the m-th step, the vectors* $\{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_m\}$ *form an orthonormal basis of the Krylov space* $\mathcal{K}_m(A, \mathbf{r}_0)$.

Proof: see (Proposition 6.4, [31]).

Proposition B.2. *The Hessenberg matrices* $H_m = Q_m^* A Q_m$ *are the orthogonal projections of* A *onto* $\mathcal{K}_m(A, \mathbf{r}_0)$ *with the columns of* Q_m *as basis.*

Proof: see (Theorem 33.1, [39]).

B-2 GMRES

GMRES, which stands for generalized minimal residual, was proposed by Saad and Schultz in 1986 in order to solve large, sparse and nonsymmetric (or non Hermitian) linear systems, [33], [17]. It is a projection method with $\mathcal{K} = \mathcal{K}_m(A, \mathbf{r}_0)$ and $\mathcal{L} = A\mathcal{K}_m$. The algorithm approximates the exact solution of the linear system (2.14) by solving a least squares problem at each step of the iteration. More precisely, at step m, it determines vector $\mathbf{v}_m \in \mathbf{v}_0 + \mathcal{K}_m(A, \mathbf{r}_0)$ that minimizes

$$||\mathbf{r}_m||_2 = ||\mathbf{f} - A\mathbf{v}_m||_2.$$

Definition B.6. K_m is the $n \times m$ Krylov matrix corresponding to $\mathcal{K}_m(A, \mathbf{r}_0)$ if

$$K_m = \left[\begin{array}{c|c|c|c} \mathbf{r}_0 & A\mathbf{r}_0 & \ldots & A^{m-1}\mathbf{r}_0 \end{array} \right].$$

Obviously, the minimization problem can be rewritten as

$$||\mathbf{f} - A(\mathbf{v}_0 + K_m\mathbf{y})||_2$$

with $\mathbf{y} \in \mathbb{C}^m$.

From Proposition B.1 follows that

$$||\mathbf{f} - A(\mathbf{v}_0 + K_m\mathbf{y})||_2 = ||\mathbf{f} - A(\mathbf{v}_0 + Q_m\mathbf{y})||_2,$$

where matrix Q_m is defined by Equation (B.4).

Using Equation (B.5), the relation results in

$$\begin{aligned} ||\mathbf{f} - A\left(\mathbf{v}_0 + Q_m\mathbf{y}\right)||_2 &= ||\mathbf{r}_0 - AQ_m\mathbf{y}||_2 \\ &= ||\beta\mathbf{q}_1 - Q_{m+1}\tilde{H}_m\mathbf{y}||_2 \\ &= ||Q_{m+1}\left(\beta\mathbf{e}_1 - \tilde{H}_m\mathbf{y}\right)||_2. \end{aligned}$$

where $\beta = ||\mathbf{r}_0||_2$, $\mathbf{q}_1 = \frac{\mathbf{r}_0}{\beta}$ and $\mathbf{e}_1$ denotes the first unit vector in $\mathbb{R}^{m+1}$.

The final simplification is based on the fact that Q_{m+1} is a unitary matrix:

$$||Q_{m+1}\left(\beta\mathbf{e}_1 - \tilde{H}_m\mathbf{y}\right)||_2 = ||\beta\mathbf{e}_1 - \tilde{H}_m\mathbf{y}||_2 \tag{B.8}$$

Thus, at step m Equation (B.8) is solved for $\mathbf{y}$. After that the numerical solution is obtained:

$$\mathbf{v}_m = \mathbf{v}_0 + Q_m\mathbf{y}.$$

A description of GMRES algorithm is provided in Algorithm 9.

Algorithm 9: GMRES

```
Data: Initial guess $v_0$.
1  Setup $(m+1) \times m$ zero-matrix $\tilde{H}_m$
2  Compute $\mathbf{r}_0 = \mathbf{f} - A\mathbf{v}_0$ and $\mathbf{q}_1 = \frac{\mathbf{v}_0}{||\mathbf{v}_0||}$.
3  for $i = 1,2, \ldots, m$ do
4      $\mathbf{w} = A\mathbf{q}_i$
5      for $j = 1, \ldots, i$ do
6          $h_{j,i} = (\mathbf{w}, \mathbf{q}_j)$
7          $\mathbf{w} = \mathbf{w} - h_{j,i}\mathbf{q}_j$
8      end
9      $h_{i+1,i} = ||\mathbf{w}||_2$
10     if $h_{i+1,i} = 0$ then $m = i$ and break
11     $\mathbf{q}_{i+1} = \frac{1}{h_{i+1,i}}\mathbf{w}$
12 end
13 Define $Q_m = \left[\ \mathbf{q}_1 \mid \mathbf{q}_2 \mid \ldots \mid \mathbf{q}_m\ \right]$.
14 Compute $\mathbf{v}_m = \mathbf{v}_0 + Q_m\mathbf{y}_m$ where $\mathbf{y}_m$ minimizes $\mathbf{y}_m = ||\beta\mathbf{e}_1 - \tilde{H}_m\mathbf{y}||_2$.
15 if convergence then stop
16 $\mathbf{v}_0 = \mathbf{v}_m$ and go to line 2.
```

The basic properties of the GMRES iteration are stated in Proposition B.3 and B.4.

Proposition B.3. *Under the assumption that A is a nonsingular matrix, the GMRES algorithm breaks down at step i, i.e. $h_{i+1,i} = 0$, if and only if the approximate solution $\mathbf{v}_m$ is exact.*

Proof: see (Proposition 6.10, [31]).

Proposition B.4. *For $m = 1, 2, \ldots,$ GMRES solves the following approximation problem successively:*

Find $p_m \in P_m = \{$polynomials p of degree $\leq m$ with $p(0) = 1\}$ such that

$$||p_m(A)\mathbf{f}||_2$$

is minimized.

Proof: see (p.268-269, [39]).

From Proposition B.4 follows that $||\mathbf{r}_m||_2 = ||p_m(A)\mathbf{f}||_2$. Therefore, $||\mathbf{r}_m||_2 \leq ||p_m(A)||_2||\mathbf{f}||_2$. Generally, the crucial factor that controls the size of this quantity is $||p_m(A)||_2$. Thus, the convergence of GMRES is usually determined by

$$\frac{||\mathbf{r}_m||_2}{||\mathbf{f}||_2} \leq \inf_{p_m \in P_m} ||p_m(A)||_2.$$

Theorem B.5. *Suppose that A is diagonalizable, satisfying $A = VDV^{-1}$ for a diagonal matrix D. At step m of the GMRES iteration, the residual satisfies*

$$\frac{||\mathbf{r}_m||_2}{||\mathbf{f}||_2} \leq \inf_{p_m \in P_m} ||p_m(A)||_2 \leq \kappa(V) \inf_{p_m \in P_m} ||p_m||_{D(A)},$$

where $D(A)$ is the set of eigenvalues of A, V is a nonsingular matrix of eigenvectors, and $||p_m||_{D(A)} = \sup_{z\in D(A)} |p_m(z)|$.

B-3 Preconditioned GMRES

Using GMRES with a preconditioner can significantly improve convergence and robustness. A preconditioner transforms the original system (2.14) into an equivalent system with the same solution, while improving the properties important for iterative methods, e.g. the condition number of the matrix. We distinguish between left and right preconditioning.

$$\begin{aligned} &\text{Left preconditioning:} \quad M^{-1}A\mathbf{u} = M^{-1}\mathbf{f}. \\ &\text{Right preconditioning:} \; AM^{-1}\bar{\mathbf{x}} = \mathbf{f}, \;\; \bar{\mathbf{x}} = M\mathbf{u}. \end{aligned}$$

The operator M should satisfy the following requirements:

- $M^{-1}\mathbf{z}$, where $\mathbf{z} \in \mathbb{R}^n$, should inexpensive to obtain,
- the eigenvalues of the preconditioned matrix should be clustered around 1.

In preconditioned GMRES, Arnoldi is invoked to create an orthonormal basis for the respective Krylov subspaces

$$\begin{aligned} \mathcal{K}_m(M^{-1}A, M^{-1}\mathbf{r}_0) &= \text{span}\{M^{-1}\mathbf{r}^0, M^{-1}AM^{-1}\mathbf{r}^0, \ldots, \left(M^{-1}A\right)^{m-1} M^{-1}\mathbf{r}^0\}, \\ \mathcal{K}_m(AM^{-1}, \mathbf{r}_0) &= \text{span}\{\mathbf{r}^0, AM^{-1}\mathbf{r}^0, \ldots, \left(AM^{-1}\right)^{m-1} \mathbf{r}^0\}. \end{aligned}$$

To construct a left preconditioned GMRES, we adjust Algorithm 9 as follows. The residual $\mathbf{r}_0$ on line 2 is replaced by $\mathbf{r}_0 = M^{-1}(\mathbf{f} - A\mathbf{v}_0)$, while $\mathbf{w}$ on line 4 is changed to $\mathbf{w} = M^{-1}A\mathbf{q}_i$.

For right preconditioning, the residual becomes

$$\mathbf{r}_0 = \mathbf{f} - A\mathbf{v}_0 = \mathbf{f} - AM^{-1}\bar{\mathbf{x}}_0.$$

Thus, the initial residual can remain equal to $\mathbf{f} - A\mathbf{v}_0$, whereas $\mathbf{v}_m = \mathbf{v}_0 + Q_m\mathbf{y}_m$ has to be multiplied by M^{-1} to obtain

$$\mathbf{v}_m = \mathbf{v}_0 + M^{-1}Q_m\mathbf{y}_m.$$

To sum this up, we provide the algorithm for right-preconditioned GMRES.

Algorithm 10: Right-preconditioned GMRES

```
Data: Initial guess $v_0$.
1  Setup $(m+1) \times m$ zero-matrix $\tilde{H}_m$
2  Compute $\mathbf{r}_0 = \mathbf{f} - A\mathbf{v}_0$ and $\mathbf{q}_1 = \frac{\mathbf{v}_0}{||\mathbf{v}_0||}$.
3  for $i$ = 1,2, ..., $m$ do
4      $\bar{\mathbf{w}}_i = M^{-1}\mathbf{q}_i$
5      $\mathbf{w} = A\bar{\mathbf{w}}_i$
6      for $j$ = 1, ..., $i$ do
7          $h_{j,i} = (\mathbf{w}, \mathbf{q}_j)$
8          $\mathbf{w} = \mathbf{w} - h_{j,i}\mathbf{q}_j$
9      end
10     $h_{i+1,i} = ||\mathbf{w}||_2$
11     if $h_{i+1,i} = 0$ then $m = i$ and break
12     $\mathbf{q}_{i+1} = \frac{1}{h_{i+1,i}}\mathbf{w}$
13 end
14 Define $Q_m = \left[\ \mathbf{q}_1 \ | \ \mathbf{q}_2 \ | \ \dots \ | \ \mathbf{q}_m\ \right]$.
15 Compute $\mathbf{v}_m = \mathbf{v}_0 + M^{-1}Q_m\mathbf{y}_m$ where $\mathbf{y}_m$ minimizes $\mathbf{y}_m = ||\beta\mathbf{e}_1 - \tilde{H}_m\mathbf{y}||_2$.
16 if convergence then stop
17 $\mathbf{v}_0 = \mathbf{v}_m$ and go to line 2.
```

B-4 Ritz Values and Ritz Vectors

Definition B.7. Let H_m be given as in Lemma B.2. Then, the eigenvalues θ_i of H_m are called Ritz values of A with respect to $\mathcal{K}_m(A, \mathbf{r}_0)$.

Definition B.8. Let H_m and Q_m be given as in Lemma B.2. If z_i is an eigenvector of H_m such that $H_m z_i = \theta_i z_i$ and $||z_i||_2 = 1$, then $Q_m z_i$ is called a Ritz vector of A.

Ritz values and Ritz vectors are approximations of the actual eigenvalues of A. At each step i of the Arnoldi iteration, the eigenvalues of the Hessenberg matrix are obtained using standard methods such as QR algorithm. Since $m \ll n$ for a feasible computation, it cannot be expected that all eigenvalues of A will be computed by this process. In general, Ritz values correspond to the extreme eigenvalues of A, i.e. eigenvalues near the edge of the spectrum of A [39].

Remark 12. For the right preconditioned GMRES, Ritz values and Ritz vectors are approximations of the eigenvalues of $M^{-1}A$.

Appendix C

Results

C-1 Aggressive Coarsening

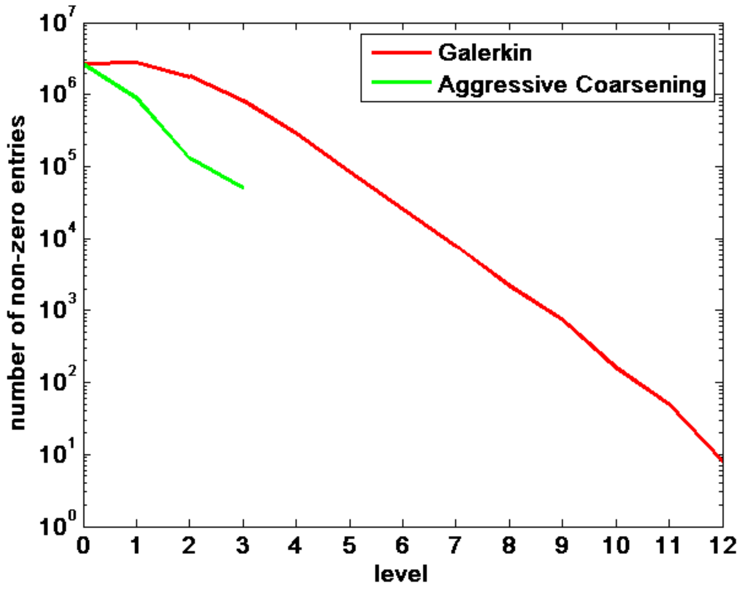


Figure C.1: The number of non-zero entries per level, the 389559-case.

Algorithm	Case				
	389559	348809	348811	1722780	164944
Galerkin	1165	2337	3036	447	28418
Aggressive coarsening	1186	2290	3033	539	28397
Galerkin H	1173	2313	3148	465	28405
Aggressive coarsening H	1184	2287	3121	479	28399
Galerkin H P	1188	2293	3073	457	unknown
Aggressive coarsening H P	1213	2316	3022	450	unknown

Table C-1: Number of time steps with the Galerkin and aggressive coarsening method. H denote a strategy with heuristics, while P corresponds to parallel simulations.

Algorithm	Case				
	389559	348809	348811	1722780	164944
Galerkin	7296	5783	12701	1689	30820
Aggressive coarsening	7574	5742	12982	2465	30760
Galerkin H	7304	5668	13411	1767	30804
Aggressive coarsening H	7570	5712	15121	1944	30770
Galerkin H P	7454	5759	13860	1795	unknown
Aggressive coarsening H P	7904	5813	12785	1738	unknown

Table C-2: Number of non-linear iterations with the Galerkin and aggressive coarsening method. H denotes a strategy with heuristics, while P corresponds to parallel simulations.

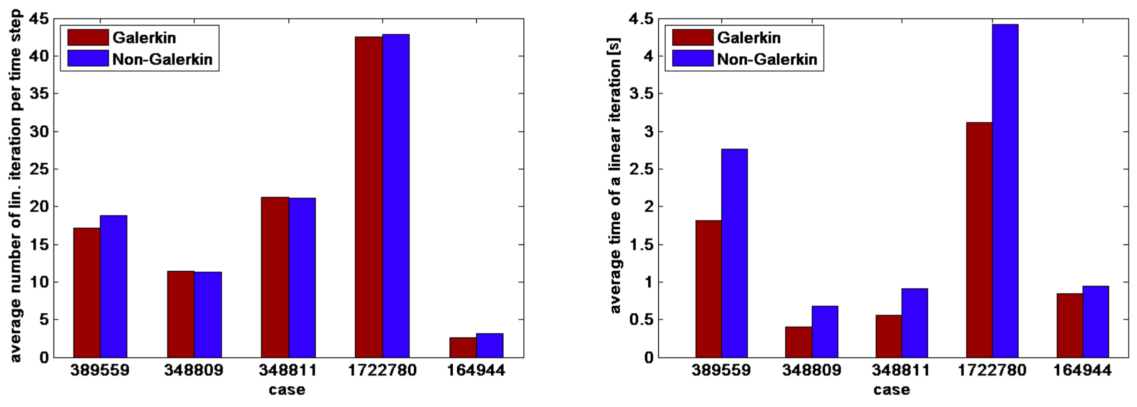
C-2 Non-Galerkin

Algorithm	Case				
	389559	348809	348811	1722780	164944
Galerkin	1165	2337	3036	447	28418
Non-Galerkin	1312	2276	3090	456	28487
Galerkin H	1173	2313	3148	465	28405
Non-Galerkin H	1618	2308	3187	465	28494
Modified non-Galerkin	1582	2319	3208	463	28403

Table C-3: Number of time steps with the Galerkin and non-Galerkin coarse grids. H denotes a strategy with heuristics.

Algorithm	Case				
	389559	348809	348811	1722780	164944
Galerkin	7296	5783	12701	1689	30820
Non-Galerkin	9394	5626	13609	1778	31118
Galerkin H	7304	5668	13411	1767	30804
Non-Galerkin H	12683	5855	15230	1768	31068
Modified non-Galerkin	12678	5834	14695	1750	30779

Table C-4: Number of non-linear iterations with the Galerkin and non-Galerkin coarse grids. H denotes a strategy with heuristics.



(a) **(b)**

Figure C.2: Linear solver statistics for the Galerkin and non-Galerkin methods: (a) average number of linear iterations per time step, (b) average time of a linear iteration.

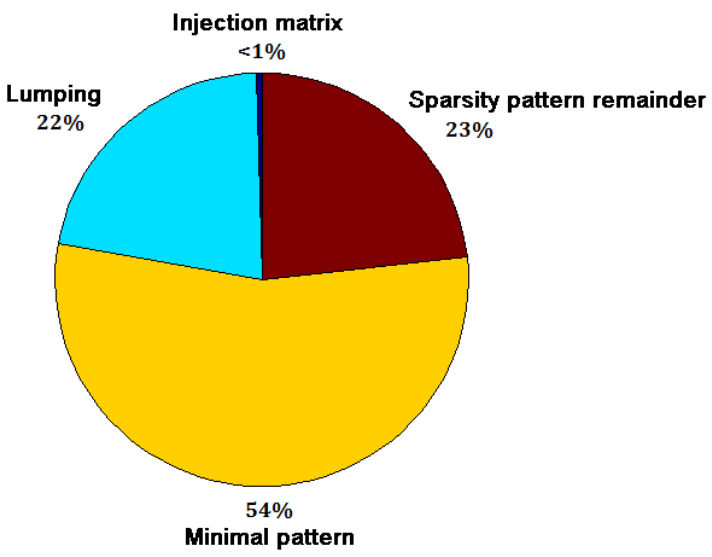


Figure C.3: Sources of the total time growth due to the use of the non-Galerkin coarse grid operators, 348809-case.

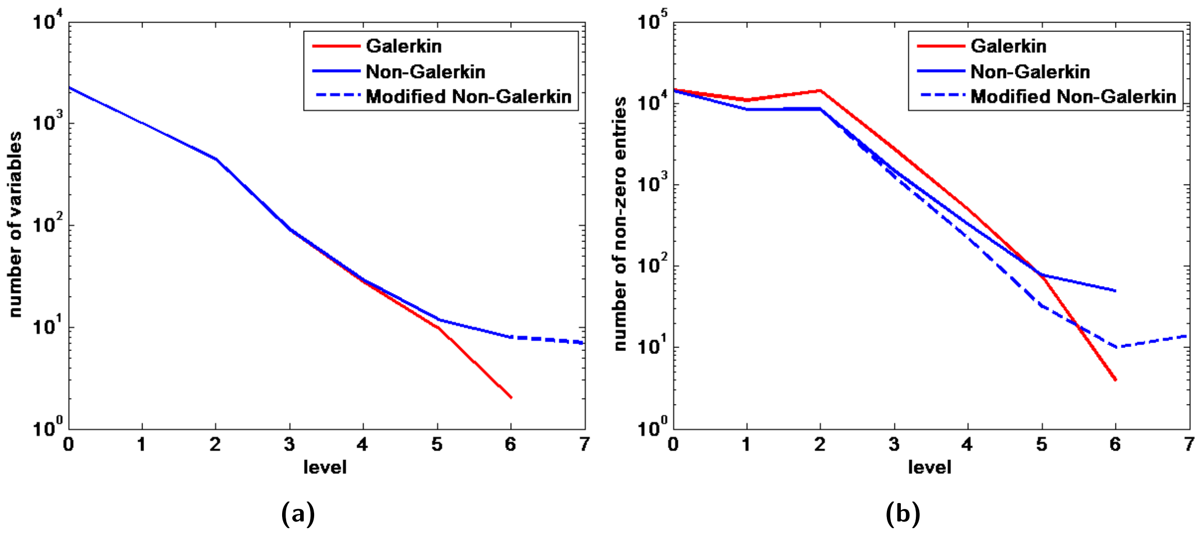
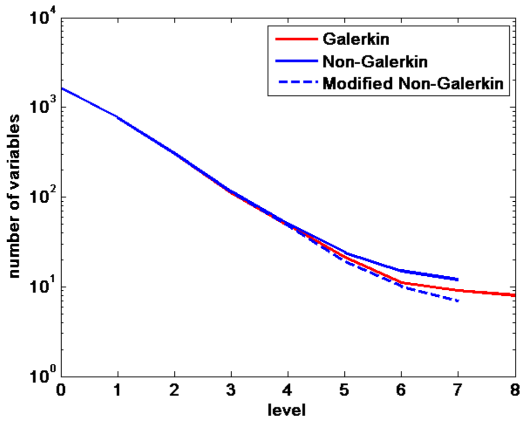
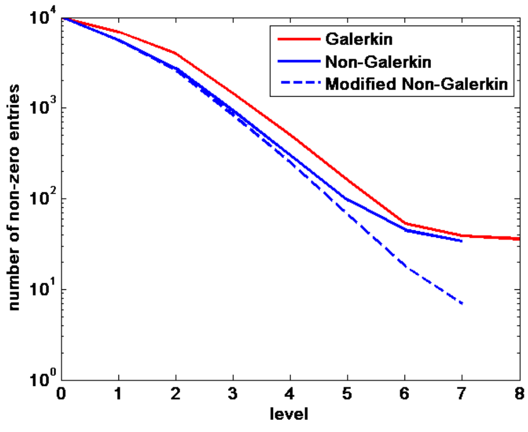


Figure C.4: AMG hierarchy, 2250-case: (a) number of variables, (b) number of non-zero elements.

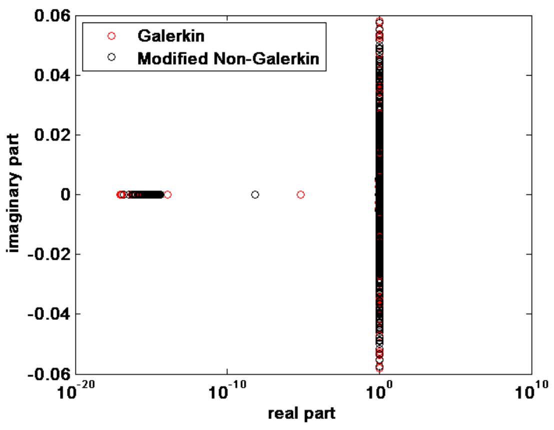


(a)

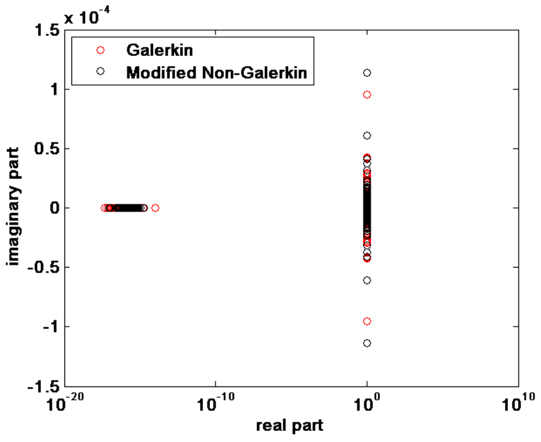


(b)

Figure C.5: AMG hierarchy, 1639-case: (a) number of variables, (b) number of non-zero elements.



(a)



(b)

Figure C.6: The approximated spectrum of the pressure matrix preconditioned by one AMG V-cycle with the Galerkin and modified non-Galerkin coarse grid operators after the first linear iteration (a) for the 2250-case , (b) for the 1639-case.

Bibliography

[1] Antonelli, M., Chartier, T., *Improving Algebraic Multigrid Efficiency For Immersed Interface Problems*, International Journal of Pure and Applied Mathematics, Vol. 10(4), 365-385, 2004.

[2] Ashby, S., Falgout, R., *A Parallel Multigrid Preconditioned Conjugate Gradient Algorithm for Groundwater Flow Simulations*, Nuclear Science and Engineering, Vol. 124(1), 145-159, 1996.

[3] Brandt, A., *Algebraic Multigrid Theory: The Symmetric Case*, Applied Mathematics and Computation, Vol. 19, 23-56, 1986.

[4] Brandt, A., McCormick, S., Ruge, J., *Algebraic Multigrid (AMG) for Automatic Multigrid Solutions with Application to Geodatic Computations*, Technical report, Institute for Computational Studies, Fort Coolins, CO, 1982.

[5] Brandt, A., McCormick, S., Ruge, J., *Algebraic Multigrid (AMG) for Sparse Matrix Equations*, In Evans, D., editor, Sparsity and Its Applications, Cambridge University Press, 1984.

[6] Briggs, W., Henson, V., McCormick, S., *A Multigrid Tutorial*, Second Edition, SIAM, Philadelphia, 2000.

[7] Brualdi, R., Mellendorf, S., *Regions in the Complex Plane Containing the Eigenvalues of a Matrix*, The American Mathematical Monthly, Vol. 101, 975-985, 1994.

[8] Coats, K., Thomas, L., Pierson, R., *Compositional and Black Oil Reservoir Simulation*, SPE Reservoir Evaluation and Engineering, Vol. 1(4), 372-379, 1998.

[9] Chen, A., Huan, G., Ma, Y., *Computational Methods for Multiphase Flow in Porous Media*, Computational Science and Engineering, SIAM, Philadelphia, 2006.

[10] Clees, T., *AMG Strategies for PDE Systems with Applications in Industrial Semiconductor Simulation*, Ph.D. dissertation, University of Cologne, Germany, 2005.

[11] Darwish, M., Saad, T., Hamdan, Z., *A High Scalability Parallel Algebraic Multigrid Solver*, European Conference on Computational Fluid Dynamics, ECCOMAS CFD 2006, P. Wesseling, E. Oñate, J. Périaux (Eds), TU Delft, The Netherlands, 2006.

[12] De Sterck, H., Yang, U., Heys, J., *Reducing Complexity in Parallel Algebraic Multigrid Preconditioners*,SIAM Journal on Matrix Analysis and Applications, Vol. 27, 1019-1039, 2006.

[13] Dong, X., Cooperman, G., *Scalable Task-Oriented Parallelism for Structure Based Incomplete LU Factorization*, CoRR, abs/0803.0048 , 2008.

[14] Falgout, R., *An Introduction to Algebraic Multigrid*, Computing in Science and Engineering, Vol. 8(6), 24-33, 2006.

[15] Falgout, R., Schroeder, J., *Non-Galerkin Coarse Grids for Algebraic Mutigrid*, SIAM Journal on Scientific Computing, Vol. 36 (3), 309-334, 2014.

[16] Foster, I., *Designing and Building Parallel Programs*, http://www.mcs.anl.gov/∼itf/dbpp/, 1995.

[17] Frayssé, V., Giraud, L., Gratton, S., *A Set of Flexible-GMRES Routines for Real and Complex Arithmetics on High Performance Computers*, ACM Transactions on Mathematical Software, Vol. 35(2), 2008.

[18] Gahvari, H., Baker, A., Schulz., M, Yang, U., Jordan, K., Gropp, W., *Modeling the Performance of an Algebraic Multigrid Cycle on HPC Patforms*, Proceedings of the international conference on Supercomputing, ICS '11, 172-181, New York, NY, USA, 2011.

[19] Gallier, J., *The Schur Complement and Symmetric Positive Semidefinite (and Definite) Matrices*, http://www.cis.upenn.edu/∼jean/schur-comp.pdf, 2010.

[20] George, A., Liu, J., *Computer Solution of Large Sparse Positive Definite Systems*, Englewood Cliffs, N.J.: Prentice Hall, Inc., 1981.

[21] Golub,G., Van Loan, C., *Matrix Computations*, Third Edition, Johns Hopkins University Press, Baltimore, MD, 1996.

[22] Hajibeygi, H., Wang, Y., Tchelepi, H., *Monotone Multiscale Finite Volume Method for Flow in Heterogeneous Porous Media*, ECMOR XIV - 14th European Conference on the Mathematics of Oil Recovery Catania, Sicily, Italy, 8-11 September, 2014.

[23] Henson, V., Yang, U., *BoomerAMG: A Parallel Algebraic Solver and Preconditioner*, Applied Numerical Mathematics, Vol. 41, 155-177, 2002.

[24] Jenny, P., Lee, S., Tchelepi, H., *Adaptive Fully Implicit Multi-scale Finite-volume Method for Multi-phase Flow and Transport in Heterogeneous Porous Media*, Elsevier, Journal of Computational Physics, Vol. 217, 627-641, 2006.

[25] Joubert, W., Cullum, J., *Scalable Algebraic Multigrid on 3500 Processors*, Los Alamos National Laboratory, Technical Report No. LA UR03-568, Submitted to Electronic Transactions on Numerical Analysis, 2003.

[26] Kaminsky, A., *Big CPU, Big Data: Solving the World's Toughest Computational Problems with Parallel Computing*, Rochester Institute of Technology, http://http://www.cs.rit.edu/∼ark/bcbd/, 2014.

[27] Krechel, A., Stüben, K., *Operator Dependent Interpolation in Algebraic Multigrid*, Proceedings of the Fifth European Multigrid Conference, Stuttgart, October 1-4, 1996, Lecture Notes in Computational Science and Engineering, Vol. 3, Springer, Berlin, 1998.

[28] Notay, Y., *An Aggregation-based Algebraic Multigrid Method*,Electronic Transactions on Numerical Analysis, Vol. 37, 123-146, 2010.

[29] Press, W., Teukolsky, S., Vetterling, W., Flannery, B., *Numerical Recepies in C: The Art of Scientific Computing*, Second Edition, ISBN 0-521-43108-5, Cambridge University Press, Cambridge, 1988-1992.

[30] Ruge, J., Stüben, K., *Algebraic Multigrid (AMG)*, In McCormick, S. editor, Multigrid methods, Frontiers in Applied Mathematics, Vol. 3, 73-130, SIAM, Philadelphia, 1987.

[31] Saad, Y., *Iterative Methods for Sparse Linear Systems*, Second Edition, SIAM, Philadelphia, 2003.

[32] Saad, Y., *A Flexible Inner-outer Preconditioned GMRES Algorithm*, SIAM Journal on Scientific Computing, Vol. 14, 461-469, 1993.

[33] Saad, Y., Schultz, M., *GMRES: A Generalized Minimal Residual Algorithm for Solving Non-symmetric Linear Systems*, SIAM Journal on Scientific and Statistical Computing, Vol. 7, 856-869, 1986.

[34] SAMGp: Algebraic multigrid methods for systems, http://www.scai.fraunhofer.de/samg.htm.

[35] Schlumberger, Chevron and Total, *Intersect Technical Description*, Version 2013.1, Houston, TX, 2013.

[36] Sibai, F., *Performance modeling and analysis of parallel Gaussian elimination on multicore computers*, Journal of King Sand University - Computer and Information Sciences, Vol. 26, 41-54, 2014.

[37] Stüben, K., *Algebraic Multigrid (AMG) An Introduction with Applications*, In: Hackbusch, U., Oosterlee, C., Schueller, A. (eds) Multigrid, Academic Press, San Diego, 2000.

[38] Stüben, K., Cless, T., *Algebraic Multigrid Method (AMG) for the Efficient Solution of Fully Implicit Formulations in Reservoir Simulation*, SPE Reservior Simulation Symposium Houston Feb. 26-28, 2007.

[39] Trefethen, L., Bau, D., *Numerical Linear Algebra*, SIAM, Philadelphia, 1997.

[40] Vassilevski, P., Yang, U., *Reducing Communication in Algebraic Multigrid Using Additive Variants*, Numerical Linear Algebra with Applications, Vol. 21 (2), 275-296, 2014.

[41] Vuik, C., Lahaye, D., *Scientific Computing (wi4201)*, Delft University of Technology, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft Institute of Applied Mathematics, the Netherlands, 2013.

[42] Wienands, R., Yvneh, I., *Collocation Coarse Approximation (CCA) in Multigrid*, SIAM Journal in Scientific Computing, Vol. 31, 3643-3660, 2009.

[43] Yang, U., *On Long-range Interpolation Operators for Aggressive Coarsening*, Numerical Linear Algebra with Applications, Vol. 17, 453-472, 2010.

[44] Yang, U., *Parallel Algebraic Multigrid Methods - High Performance Preconditioners*, In Numerical Solutions of Partial Differential Equations on Parallel Computers, Bruaset A. and Tveito A. (eds.), Lecture Notes in Computational Science and Engineering, Spinger-Verlag, 209-236, 2006.