Algebraic Multigrid for Meshfree Methods

Dissertation

zur

Erlangung des Doktorgrades (Dr. rer. nat.)

der

Mathematisch-Naturwissenschaftlichen Fakultät

der

Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von

Fabian Pascal Nick

aus

Wuppertal

Bonn 2019

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Rheinischen Friedrich-Wilhelms-Universität Bonn

1. Gutachter: Prof. Dr. Marc Alexander Schweitzer

2. Gutachter: Prof. Dr. Jochen Garcke

Tag der mündlichen Prüfung: 10. Januar 2020

Erscheinungsjahr: 2020

Zusammenfassung

Diese Dissertation beschäftigt sich mit der Entwicklung einer neuen Algebraischen Mehrgittermethode für die Lösung linearer Gleichungssysteme aus Generalisierten Finite Differenzen Methoden. Im Speziellen betrachten wir die sogenannte Finite Pointset Method, eine gitterfreie Lagrange Methode, welche auf Generalisierten Finite Differenzen Methoden basiert. Die Finite Pointset Method wurde insbesondere für Simulationen von Vorgängen mit freien Oberflächen und bewegten Geometrien entwickelt, bei denen der gitterfreie Charakter der Methode besonders große Vorteile liefert: An den freien Oberflächen und nahe der Geometrie muss zu keinem Zeitpunkt – auch nicht zu Beginn der Simulation – ein Gitter erstellt oder angepasst werden. Dies ist ein großer Vorteil gegenüber klassischen gitterbasierten Methoden.

Wie in gitterbasierten Methoden entstehen auch in der Finite Pointset Method und anderen Generalisierten Finite Differenzen Methoden große, dünn besetze lineare Gleichungssysteme. Das Lösen dieser Gleichungssysteme wird bei fein aufgelösten Simulationen, wie sie in der Industrie oft nötig sind, schnell zum zeitlichen Flaschenhals der Gesamtsimulation. Ohne eine geeignete Methode zur Lösung dieser Gleichungssysteme dauern Simulationen oft sehr lange oder sind praktisch nicht durchführbar. Auch kann es vorkommen, dass klassische Lösungsverfahren divergieren und die Simulation damit unmöglich wird.

Im Kontext von gitterbasierten Methoden sind Mehrgittermethoden ein etabliertes Werkzeug, um die entstehenden linearen Gleichungssysteme effizient und robust zu lösen. Besonders hervorzuheben ist dabei die lineare Skalierbarkeit dieser Methoden in der Größe der Matrix. Damit eignen sie sich besonders für fein aufgelöste Simulationen. Algebraische Mehrgittermethoden sind natürliche Kandidaten für die Lösung der Gleichungssysteme aus Generalisierten Finite Differenzen Methoden, wie diese Dissertation zeigen wird. Außerdem entwickeln wir eine neue Algebraische Mehrgittermethode, die auf den Einsatz in der Finite Pointset Method zugeschnitten ist und die Besonderheiten dieser Methode beachtet. Dazu zählen die Eigenschaften der einzelnen Matrizen, die wir ebenfalls analysieren werden, und auch die Veränderung der Matrizen über mehrere Zeitschritte hinweg, die im Vergleich mit gitterbasierten Verfahren eine größere Schwierigkeit darstellt. Wir evaluieren unsere neue Methode anhand von akademischen und realen Beispielen, sowohl mit nur einem Prozess als auch mit mehreren (MPI-)Prozessen. Die hier neu entwickelte Algebraische Mehrgittermethode ist um ein Vielfaches schneller als klassische Verfahren zur Lösung linearer Gleichungssysteme und erlaubt damit neue, genauere Simulationen mit gitterfreien Methoden.

Danksagung

Danken möchte ich zunächst allen Kolleginnen und Kollegen am Fraunhofer-Institut für Algorithmen und Wissenschaftliches Rechnen (SCAI) und am Fraunhofer-Institut für Techno- und Wirtschaftsmathematik (ITWM), mit denen ich in den letzten Jahren zusammenarbeiten durfte.
Besonders bedanken möchte ich mich bei meinem Doktorvater Prof. Dr. Marc Alexander Schweitzer für die Betreuung und den Freiraum, den er mir zur Bearbeitung dieses Themas und zur Anfertigung dieser Dissertation eingeräumt hat. Ich danke Prof. Dr. Jochen Garcke für die Übernahme des zweiten Gutachtens. Dr. Bram Metsch und Dr. Hans-Joachim Plum haben durch viel Fachwissen und kritische Nachfragen maßgeblich zur Entstehung dieser Dissertation und zur Umsetzung der Ergebnisse in Software beigetragen. Besonderer Dank gilt Dr. Jörg Kuhnert dafür, dass er das Projekt, aus dem diese Dissertation entstanden ist, initiiert und mich mit seiner Expertise zur Finite Pointset Method unterstützt hat. Mein Dank gilt auch Dr. Pratik Suchde für viele E-Mails und Telefonate zur Theorie der Generalisierten Finte Differenzen Methoden sowie der Finite Pointset Method.
Meine Eltern, meine Schwester und meine Großeltern haben mich auf meinem bisherigen Weg stets unterstützt, wofür ich ihnen zutiefst dankbar bin. Celin danke ich für ihr Verständnis und die Motivation, die sie mir in den letzten Monaten gegeben hat.

Bonn, im Juni 2019 Fabian Pascal Nick

Contents

Chapter 1

Introduction

This thesis establishes an efficient and robust Algebraic Multigrid method (AMG) for the solution of the linear systems of equations arising in Generalized Finite Difference Methods (GFDMs). In particular, it focuses on the Finite Pointset Method (FPM), which is a meshfree Lagrangian Generalized Finite Difference Method developed to simulate fluid flow problems with free surfaces or moving geometries. In this process, large sparse linear systems of equations arise. Upon refining the GFDM discretization, solving these systems can quickly become the main bottleneck of the simulation. In classical mesh-based simulation methods, Multigrid methods have proven to be very efficient linear solvers. As we will see, Algebraic Multigrid methods are a natural candidate in the context of GFDMs. However, their application is not straight forward and they need to be tailored to the specifics of the linear systems being solved, which is the main objective of this thesis.

1.1 Reasons for Meshfree Methods

The physical problems we are dealing with in this thesis range from classical CFD problems from aerodynamics to more sophisticated problems like water management for automobiles. Most of these problems are modeled by the incompressible Navier–Stokes equations. Our main interest lies in applications that cannot be solved easily using classical CFD methods, like Finite Volume Methods or Finite Difference Methods (FDMs). A good example is the aforementioned area of water management in the automotive industry. Questions here include the induction of water into the engine when an automotive is crossing a body of water or when exposed to heavy rain. Another example is the opening of a valve. Here the aim of the simulation is to find out how long it takes until a constant fluid flow is established after the valve has started to open. All these examples have in common that they either include moving geometries, free surfaces or both.

Such processes are difficult to simulate using mesh-based methods as the free surfaces or the boundaries at moving geometries would require a frequent adaption of the mesh. In addition to that, in many cases the geometries involved are complicated, which means that an extensive pre-processing phase is required in order to create a mesh in the first place. Oftentimes this process can only partially be automated and requires a lot of human resources and specialized skills. This is despite the fact that there has been substantial work on trying to automate these processes, see for exam-

ple the work of Yerry and Shephard [171] [131] or Baker [8]. Although in some cases and for certain types of geometries such techniques work, many complex geometries cannot be meshed automatically. Another drawback of automatic mesh generation is that different discretization methods like Finite Elements for example put different constraints on the type of mesh needed for a stable discretization. Therefore, automatic meshing tools often only work for one specific type of discretization.

As the name suggest, meshfree methods have their key advantage in that they do not require the explicit generation of a mesh. This frees up human resources in preprocessing and is also advantageous in terms of the algorithmic complexity, as no mesh adaption like in mesh-based methods is required. In some cases simulating a phenomenon using a mesh-based method is not feasible, because the deformation of the domain is too complex. Then, meshfree methods become an important option.
Generally, we can classify two sorts of meshfree methods: Strong form methods, that are based on the strong form formulations of the underlying partial differential equations (PDEs), and weak form methods, that use their weak formulation.
The term *meshfree method* is not well defined though, as there are a lot of methods that can be called *meshfree* but still use some sort of mesh to some extent. Chen et al. [24] give an extensive overview of many of the methods that are commonly referred to as meshfree methods. The work of Melenk, Babuška et al. [97] [4] [5] is focused more towards the weak form methods whereas Huerta et al. [62] focus on strong form methods.

In this thesis, we will focus on a special class of strong form methods [1], the Generalized Finite Difference Methods (GFDMs) which are a generalization of classical Finite Difference Methods. Although the term *Generalized Finite Difference Method* only came up in recent years, the idea has been around a lot longer. Liszka et al. [84] [83] laid the foundation for the method discussed in this thesis in 1980. As a *Generalized* Finite Difference Method we understand methods that use the basic concept of approximating derivatives of functions by finite difference stencils in a setting that does not necessarily need a regular mesh of any kind.

In particular, we will discuss how the Finite Pointset Method (FPM) uses a GFDM to solve the conservation equations for momentum, mass and energy. FPM as the method we are using in this thesis[2] started as a Generalized Smoothed Particle Hydrodynamics method in the dissertation of Kuhnert [75]. It has since been developed further by Tiwari and Kuhnert [151] [152], among others, and extended in numerous ways, for example towards multi-phase flows [153] [35]. It is also well established in a number of industrial applications like automotive water management [68], industrial processing of fluids in extractors [35] and metal cutting [158]. Being a meshfree method, the FPM is very well suited for problems with moving geometries and free surfaces that were mentioned before. Like standard FDMs, GFDMs lead to large

[1] More meshfree strong form methods can be found in, among others, the work of Li and Liu [81] [82].
[2] Note that there is also a Finite Point Method (as opposed to Finite Point**set** Method) that is very similar, but different in some details [107]. There is also a Finite Particle Method [86], which is similar to Smoothed Particle Hydrodynamics (SPH) [44] [91]. Both methods are not discussed here.

sparse linear systems of equations that need to be solved. The properties of those systems however are different from those arising from FDMs: They are non-symmetric, denser and in a Lagrangian method that involves time stepping, their structure changes more radically than in comparable mesh-based schemes using FDMs. Together with the observation that solving linear systems is the main bottleneck in most simulation methods and the lack of an efficient linear solver for such systems, this makes linear solvers an interesting and essential subject for this research.
A key contribution of this thesis is the analysis of the linear systems of equations arising in Generalized Finite Difference Methods, especially in the framework of the FPM. Based on this analysis, a new AMG method is developed that meets the demand of having an efficient linear solver for linear systems arising from GFDMs.

The freedom of not needing to generate or adapt a mesh over time in meshfree methods like the FPM comes at the price of having to manage a *point cloud* that moves with the velocity field in every time step. Although the requirements for this point cloud are not as strict as they would be for a FD mesh or even a FE mesh, it does require some effort to make sure that the point cloud is always well suited to discretize the problem. This is mainly due to the Lagrangian character of the FPM, i.e. the movement of the points. Together with the potentially moving geometry and the parallelization of the method this leads to a number of tasks that need to be carried out in order to organize the point cloud in every time step. These tasks include neighbor determination, adding and merging points and finding distances between points and the boundary. Besides the solution of the linear systems, these are the key bottlenecks when it comes to the performance of the FPM.

The other category of meshfree methods is the weak form methods. One example for such methods is the Partition of Unity Method [6] where a partition of unity $\{\phi_i\}$ is constructed based on *patches* that form an open cover of the computational domain. These functions are then multiplied by local approximation functions ν_i^n, that exist separately on every patch, to form the global approximation space. The benefit of this method comes from the fact that it is possible to have different approximation functions on every patch, allowing to improve the approximation quality where necessary without having to introduce additional patches. Instead, those patches that (partially) include the area of the domain that is to be refined, are *enriched* with additional local approximation functions.
While strong form meshfree methods are more often used in the field of CFD, the weak form meshfree methods have a broader applicability in mechanics applications like crack propagation [113] and composite structure analysis [63]. Schweitzer [125] and Fries and Belytschko [41] wrote survey articles that both focus on weak form meshfree methods. The present thesis focuses on strong form meshfree methods though, as the differences in the linear systems arising from the two different categories are substantial.

1.2 Linear Solvers and Their Importance in Simulation Software

All implicit methods – meshfree or mesh-based – have in common that in the processes of solving the PDEs, the solution of some sort of linear systems is needed. Noting that solving these systems often is the most time-consuming part in the simulation, the importance of finding the most efficient algorithm becomes evident. This is mainly because when refining the discretization, the number of rows in the matrix grows and classical linear solvers have a non-linear complexity. In addition to that, specialized discretization methods like the FPM can introduce other challenges: For example, when dealing with valves, the linear systems arising from the FPM are particularly challenging because of the thin channels in the geometry when the valve is opening and because of the saddle point structure of the linear systems when the fluid is highly viscous. The saddle point structure arises because the FPM uses a special pressure correction approach for highly viscous fluids. Saddle point systems are difficult to solve for classical one-level iterative solvers, meaning that the computational effort to solve these systems is very high compared to the rest of the simulation.

1.2.1 Algebraic Multigrid

In order to allow fast simulations with the FPM, this thesis will examine *Algebraic Multigrid methods*, see for example Ruge and Stüben [117] and Stüben [138], for the solution of the arising linear systems. The first Multigrid methods were so-called *Geometric Multigrid methods*, which have been developed in the 1980's by Brandt [21] and Hackbusch [51]. Multigrid methods have proven to be very efficient iterative solvers for large sparse linear systems of equations that stem from mesh-based discretizations of elliptic PDEs. They can be applied to systems with even millions of equations, because they *scale* linearly with the system size. Additionally, Algebraic Multigrid methods are *robust* with respect to the specific problem they are being applied to, if they are constructed correctly.

The major difference between Geometric Multigrid methods and Algebraic Multigrid methods is the way in which the *coarse grids* are created. Coarse grids are used to reduce the low frequency error components that are difficult to reduce by classical relaxation schemes like Gauss-Seidel methods on the fine level alone. In Geometric Multigrid methods, geometric analogues of the fine grid are used as coarse grids, i.e. grids with smaller resolutions. They can only be applied when a suitable coarse grid is available from the discretization. This may require the explicit generation of multiple grids of different resolutions, which is a major drawback of Geometric Multigrid methods, especially for simulations involving complex geometries.
Algebraic Multigrid methods do not need any information about the underlying geometry, instead they use the matrix representing the linear system to construct a hierarchy of "grids", which are sometimes called "levels"[3]. There is a variety of different approaches on how to construct the coarse levels, which can be clustered into two main

[3]The term "Algebraic Multi*grid* methods" has historical reasons.

ideas: The Ruge-Stüben-coarsening by Ruge and Stüben [117] [138] and the aggregation techniques primarily influenced by Vanek, Brezina and Mandel [163] [164] [162]. The creation of the levels and the necessary transfer operators in AMG is called *setup phase* whereas the iterations are done in the *solution phase*. This distinction is important as the setup phase can be re-used in simulations involving time-stepping, as this thesis will demonstrate.

Algebraic Multigrid theory usually assumes that the matrix representing the linear system is sparse, symmetric, positive definite and has the M-matrix property, although there are various efforts to extend this theory to other cases. The matrices arising from the FPM or GFDMs in general do not have some of these properties. Therefore, the aim of this thesis is to better understand the applicability of AMG techniques to those matrices, despite the lack of a theoretical framework, and to develop a new AMG method that is as efficient and robust as possible on those matrices. We construct a Geometric Multigrid method in order to motivate pursuing the idea of using AMG techniques. Since the algebraic method is much more robust, we tune this method using the knowledge we gained from analyzing the matrices and apply it to real world industrial applications.

One particular property of the FPM that poses a challenge to AMG is that the linear systems can potentially decompose into multiple independent subsystems that can be singular in some cases. Due to the Lagrangian character of the method this is hard to avoid from the discretization side for two reasons: First, the computational domain might physically decompose into multiple domains, for example when droplets of fluid are formed due to external forces. Secondly, thin features in the domain, like in the opening valve mentioned before, can cause the point cloud to decompose into multiple *components* because the discretization size is not small enough to properly resolve these features. In some cases, these components lead to singular subsystems in the linear system. Consequently, the linear solver needs to take this property into account. For our new robust AMG method, we therefore need to find those subsystems and treat them separately from the rest of the system. To this end, we develop a parallel algorithm for finding graph components based on local diffusion that has linear complexity, which means that it does not hamper the linear complexity of the AMG algorithm.

Another important aspect of the AMG method we develop in this thesis is to re-use the setup phase as often as possible. When re-using the setup phase, we do not build a new AMG hierarchy based on the fine level matrix, but re-use the hierarchy from a previous linear system. This way we save most of the computational effort in the setup phase, but the convergence for the current linear system may be slightly worse because the AMG hierarchy was built for a different linear system. This is of particular interest in cases where the number of iterations needed in the solution phase is low. We show how this is the case in some of the systems in the FPM and determine strategies of re-using the setup where appropriate. Re-using the AMG setup in the FPM is more involved than in mesh-based discretizations, because the FPM may insert or delete points in the point cloud between two time steps. Therefore, the size of the linear sys-

tems may change as well. We discuss different approaches to overcome this problem. The strategy of our choice has an impact on the FPM algorithm itself: Namely, parts of the FPM point cloud organization can be re-used, similar to the AMG setup. With this re-use of the point cloud, we can then also re-use the AMG hierarchy. The re-use of the FPM point cloud originates in the work that went into this thesis.

The new AMG method developed in this thesis proves to be efficient in real world cases, as we will demonstrate using some examples. It gives significant speed-ups over classical one-level linear solvers, like the BiCGStab2 method that we will frequently compare it with.

1.3 Outline of this Thesis

The rest of this thesis is organized as follows: In order to understand the challenges that the linear systems arising in GFDMs pose to linear solvers and Algebraic Multigrid methods in particular, we introduce a particular GFDM and how it is used in the FPM in Chapter 2. In this chapter, we also analyze the properties of the arising linear systems in the FPM. It is worth noting that in the FPM there are three types of linear systems that need to be solved, so many of the arguments in this thesis are divided into three cases, one for every type of linear system. Chapter 3 reviews state of the art linear solvers and their applicability to the linear systems in this thesis. This chapter also develops a multigrid method based on a geometric idea in order to motivate the construction of an Algebraic Multigrid method. The latter is the basis for our new method in Chapter 4, in which we describe the AMG techniques that will be used for the experiments in Chapter 6. Before going into the experiments though, we make comments on the parallelization of the method in Chapter 5. An important contribution in this chapter is the analysis of the parallel subsystem detection that we implemented in order to extract singular subsystems from the linear system. The chapter also demonstrates why those subsystems occur. Finally, Chapter 7 will give a summary of the thesis and an outlook as to what further questions this thesis rises and what could be future directions of research concerning the efficiency of meshfree GFDMs and their linear solvers.

Chapter 2

Linear Systems Generated by Meshfree Methods

In this chapter we look at various forms of (Generalized) Finite Difference discretizations and the linear systems of equations they impose. It is important to understand the properties of these linear systems, because this will allow us to create an efficient and robust linear solver later. Without an efficient linear solver, most industrial size applications take too long to run within a production environment.

We start by recalling some properties of standard Finite Difference Methods in 1D and then show some non-regular discretizations in 1D and 2D, pointing out the differences to the standard case. For the non-regular case, we look at a non-equidistant discretization of the Poisson problem in 1D using a method by Fornberg [40] in order to generate Finite Difference stencils. Secondly, we will look at an equidistant 2D discretization of the Poisson problem on the unit square using different (both second order) stencils at different points, showing that this leads to a non-symmetric matrix while still discretizing a self-adjoint operator.

After that, we move on to the most general case of GFDMs on point clouds, which is the method used to discretize the computational domain in the FPM. The FPM is a pressure-correction method mainly used for solving incompressible Navier–Stokes equations. In the FPM, the stencils are created by solving a least squares problem on the points within a circular neighborhood of the central point.

We will then analyze the properties of the linear systems arising from GFDMs as they are used in the FPM based on a simple Poisson problem first, because the pressure systems in the FPM are similar to a Poisson system. Afterwards, we will discuss the properties of the two other types of linear systems arising in the FPM: The velocity system and the coupled velocity-pressure system which arises only in a specialized version of the FPM for flows with low Reynolds numbers. This analysis is necessary because the size of these linear systems in industrial applications makes the application of standard linear solvers unfeasible, as we will see in the next chapter. Therefore, more sophisticated linear solvers, like multigrid solvers, are required. These solvers must be carefully tailored to the specific discretization though in order to be as robust and efficient as possible, which will be the overarching objective of all the following chapters.

2.1 Overview of Meshfree Methods

Meshfree Methods in the context of this work are methods that can be used to numerically solve PDEs. In contrast to classical mesh-based methods like Finite Difference Methods or Finite Element Methods, they do not need an explicit mesh to be generated. Instead, they operate on a *cloud* of points or particles or they use a set of *patches* in order to discretize the PDE in space. Note that many meshfree methods use an implicit "mesh" based on some sort of neighborhood relationships. For example, the Finite Pointset Method [35] [68] which will be in the focus of this work, establishes new neighborhood relationships between points in every time step, which then serve as means to discretize the differential operators. Similarly, Smoothed Particle Hydrodynamics (SPH) [44] [91] uses a compactly supported smoothing kernel that smooths the physical properties between the discrete particles, implying a neighborhood relationship through the size of the compact support. While SPH was first used in astrophysics, it is nowadays a popular method to model fluids in computer graphics [64] [146] and is also used in the classical CFD regime, for example to model the free surface flow in a Pelton turbine [93]. An example for a meshfree method using a background mesh is the Material Point Method, which is a hybrid Lagrangian-Eulerian method, in which a fixed background mesh is used to determine the spatial gradients [145].
Nevertheless, all these methods are commonly agreed to be *meshfree* in nature. Chen et al. [24] observe that meshfree methods "share a common feature [...]: the approximation of unknowns in the PDE is constructed based on scattered points without mesh connectivity". Liu [85] even dedicates a whole section to the definition of *meshfree methods*.

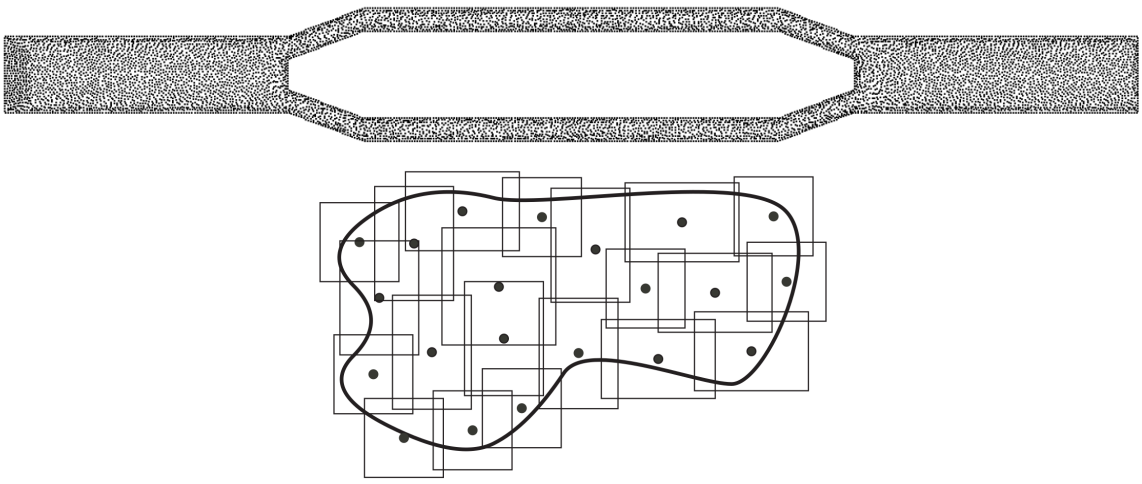


Figure 2.1: Point cloud for a bifurcated tube [141] and a covering of a domain by patches [123].

The class of meshfree methods is very large and can be divided into two main classes: *Strong-form* meshfree methods and *weak-form* Meshfree methods. Chen et al. provide an overview and classification of meshfree methods with references to further literature on each method in [24]. There are some weak-form meshfree methods that allow

to improve the solution accuracy through adding special enrichment functions to the trial space, like in GFEM [5]. These can be especially helpful when dealing with crack propagation problems, see for example [124].

The focus of this thesis is on strong-form meshfree methods like Generalized Finite Difference Methods (GFDMs). The Finite Pointset Method (FPM) [35] [68] uses GFDMs in order to discretize the differential operators arising in the Navier–Stokes equations, for example. GFDMs started to emerge in the early seventies [110], although the term *meshfree method* only came up later [83].

2.1.1 Advantages

In mesh-based methods, meshes (or "grids") need to be generated either algorithmically or by an engineer. Either way, this task adds another layer of complexity to the simulation. Keep in mind that in most cases the mesh is not what engineers are interested in, i.e. it is not part of the actual result of the simulation, but it is a vehicle that is necessary to obtain the desired result. Also, creating a mesh for a simulation can often be the most time-consuming and manual part of the whole simulation process.

The task of creating meshes becomes particularly time-consuming when a simulation involves moving geometries, e.g. when simulating air bag inflation [30], or fluids with free surfaces like we find in sloshing simulations [130]. Frequent re-meshing or mesh-adoption techniques [111] then becomes mandatory with mesh-based methods. Another process where frequent re-meshing would be necessary when using a mesh-based method is the propagation of cracks in a (possibly composite) material.

Due to their nature, most meshfree methods can handle those situations naturally. Many of them can also handle multiphase flows, where the boundary between both fluids moves over time. Examples include the FPM as described in [153] and later in this chapter, or the Moving Particle Semi-implicit Method (MPS) [129].

Especially for the weak-form meshfree methods like the Generalized Finite Element Method (GFEM), the Extended Finite Element Method (XFEM) and the (Particle) Partition of Unity Method ((P)PUM), one key advantage is that they allow an easy, local p-refinement by adding additional local approximation functions on specific *patches*, i.e. in certain areas of interest. This allows them to incorporate problem specific knowledge into their discretization. If it is known – either from physical understanding or from previous, more localized simulations – that a certain type of function gives good approximations in some area, for example around a crack, then this function can be used as a local approximation function in that area without affecting the rest of the discretization. For this reason, weak-form meshfree methods can often work with a much lower number of degrees of freedom than other methods, because their power lies in the quality of the (local) approximation functions rather than in the resolution of the discretization.

2.2 Generalized Finite Difference Methods

In this section we want to introduce *Generalized* Finite Difference Methods, especially the Least Squares method that will be used in the FPM. For every method we also point out the differences to classical Finite Difference Methods.

2.2.1 Notation and First Observations for Classical Mesh-Based Finite Difference Methods

Consider Poisson's equation with Dirichlet boundary conditions

$$\begin{cases} -\Delta u &= f \text{ on } \Omega \\ u &= g \text{ on } \partial\Omega \end{cases}, \tag{2.1}$$

for $\Omega = (0,1)$ and $g \equiv 0$. For a given $0 < h < 1$ the points $x_i = hi,\ i = 0, \dots, N,\ N = 1/h$ form a structured grid $\Omega_h = \{x_i\}$. The classical Finite Difference Method uses this structured grid to approximate the derivatives of a function $u : \Omega \mapsto \mathbb{R}$ by finite differences. It is a well-known result[1] that

$$u''(x_i) = \frac{u(x_{i-1}) - 2u(x_i) + u(x_{i+1})}{h^2} + \mathcal{O}(h^2)$$

and we therefore write

$$u''(x_i) \approx \frac{1}{h^2}\left[u(x_{i-1}) - 2u(x_i) + u(x_{i+1})\right] =: \underbrace{[1 \quad -2 \quad 1]_h}_{\mathfrak{s}_i} u(x_i). \tag{2.2}$$

The last part of equation (2.2) introduces the *stencil* notation. With $\mathfrak{s}_i$ we denote a set of coefficients $\{c_j\}_i$ that are associated with the neighbors of point x_i. We will use $\mathfrak{s}_{ij}$ to refer to the coefficient c_j in the stencil $\mathfrak{s}_i$.

Together with the boundary conditions, the linear system arising from this discretization is

$$\frac{1}{h^2}\begin{pmatrix} 1 & & & & & \\ -1 & 2 & -1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & -1 & 2 & -1 & \\ & & & & 1 \end{pmatrix}\begin{pmatrix} u_h(0) \\ u_h(x_1) \\ \vdots \\ u_h(x_n) \\ u_h(1) \end{pmatrix} = \begin{pmatrix} 0 \\ f_h(x_1) \\ \vdots \\ f_h(x_n) \\ 0 \end{pmatrix}, \tag{2.3}$$

from which we can eliminate the rows corresponding to boundary conditions to obtain

$$\underbrace{\frac{1}{h^2}\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}}_{=:A_h}\begin{pmatrix} u_h(x_1) \\ u_h(x_2) \\ \vdots \\ u_h(x_{n-1}) \\ u_h(x_n) \end{pmatrix} = \begin{pmatrix} f_h(x_1) \\ f_h(x_2) \\ \vdots \\ f_h(x_{n-1}) \\ f_h(x_n) \end{pmatrix}. \tag{2.4}$$

[1] For a derivation see for example [80], p. 8.

Note that this elimination is possible because of the simple incorporation of the boundary conditions in the discretization. The matrix row corresponding to the boundary point is coupled with only one interior point in this case which makes it easy to eliminate the row from the matrix.
The subscript h indicates that we are dealing with discrete analogues of the functions u and f, i.e. $u_h, f_h : \Omega_h \to \mathbb{R}$. Similarly, A_h denotes the matrix that is associated with the grid of size h.

Before we can make a remark about the properties of the matrix A_h that are of interest in the context of this thesis, we define a less known property:

Definition 2.1. The matrix A is called *essentially diagonally dominant* if it is weakly diagonally dominant and every row is coupled directly or indirectly to at least one row that is diagonally dominant. In Section 5.2 we will define more precisely what is meant by *coupled*.

Remark 2.1. Note that A_h has some properties that we will be dealing with throughout this thesis:

- *Sparsity:* Because the stencil at every point only includes the point itself and two or one neighboring points in the interior and at the boundary respectively, every row in the matrix contains only two to three non-zero entries. In 2D and 3D the minimum number of neighbors required for a second order discretization of the Laplace operator Δ is four and six respectively, which still leads to sparse linear systems[2].

- *Symmetry:* The stencil
$$[1 \quad -2 \quad 1]_h \tag{2.5}$$
is symmetric, meaning that the left neighbor and the right neighbor have the same coefficient $c_j = -1$. In a standard Finite Difference Method we are also applying the same stencil at every interior point. As a result, after eliminating the boundary conditions or taking other precautions to preserve symmetry at the boundary, the matrix A_h is also symmetric.

- *M-matrix:* A_h is an L-matrix[3] as its diagonal entries are all positive and its off-diagonal entries are all negative. It is also essentially diagonally dominant and is therefore an M-matrix, see [127] and [52].

- *Positive definite:* Since A_h is essentially diagonally dominant, we know by using Gershgorin discs [43] that all eigenvalues are positive or zero. If the problem is well posed, then there exists a unique solution and therefore A_h is regular, if a consistent Finite Difference stencil has been used. In this case, 0 cannot be an eigenvalue of A_h and thus A_h is positive definite.

[2] A linear system and the associated matrix are considered *sparse* if the number of non-zero entries per row is much smaller than the number of zero entries. See [45] for some remarks on sparse matrices.

[3] An L-matrix has positive diagonal entries and negative or zero off-diagonal entries.

- *Banded:* The matrix is a band matrix with bandwidth three. Note that this is the case only because of the numbering of the grid points we chose. A different numbering of the grid points will generally result in a different structure of A_h.

In this section we have introduced some notation related to Finite Difference Methods on structured grids. In meshfree methods we generally deal with unstructured discretizations. The next section will therefore summarize some observations for Finite Difference Methods on unstructured grids in one dimension.

2.2.2 The Unstructured Case in 1D: Fornberg's Algorithm

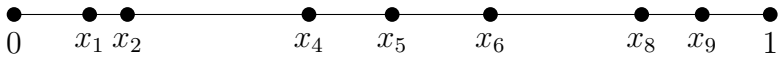


Figure 2.2: A non-uniformly spaced grid in 1D.

In his 1988 paper [40], Fornberg developed an algorithm to find Finite Difference formulas for arbitrarily spaced grids in one dimension. A MATLAB implementation of his algorithm can be found in appendix A. The major difference between Fornberg's work and classical Finite Difference Methods is the arbitrary spacing between the grid points. This is a property that we will also encounter when looking at GFDMs in higher dimensions in Section 2.2.4. For the Poisson problem (2.1), consider a discretization with a non-uniformly spaced grid as shown in Figure 2.2. The algorithm in appendix A can compute the matrix corresponding to this unstructured grid after elimination of the boundary conditions:

$$F := \begin{pmatrix} 4 & -2.6667 & & & & & \\ -1.3793 & 1.6667 & -0.28736 & & & & \\ & -0.2381 & 0.75758 & -0.51948 & & & \\ & & -0.75758 & 1.3986 & -0.6413 & & \\ & & & -0.4662 & 0.76923 & -0.333 & \\ & & & & -0.35714 & 1.25 & -0.89286 \\ & & & & & -1.476 & 2.7778 \end{pmatrix}$$

In this example, we have used two neighboring grid points at every grid point to construct the stencil. Fornberg's algorithm also allows to use more than only two neighbors.

Remark 2.2. This discretization will in general only be of first order (cf. [40] Section 2), while the discretization in Section 2.2.1 is of second order. Fornberg's algorithm in [40] could also generate second order stencils if we used four instead of two neighbors at every point.

Remark 2.3. Comparing F to A_h from Section 2.2.1 we can make the following observations:

- *Sparsity:* We decided to use only two neighbors in this example, which gives us the same sparsity as in Section 2.2.1. However, if we wanted to ensure a

second order accurate discretization, we would have had to use four neighbors, which would have made the matrix less sparse. In two and three dimensions considerations regarding the necessary points to ensure a certain approximation order become much more sophisticated, see for example [126].

- *Symmetry:* The main difference between A_h in Section 2.2.1 and F is that F is not a symmetric matrix. This is because we constructed each stencil independently from all the other stencils and based only on local neighborhoods. Since we used an arbitrary distribution of those grid points, the neighborhoods of two distinct points are in general different from each other.
  This is a common observation for non-uniform grids, despite some symmetry-preserving discretizations for special cases, see for example [155] or [166].

- *M-matrix:* Note that the same arguments as in Section 2.2.1 can be applied here, so F is an M-matrix.
  Note however that for higher order discretizations, Fornberg's formulas yield mixed-sign stencils, which means that those matrices are no L-matrices any more because they can have positive off-diagonal coefficients. Therefore, they cannot be M-matrices.

- *Positive definite:* As long as Fornberg's method does not yield mixed sign stencils, F is positive definite by the same arguments that we used in Remark 2.1. However, if positive off-diagonals occur, while the sum of all stencil coefficients for interior points is still 0, the Gershgorin discs do not exclude negative eigenvalues any more.

- *Banded:* Although we are now dealing with an arbitrarily spaced grid, the numbering of the points from left to right is still a natural choice and leads to a matrix with bandwidth three, like in Section 2.2.1. As we will see later, the situation is different when looking at the more general case of point clouds in dimensions two or three.

We have now seen how an unstructured grid impacts the linear system that has to be solved. In the next section we will see that it is also possible to obtain non-symmetric matrices on a regular grid.

2.2.3 A Non-Symmetric 2D-Discretization on a Regular Grid

As mentioned in Remark 2.3 one major difference between standard Finite Difference matrices on a regular grid and the Finite Difference formulas obtained by Fornberg's algorithm on non-regular grids is that the latter yields non-symmetric matrices. The reason was that we are not using the same stencil at every grid point, because the neighborhood of every two distinct grid points is different. Something similar happens if we decide to use different stencils at different grid points, even on a regular grid. Consider the 2D Poisson equation with Dirichlet boundary conditions

$$\begin{cases} -\Delta u &= f \text{ on } \Omega \\ u &= g \text{ on } \partial\Omega \end{cases}, \tag{2.6}$$

with $\Omega = (0,1)^2$ and $g \equiv 0$. Like in Section 2.2.1 a structured grid of discretization size h is defined by $\Omega_h = \{x_{ij}\}$ with $x_{ij} = (ih, jh)$ where $i, j = 0, \ldots, n,\ n = 1/h$.

Consider the following two stencils from [80] discretizing the Laplace operator in two dimensions that are of second order:

$$\mathfrak{s} = \begin{bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix}_h \quad \text{and} \quad \mathfrak{t} = \begin{bmatrix} -1 & -4 & -1 \\ -4 & 20 & -4 \\ -1 & -4 & -1 \end{bmatrix}_h . \tag{2.7}$$

Remark 2.4. LeVeque ([80], pp. 64) notes that using the stencil $\mathfrak{t}$ can be beneficial in some situations. On the other hand, the additional entries in the matrix will lead to a denser matrix (cf. Remark 2.5) and therefore more floating point operations when computing matrix-vector products or solving linear systems.

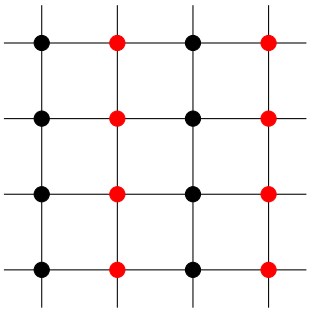


Figure 2.3: Subgrid splitting of a regular 2D grid.

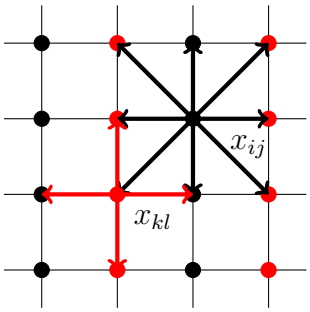


Figure 2.4: A regular 2D grid with two different discretization stencils for the Laplace operator: x_{ij} connects to x_{kl} but x_{kl} does not connect to x_{ij}.

Splitting the grid Ω_h into two disjoint subgrids $\Omega_h^e = \{x_{ij}\}$ with i even and $\Omega_h^o = \{x_{ij}\}$ with i odd we obtain a line-wise splitting of the grid, see Figure 2.3. We can then apply the stencil $\mathfrak{s}$ to all points in Ω_h^e and the stencil $\mathfrak{t}$ to all points in Ω_h^o. The resulting matrix $A_h = a_{ij}$ will also be non-symmetric, because the points in Ω_h^o where $\mathfrak{t}$ is applied have four diagonal connections to points in Ω_h^e, which do not have any connections in the diagonal directions, see Figure 2.4. This means that for $x_{ij} \in \Omega_h^o$ corresponding to the p-th row in A_h and $x_{kl} \in \Omega_h^e$ corresponding to the q-th row in A_h we will have

$$a_{pq} = -1 \quad \text{and} \quad a_{qp} = 0. \tag{2.8}$$

Definition 2.2. A matrix $M = (m_{ij})$ is called *structurally symmetric* if and only if

$$m_{ij} \neq 0 \Leftrightarrow m_{ji} \neq 0$$

In this case here, A_h is not structurally symmetric and hence non-symmetric.

Remark 2.5. Comparing A_h to the matrix arising from a discretization with solely the stencil $\mathfrak{s}$ [4] we can make the following observations:

- *Sparsity:* With this discretization, the overall sparsity of the matrix decreases, because the stencil $\mathfrak{t}$ is denser than the standard stencil for the 2D Laplace operator $\mathfrak{s}$.
- *Symmetry:* As already mentioned before, A_h is non-symmetric and also not structurally symmetric.
- *M-matrix:* Again, we can apply the arguments from [127] and [52]: Due to the nature of the stencils $\mathfrak{s}$ and $\mathfrak{t}$, the matrix is an L-matrix. Together with Dirichlet boundary conditions, it is also essentially diagonally dominant and therefore an M-matrix.
- *Positive definite:* Because both stencils used here have only negative off-diagonal coefficients, the same arguments as in Remark 2.1 apply, making A_h a positive definite matrix.
- *Banded:* Due to diagonal couplings in the stencil $\mathfrak{s}$, A_h does not have a small bandwidth like the matrices in Section 2.2.1 and Section 2.2.2. Numbering the grid points in the natural row-wise order will lead to a matrix A_h with bandwidth $2n + 2$. However, this matrix can still be written as a block tridiagonal matrix, see [80].

Remark 2.6. Remark 2.5 holds for any disjoint splitting $\Omega_h = \Omega_h^1 \cup \Omega_h^2$, except that for certain special cases, the symmetry of the matrix is restored. For example, consider $\Omega_h^1 = \{x_{ij}\}$ with $i + j$ even and $\Omega_h^2 = \{x_{ij}\}$ with $i + j$ odd, resulting in $\mathfrak{s}$ and $\mathfrak{t}$ being applied in a checkerboard manner.

We have now seen two examples for non-standard discretizations with Finite Difference Methods. The next section will deal with the more general case of Generalized Finite Difference Methods on point clouds, which then leads to the description of the Finite Pointset Method in Section 2.3.

2.2.4 Generalized Finite Difference Methods on Point Clouds

As we are mostly interested in simulations in two or three dimensions, and at the same time want to drop the idea of having a mesh and therefore the need to generate one, we need to find means to define Finite Difference Methods on unstructured sets of points $\mathcal{P} = \left\{x_i \in \Omega \subset \mathbb{R}^d\right\}, i = 1, \ldots, N$, which we will call *point clouds*. A point cloud $\mathcal{P}$ does not per se include any neighborhood relationships between the points.

[4] See [80] for a detailed discussion.

In this thesis, we will only be dealing with the cases $d = 1, 2, 3$. In higher dimensions, which can occur in Big Data or Machine Learning applications, the cost of finding neighbors can increase significantly, affecting the overall efficiency of the method. In these areas where d becomes large, efficiently finding distances and nearest neighbors is a research topic in itself, see for example [3].

Uniformness of a Point Cloud

In the case of non-uniform grids or point clouds, it is not as easy to define a discretization size h as it is in the uniform case.

Definition 2.3. There are four main quantities that we will use in order to quantify the discretization size of a point cloud $\mathcal{P}$:

1. $\mathcal{N}(\mathcal{P}) = |\mathcal{P}|$ is the number of points in the point cloud.
2. $h_{\max}(\mathcal{P})$ is the maximum distance between any point and its nearest neighbor.
3. $h_{\min}(\mathcal{P})$ is the minimum distance between any two points.
4. $\mathcal{U}(\mathcal{P}) = h_{\max}(\mathcal{P}) / h_{\min}(\mathcal{P})$ measures the *uniformness* of the point cloud.

In cases where it is obvious which point cloud $\mathcal{P}$ we are referring to, we will omit the argument $\mathcal{P}$ and only write $\mathcal{N}$, $h_{\min}$, $h_{\max}$ and $\mathcal{U}$ respectively.

$\mathcal{N}(\mathcal{P})$ dictates the size of the resulting linear system, while $h_{\min}$ has a major impact on its condition number, see Section 2.4. On the other hand, $h_{\max}(\mathcal{P})$ determines the approximation quality of the discretization. It is therefore desirable to keep $\mathcal{U}(\mathcal{P})$ close to 1, i.e. making the point cloud as uniform as possible. To achieve this, a proper point cloud management needs to be employed, cf. Section 2.3.5 and Seibold [126]. It is possible though to refine point clouds locally, which makes these considerations more complicated, see the work of Hagmann [55].

Least Squares Methods

Given a point cloud $\mathcal{P}$ discretizing a domain Ω and the 2D Poisson equation (2.6), we are interested in stencils $\mathfrak{s}_i^\Delta$ such that

$$\Delta u \approx \sum_{j \in N_i} c_{ij}^\Delta u_j. \tag{2.9}$$

where N_i is a set of indices defining a neighborhood of point x_i. To this end, Least Squares methods can be employed, which originate in the idea of *scattered data interpolation*. From this method, Liszka et al. [83][5] and Liszka and Orkisz [84], among others, derived a method for the application of the Least Squares idea to the solution of PDEs.

[5] Geometric criteria have to be satisfied as noted in [83] and the references therein.

Remark 2.7. The same idea can also be applied to differential operators other than the Laplacian. To account for that, we use the superscript $*$ to indicate a desired differential operator.

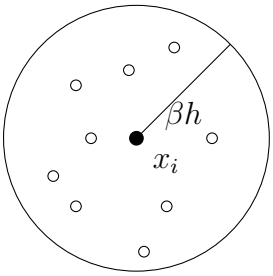


Figure 2.5: Neighborhood of a point x_i.

Kuhnert [75] and Tiwari and Kuhnert [35] [68] follow a Moving Least Squares idea [78]. Their approach has been further developed in [76]. The neighborhood N_i of point x_i in equation (2.9) is chosen to consist of all points x_j within a ball of radius βh around x_i, see Figure 2.5. When using adaptive refinement of the point cloud, $h(x_i)$ can vary from one point to another, see the following definition. This makes the neighboring relationship non-symmetric. For reasons of computational efficiency, the size of the neighborhood is often restricted to a fixed number of neighbors per point. This introduces yet another cause for non-symmetry as the decision as to which points are kept in the neighborhood is made individually for each point without accounting for possible non-symmetric effects.

Definition 2.4. The parameter h is called *smoothing length* and is similar to the *discretization size* h in standard Finite Difference Methods on uniform grids. It is the radius of the support of the weighting function w we will introduce later. Like in Finite Difference or Finite Element Methods, the smoothing length can also be varied locally, cf. Section 2.3.5. We then write $h(x_i)$, indicating the radius of the support of the weighting function at the point x_i. The parameter β is used to control the number of neighboring points independently of the size of the support. Although all values $\beta \in [0, 1]$ are admissible, a standard choice is $\beta \in [0.75, 1]$ [141]. For very small values of β, a consistent discretization of the desired order may not be possible, because the number of neighbors at some points, and also their relative position to the central point, may not be sufficient. For reasons of computational efficiency, we will artificially reduce the number of neighbors for each point to 20 in 2D and 40 in 3D, while using $\beta = 1$. Therefore, we will omit the parameter in the following and thus the *interaction radius* is then equal to the smoothing length h.

Requiring that the coefficients c_{ij}^* are chosen so that equation (2.9) is satisfied exactly for monomials u of a certain order[6] leads to M conditions that can be written as a linear system of the form

$$K_i^T \mathfrak{s}_i^* = b^*, \tag{2.10}$$

[6]In the FPM, order 2 is a standard choice, but other choices are possible [76].

where

$$K_i^T = \left((k_i^m)^T\right)_{1\le m\le M}, \quad k_i^m \in \mathbb{R}^{|N_i|} \tag{2.11}$$

$$\mathfrak{s}_i^* = \left(c_{ij}^*\right)_{1\le j\le |N_i|} \tag{2.12}$$

$$b^* = (b_m^*)_{1\le m\le M}\,. \tag{2.13}$$

Remark 2.8. An equivalent result can be obtained using Taylor expansion, see [141] Appendix A.3.

Example 2.1. For the Laplacian in 2D, the numerical operator applied to the constant function $u \equiv 1$ as well as the three linear functions $u = x, u = y, u = xy$ shall deliver 0 and for the quadratic functions $u = x^2, u = y^2$ it shall deliver 2. Therefore we have

$$\begin{pmatrix} 1 & \cdots & 1 \\ (x_1 - x_0) & \cdots & \left(x_{|N_i|} - x_0\right) \\ (y_1 - y_0) & \cdots & \left(y_{|N_i|} - y_0\right) \\ (x_1 - x_0)(y_1 - y_0) & \cdots & \left(x_{|N_i|} - x_0\right)\left(y_{|N_i|} - y_0\right) \\ (x_1 - x_0)^2 & \cdots & \left(x_{|N_i|} - x_0\right)^2 \\ (y_1 - y_0)^2 & \cdots & \left(y_{|N_i|} - y_0\right)^2 \end{pmatrix} \mathfrak{s}_i^{\Delta} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 2 \\ 2 \end{pmatrix}, \tag{2.14}$$

where the central point has coordinates (x_0, y_0).

Geometric conditions on the positions of the points $x_j \in N_i$ would be necessary in order to guarantee a unique solution to the linear system (2.10) [83][110]. As checking and satisfying those conditions would become rather involved, especially if the point cloud $\mathcal{P}$ is changing between time steps, see Section 2.3, it is more practical to choose the neighborhood sufficiently large so that the system becomes underdetermined, therefore

$$|N_i| > M. \tag{2.15}$$

This *point cloud management*, which also affects the quantities defined in Definition 2.3, is discussed in more detail in [127], [141] and Section 2.3.5.

Remark 2.9. Point cloud management is also an issue in other meshfree methods, see for example [65]. In the Finite Pointset Method, which is introduced in Section 2.3, there is no mass associated with the points though. This feature allows to easily delete, merge or insert points in the point cloud management phase.

The (underdetermined) system (2.10) is then solved in a least squares sense, whereas an additional radial weighting function

$$w_i(r(x_i, x_j)) = \begin{cases} w(r) \ge 0, & \text{if } r(x_i, x_j) < 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.16}$$

is introduced. A standard choice is to use

$$w(r) = \begin{cases} \exp(-4r^2) - \exp(-4), & \text{if } r < 1 \\ 0, & \text{otherwise} \end{cases}. \tag{2.17}$$

Because values at points that are closer to the central point x_i have a greater impact on the derivative at x_i, the weighting function should be a non-increasing function [83]. But apart from that, choices other than the function (2.17) are possible as well.

Furthermore, the distance function being used is

$$r(x_i, x_j) = 2\frac{\|x_i - x_j\|}{h(x_i) + h(x_j)}. \tag{2.18}$$

Because of the definition of the weighting function in (2.16), this links the size of the support of w_i to the local smoothing length $h(x_i)$.

With

$$W_i = \begin{pmatrix} w_i(r(x_i, x_{J_1})) & & \\ & \ddots & \\ & & w_i(r(x_i, x_{J_{|N_i|}})) \end{pmatrix}, \tag{2.19}$$

$J_k \in N_i,\ k = 1, \ldots, |N_i|$, we can formulate the conditions for $\mathfrak{s}_i$ in a least squares sense:

$$\frac{1}{2}\left(\mathfrak{s}_i^*\right)^T W_i^{-2} \mathfrak{s}_i^* \to \min \tag{2.20}$$

under the constraints (2.10). This minimization problem can be solved using Lagrangian Multipliers [76] by solving the linear system

$$K_i^T W_i^2 K_i \lambda_i = -b^* \tag{2.21}$$

yielding a solution for $\mathfrak{s}_i^*$:

$$\mathfrak{s}_i^* = -W_i^2 K_i \lambda_i = \left(W_i^2 K_i\right)\left(K_i^T W_i^2 K_i\right)^{-1} b^* \tag{2.22}$$

Remark 2.10. Some remarks on the computational effort involved in solving the least squares problem:

- The weighting function w_i needs to be evaluated for every neighboring point. Therefore, the evaluation should be implemented as efficiently as possible.
- W_i is a diagonal matrix. Its multiplication with K_i comes down to multiplying each row of K_i with a constant.
- The main computational effort therefore lies in the evaluation of the two dense matrix-matrix products

  $$K_i^T \cdot \left(W_i^2 K_i\right) \quad \text{and} \quad \left(W_i^2 K_i\right) \cdot \left(K_i^T W_i^2 K_i\right)^{-1} \tag{2.23}$$

  as well as in inverting the term[7]

  $$\left(K_i^T W_i^2 K_i\right). \tag{2.24}$$

  Since $K_i \in \mathbb{R}^{|N_i| \times M}$, $W_i \in \mathbb{R}^{|N_i| \times |N_i|}$ and $M, |N_i| \in \mathcal{O}(1)$, the effort per point is independent of the number of overall points in the point cloud $\mathcal{N}$.

[7] which is usually done by computing a QR-decomposition, see [141].

- Equation (2.24) is an $M \times M$ system where M is small, usually $M < 20$. A direct solver can be used to efficiently solve

$$\left(K_i^T W_i^2 K_i\right) y = b^* \tag{2.25}$$

  in the process of solving the equation (2.22).

- All these computations are point-local and can be done in parallel.

Remark 2.11. What is described here is a *Weighted Least Squares method* applied at each point [107]. There are also other Least Squares methods that are relevant in the context of meshfree methods, see for example [24]. Regarding the matrix properties and the consequences for Multigrid methods, these methods are similar.

Remark 2.12. Remarks 2.1, 2.3 and 2.5 summarized the properties of the matrices arising in the discretization methods discussed in the respective sections. The properties of the matrices arising in GFDMs will be discussed extensively in Section 2.4. Figure 2.9 gives a quick summary of the findings in that section.

2.3 The Finite Pointset Method (FPM)

In this thesis we will use the Finite Pointset Method [76] [35] [68] as an example for a method using Generalized Finite Difference Methods in order to solve the incompressible Navier–Stokes equations. This section will outline the main features of the FPM with a special focus on those features that are relevant to the application of Multigrid methods on the resulting linear systems.
Section 2.3.1 will outline the physical problems that the FPM can solve. The next section will then introduce the continuous forms of the conservation equations that the FPM preserves. In Section 2.3.3 we will develop the discrete versions of those conservation equations and show how we can use them together with two linearization approaches in order to solve the incompressible Navier–Stokes equations. We will comment on the differences of the two approaches in Section 2.3.4, showing that the Reynolds number is a good indicator to decide which approach to use for a specific application. We will have a look at how to manage the underlying point cloud for the FPM discretization in Section 2.3.5. This is necessary as the point cloud management has impact on the way that we can use Multigrid techniques in order to solve the arising linear systems. In many meshfree methods, the boundary conditions require special (in some cases non-trivial) treatment, which we will discuss for the FPM in Section 2.3.6.

2.3.1 Physical Problems to be Solved

The numerical experiments in Chapter 6 will deal with variety of physical processes that can be simulated using the FPM. Some of these processes are classical CFD problems like the airflow around a car or the flow through a bifurcated tube or a static mixer. Although these problems can be solved very well using classical methods like Finite Volumes, meshfree methods still offer the advantage of not needing a mesh which

would have to be generated before the simulation can start. Some other processes that we will discuss involve more sophisticated problems like moving geometries or free surfaces. An example for the former is the opening or closing of an oil valve and an example for the latter would again be the airflow around a car, but this time with drops of rain or, similarly, a car passing through a body of water [68]. In the case of a valve we are interested in either how much fluid passes the valve after the valve has started to close until it is fully closed or how long it takes until the fluid flow is static after opening the valve. Similarly, when simulating a car in a virtual wind tunnel and virtually spraying it with water, the question is how much water gets into the air intake for the engine or for the air conditioning. Most of these processes can be modeled using the incompressible Navier–Stokes equations together with the continuity equation:

$$\begin{aligned} \frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\,\mathbf{v} &= -\frac{1}{\rho_0}\nabla p + \nu \Delta \mathbf{v} + \mathbf{f} \\ \mathrm{div}\mathbf{v} &= 0 \end{aligned}, \tag{2.26}$$

where ρ_0 is the density of the fluid, $\nu = \mu/\rho_0$ is the kinematic viscosity, $\mathbf{v}$ is the velocity vector, p is the pressure and $\mathbf{f}$ are external forces. While the Navier–Stokes equations enforce the conservation of momentum, the continuity equation enforces the conservation of mass. The Reynolds Number

$$\mathrm{Re} = \frac{L \rho_\infty v_\infty}{\mu} \tag{2.27}$$

determines the characteristics of the flow. Here, L is the characteristic length of the fluid domain, ρ_∞ its characteristic density and v_∞ its characteristic velocity as defined in [108], pp. 267. For small Reynolds numbers ($\mathrm{Re} << 1$) the inertial forces

$$\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\,\mathbf{v} \tag{2.28}$$

are small compared to the viscous forces

$$\nu \Delta \mathbf{v}, \tag{2.29}$$

leading to a laminar flow. In the opposite case of large Reynolds numbers, the inertial forces outweigh the viscous forces and the flow becomes turbulent. Depending on the characteristics of the flow, the FPM employs different approaches of solving the Navier–Stokes equations, see Section 2.3.3. These changes however lead to very different linear systems that need to be solved. The numerical experiments in Chapter 6 will include models with both small and large Reynolds numbers, therefore we will be using both approaches in the FPM in order to accurately simulate the respective processes.

Opposed to the processes that we can model by using the Navier–Stokes equations (2.26), there are other processes that we want to simulate that require to use a different set of equations. One of these processes which we want to examine in Chapter 6 is the flanging of metal. It is possible to incorporate other material models due to the fairly general operator splitting of the stress tensor S that we will introduce in the following section.

2.3.2 Equations to be Solved

In order to solve the Navier–Stokes equations, we solve the three conservation equations (mass, momentum and energy) [35], which describe the relation between the velocity $\mathbf{v} = (v_1, v_2, v_3)^T$, the pressure p, and the temperature T.

Let

$$\frac{\mathrm{d}}{\mathrm{d}t} \equiv \frac{\partial}{\partial t} + \mathbf{v}^T \cdot \nabla \tag{2.30}$$

denote the material derivative, which models the change of a physical property along the trajectory of a fluid particle. As the material derivative is naturally represented by the discretization technique in the FPM, we will employ this operator wherever a time derivative is involved. For the scope of this thesis, we restrict ourselves to the case of incompressible fluids, that is, we have a constant density ($\mathrm{d}\rho/\mathrm{d}t = 0$).
We then have the following formulations of the three relevant conservation equations:

- First, the conservation of momentum is given by

$$\frac{\mathrm{d}}{\mathrm{d}t}(\rho\mathbf{v}) = (\mathrm{div}S)^T - \nabla p + \rho g, \tag{2.31}$$

  where ρ denotes the density, S is the stress tensor (see below), and g contains the external body forces.

- Second, the conservation of mass is enforced by the continuity equation

$$\mathrm{div}\,(\rho\mathbf{v}) = 0. \tag{2.32}$$

- Finally, the conservation of energy is controlled by the temperature equation

$$\begin{aligned}(\rho c_V)\frac{\mathrm{d}}{\mathrm{d}t}T = {} & \mathrm{div}(S\cdot\mathbf{v}) - (\mathrm{div}S)\cdot\mathbf{v} \\ & - p(\mathrm{div}\mathbf{v}) + \mathrm{div}(k\nabla T) + q,\end{aligned} \tag{2.33}$$

  where T denotes the temperature, k the heat conductivity, q the external heat sources and c_V the heat capacity.

We decompose the stress tensor S into its solid and viscous components:

$$S = S_{\mathrm{visc}} + S_{\mathrm{solid}}. \tag{2.34}$$

The viscous stress S_{visc} is given by

$$S_{\mathrm{visc}} = \mu\frac{\mathrm{d}\epsilon}{\mathrm{d}t}, \tag{2.35}$$

where μ denotes the viscosity of the fluid, and

$$\frac{\mathrm{d}\epsilon}{\mathrm{d}t} = \left\{\left[\nabla\mathbf{v} + (\nabla\mathbf{v})^T\right] - \frac{2}{3}\left[\mathrm{div}\mathbf{v}\right] I\right\}. \tag{2.36}$$

is the strain rate tensor[8]. For the solid stress tensor component, we use the formulation

$$\frac{\mathrm{d}S_{\text{solid}}}{\mathrm{d}t} = G\frac{\mathrm{d}\epsilon}{\mathrm{d}t} + (K \cdot S_{\text{solid}} - S_{\text{solid}} \cdot K), \tag{2.37}$$

with the shear modulus G. The tensor K represents rigid rotations, for example (depending on the material) the Jaumann rate [67] can be used here. Details on the choice of K can be found in [121]. Note that for the Navier–Stokes equations we are focusing on we have $S_{\text{solid}} = 0$ which simplifies many of the right-hand sides in the next section. When modeling sand or visco-elastic materials though, $S_{\text{solid}} \neq 0$. Therefore we include S_{solid} in the description here.

Solving equations (2.31), (2.32) and (2.33) for velocity $\mathbf{v}$, pressure p and temperature T using the stress tensor (2.35) yields a solution for the incompressible Navier–Stokes equations (2.26)[9]. Therefore we now turn ourselves to the discretized counterparts of these equations.

2.3.3 Discrete Versions of the Conservation Equations

The Finite Pointset Method employs a Lagrangian scheme, i.e. the points move with the velocity field $\mathbf{v}$. In consequence, using an implicit Euler time integration, the discretization of the material derivative reduces to

$$\frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = \frac{\mathbf{v}_i^{n+1} - \mathbf{v}_i^n}{\Delta t} + \mathcal{O}(\Delta t), \tag{2.38}$$

where the index n denotes the time step.

We use a splitting of the pressure into its hydrostatic part p_{hyd} and its dynamic part p_{dyn}. With the hydrostatic pressure from the current time step $n+1$, which we can compute without knowing the velocity in that time step, and the dynamic pressure from the last time step we have a pressure guess $\tilde{p}$. With that, we can compute an intermediate velocity $\tilde{\mathbf{v}}^{n+1}$. Then, we compute a pressure correction p_{corr} which we use to project the intermediate velocity into a divergence free space. Lastly, we compute the dynamic pressure p_{dyn} using the update velocity.

Solving for the Intermediate Velocity

The momentum equation (2.31) for the intermediate velocity using a pressure guess $\tilde{p}$ at a single point x_i reads as follows:

$$\frac{\tilde{\mathbf{v}}_i^{n+1} - \mathbf{v}_i^n}{\Delta t} = \frac{1}{\rho}\left(\mathrm{div} S^{n+1}(x_i)\right)^T - \frac{1}{\rho}\nabla\tilde{p}(x_i) + g(x_i). \tag{2.39}$$

Analogously to equations (2.35)-(2.37) we decompose $S^n(x_i) = S^n_{\text{visc}}(x_i) + S^n_{\text{solid}}(x_i)$ and write $\frac{\mathrm{d}\epsilon}{\mathrm{d}t}\Big|^n$ to denote the evaluation of the strain rate tensor at time step n. In

[8] Note that in the FPM literature, the strain rat tensor is sometimes given as 1/2 of the right-hand side term in equation (2.36). This leads to factors of 2 in the following equations.

[9] Note that by using different stress tensors here we can use the same approach to solve a different set of equations / material model.

the following, we omit the point coordinates (x_i) as well as the subscript i for better readability. For the solid stress tensor in equation (2.37), the time integration is given by the semi-implicit formulation from [76]:

$$S_{\text{solid}}^{n+1} = S_{\text{solid}}^{n} + \Delta t \left(G \cdot \frac{\mathrm{d}\epsilon}{\mathrm{d}t}\bigg|^{n+1} \right) + \Delta t \left(K^n \cdot S_{\text{solid}}^{n} - S_{\text{solid}}^{n} \cdot K^n \right). \tag{2.40}$$

Using this relation, we can add the contributions of the previous time step to the right-hand side of the discrete momentum equation

$$\hat{g} = g + \frac{1}{\rho} \text{div} \left(S_{\text{solid}}^{n} + \Delta t (K^n \cdot S_{\text{solid}}^{n} - S_{\text{solid}}^{n} \cdot K^n) \right)^T, \tag{2.41}$$

and obtain a much simpler formulation of equation (2.39),

$$\tilde{\mathbf{v}}^{n+1} - \frac{\Delta t}{\rho} \left(\text{div} \hat{\mu} \frac{\mathrm{d}\epsilon}{\mathrm{d}t}\bigg|^{n+1} \right)^T = \mathbf{v}^n - \frac{\Delta t}{\rho} \nabla \tilde{p} + \Delta t \cdot \hat{g}, \tag{2.42}$$

where

$$\hat{\mu} = \mu + \Delta t \cdot G \tag{2.43}$$

denotes the numerical viscosity. From the strain rate tensor equation (2.36), we know that $\frac{\mathrm{d}\epsilon}{\mathrm{d}t}$ only depends on $\mathbf{v}$ and its derivatives, hence we define

$$\Psi_{\hat{\mu}}(\mathbf{v}) = \left(\text{div} \left(\hat{\mu} \frac{\mathrm{d}\epsilon}{\mathrm{d}t} \right) \right)^T. \tag{2.44}$$

For example, for incompressible flow with constant viscosity, we have

$$\Psi_{\hat{\mu}}(\mathbf{v}) = \left(\text{div} \left(\hat{\mu} \nabla \mathbf{v}^T \right) \right)^T. \tag{2.45}$$

With the help of $\Psi_{\hat{\mu}}$, we can rewrite the momentum equation equation (2.39) such that

$$\boxed{\left(I - \frac{\Delta t}{\rho} \Psi_{\hat{\mu}} \right) (\tilde{\mathbf{v}}^{n+1}) = \mathbf{v}^n - \frac{\Delta t}{\rho} \nabla \tilde{p} + \Delta t \cdot \hat{g},} \tag{2.46}$$

where $\tilde{p}$ is an approximate pressure guess that we will define later. The discretization of this equation yields a sparse linear system of equations

$$A_v \tilde{\mathbf{v}}_h^{n+1} = f_h \tag{2.47}$$

for the velocity at time step $n + 1$.

Solving for the Pressure

The velocity $\tilde{\mathbf{v}}^{n+1}$ computed by equation (2.46) does not yet satisfy the continuity equation for incompressible fluids,

$$\text{div} \mathbf{v} = 0. \tag{2.48}$$

To correct the velocity, we need the solution of the pressure for the new time step $n+1$ which is not yet available. Hence, we compute the pressure in three separate steps. Like in standard Chorin projection approaches [28], we take the divergence[10] of the conservation equation for the momentum (2.31) and obtain a Poisson equation for the pressure,

$$\begin{aligned} \mathrm{div}\left(\frac{1}{\rho}\nabla p^{n+1}\right) = & -\mathrm{div}\left(\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{v}^{n+1}\right) \\ & + \mathrm{div}\left(\frac{1}{\rho}\left(\mathrm{div}S_{\mathrm{visc}}\right)^T\right) \\ & + \mathrm{div}\left(\frac{1}{\rho}(g+\mathrm{div}S_{\mathrm{solid}})^T\right). \end{aligned} \tag{2.49}$$

We use the splitting of the tensor S to also split the pressure into two parts

$$p^{n+1} = p_{\mathrm{hyd}}^{n+1} + p_{\mathrm{dyn}}^{n+1}. \tag{2.50}$$

First, we formulate the equation for the hydrostatic pressure,

$$\boxed{\mathrm{div}\left(\frac{1}{\rho}\nabla p_{\mathrm{hyd}}^{n+1}\right) = \mathrm{div}\left(\frac{1}{\rho}(g+\mathrm{div}S_{\mathrm{solid}}^{n+1})^T\right) = \mathrm{div}\hat{g},} \tag{2.51}$$

which can be computed without knowing the new velocity $\mathbf{v}^{n+1}$ (cf. equation (2.41)).

In contrast, to update the dynamic pressure via

$$\boxed{\mathrm{div}\left(\frac{1}{\rho}\nabla p_{\mathrm{dyn}}^{n+1}\right) = -\mathrm{div}\left(\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{v}^{n+1}\right) + \mathrm{div}\left(\frac{1}{\rho}(\mathrm{div}S_{\mathrm{visc}})^T\right),} \tag{2.52}$$

the updated velocity $\mathbf{v}^{n+1}$ is needed before we can solve the system. Hence, in the momentum equation (2.46) we use

$$\tilde{p} = p_{\mathrm{hyd}}^{n+1} + p_{\mathrm{dyn}}^{n}, \tag{2.53}$$

i.e. we use the dynamic pressure from the previous time step. To correct the expected error we introduce at this point, an additional correction pressure p_{corr}^{n+1} that needs to be determined after the predicted velocity $\tilde{\mathbf{v}}^{n+1}$ is computed.
To this end, let us reconsider equation (2.46) and replace $\tilde{p}$ by the corrected pressure $\tilde{p} + p_{\mathrm{corr}}^{n+1}$ as well as the predicted velocity $\tilde{\mathbf{v}}^{n+1}$ by the corrected velocity $\mathbf{v}^{n+1}$,

$$\left(I - \frac{\Delta t}{\rho}\Psi_{\hat{\mu}}\right)(\mathbf{v}^{n+1}) = \mathbf{v}^n - \frac{\Delta t}{\rho}\nabla\tilde{p} - \frac{\Delta t}{\rho}\nabla p_{\mathrm{corr}}^{n+1} + \Delta t\cdot\hat{g}. \tag{2.54}$$

Subtracting equation (2.46) from equation (2.54) and applying the divergence operator yields

$$\mathrm{div}\mathbf{v}^{n+1} - \mathrm{div}\tilde{\mathbf{v}}^{n+1} - \mathrm{div}\left[\frac{\Delta t}{\rho}\nabla\Psi_{\hat{\mu}}(\mathbf{v}^{n+1} - \tilde{\mathbf{v}}^{n+1})\right] = -\mathrm{div}\left(\frac{\Delta t}{\rho}\nabla p_{\mathrm{corr}}^{n+1}\right). \tag{2.55}$$

[10] Using the convention $\mathrm{div}S = \sum_k \frac{\partial S_{ki}}{\partial x_k}\mathbf{e_i}$.

We can neglect the term

$$\operatorname{div}\left[\frac{\Delta t}{\rho}\nabla\Psi_{\hat{\mu}}(\mathbf{v}^{n+1}-\tilde{\mathbf{v}}^{n+1})\right]\approx 0, \tag{2.56}$$

if the time step is small enough, while for incompressible flow we require

$$\operatorname{div}\mathbf{v}^{n+1}=0. \tag{2.57}$$

If, however, the ration of viscosity over density $\hat{\mu}/\rho$ is very high, then a suitable time step might be too small to allow for an effective simulation, which would then need a very high number of time steps. In these cases, we cannot use Chorin's projection method and need to solve for velocity and correction pressure in one single linear solve, see equations (2.61).

From equation (2.55), it remains to solve the Poisson equation

$$\boxed{\operatorname{div}\left(\frac{\Delta t}{\rho}\nabla p_{\text{corr}}^{n+1}\right)=\operatorname{div}\tilde{\mathbf{v}}^{n+1},} \tag{2.58}$$

and use the gradient of the pressure correction p_{corr}^{n+1} to update the velocity field:

$$\mathbf{v}^{n+1}=\tilde{\mathbf{v}}^{n+1}-\frac{\Delta t}{\rho}\nabla p_{\text{corr}}. \tag{2.59}$$

This is called the *segregated* approach.

The discrete forms of the hydrostatic pressure equation (2.51), the dynamic pressure equation (2.52) and the correction pressure equation (2.58) lead to scalar linear systems of the form

$$A_p p_h = f_h, \tag{2.60}$$

which need to be solved in every time step.

Remark 2.13. If the same boundary conditions are enforced, the matrices are equal within a time step. In the FPM, this is the case for the hydrostatic pressure and the correction pressure. For the dynamic pressure however, the boundary conditions may be different from the other two equations. This may be necessary in order to accurately capture the physical pressure observation. Both Neumann and Dirichlet boundary conditions at the outflow boundary can be physically justified. Neumann boundary conditions lead to singular linear systems though and Dirichlet boundary conditions can introduce a steep pressure gradient at the boundary that has no physical meaning. Therefore, the FPM has the option to assign different boundary conditions in the dynamic pressure system. For example, a mixed boundary condition can be used or parts of the boundary can be treated like interior points, so no boundary condition is posed on them. The latter option can require some tweaking of the linear system before it goes into the linear solver in order to make it regular again. This is a very special case though which we will not go into here.

A drawback of the segregated approach is the lack of accuracy in the case of low Reynolds numbers: The term in equation (2.56) becomes large because of the high viscosity. Therefore neglecting it introduces an error. This leads to non-accurate results, as we will show in Section 2.3.4. An option to overcome this problem is to solve for velocity and correction pressure in one single linear solve, as described in [76]. In this formulation, we do not omit the term (2.56) but use the original momentum equation (2.54). The correction pressure then serves as a Lagrangian multiplier, i.e. we solve the saddle point system

$$\boxed{\begin{aligned}&\left(I - \frac{\Delta t}{\rho}\Psi_{\hat{\mu}}\right)\left(\tilde{\mathbf{v}}^{n+1}\right) + \frac{\Delta t}{\rho}\nabla p_{\text{corr}}^{n+1} = \mathbf{v}^n - \frac{\Delta t}{\rho}\nabla\tilde{p} + \quad \Delta t \cdot \hat{g} \\ &\left(\text{div}\tilde{\mathbf{v}}^{n+1}\right) - \text{div}\left(\frac{\Delta t_{\text{virt}}}{\rho}\nabla p_{\text{corr}}^{n+1}\right) = 0,\end{aligned}} \tag{2.61}$$

or, in matrix form,

$$\begin{pmatrix}\mathbf{A_h} & B_h \\ C_h & -D_h\end{pmatrix}\begin{pmatrix}\tilde{\mathbf{v}}_h \\ p_h\end{pmatrix} = \begin{pmatrix}\mathbf{g_h} \\ f_h\end{pmatrix}. \tag{2.62}$$

The virtual time step Δt_{virt} is chosen as a fraction of the time step Δt. Theoretically, the best possible value is $\Delta t_{\text{virt}} = 0$, but smaller values make the linear system harder to solve. Section 2.4.3 will deal with the special properties of these coupled systems.

Solving for the Temperature

Finally, the new temperature T^{n+1} needs to be determined. As for velocity and pressure, we use an implicit scheme [76],

$$\begin{aligned}&(\rho c_V)\cdot T^{n+1} - \Delta t\cdot\left(k\cdot\nabla T^{n+1}\right) \\ =&(\rho C_V)\cdot T^n + \Delta t\left(\text{div}(S^{n+1}\cdot\mathbf{v}^{n+1}) - (\text{div}S^{n+1})\cdot\mathbf{v}^{n+1}\right) \\ &- \Delta t\left(p^{n+1}(\text{div}\mathbf{v}^{n+1}) + q\right) \\ =&\,\hat{q}.\end{aligned} \tag{2.63}$$

The right-hand side $\hat{q}$ only depends on quantities known at this stage. For the left-hand side, we introduce $I_T - \Theta_T = (\rho c_V) - \Delta t\nabla^T\cdot(k\nabla)$ and obtain the heat equation

$$\left(I_T - \Theta_T\right)T^{n+1} = \hat{q}. \tag{2.64}$$

Conclusion and Algorithm

To conclude this section, we recapitulate the equations we need to solve in every time step:

1. Compute the hydrostatic pressure p_{hyd}^{n+1} according to equation (2.51).
2. Set $\hat{p} = p_{\text{hyd}}^{n+1} + p_{\text{dyn}}^n$ and compute the new velocity field $\mathbf{v}^{n+1}$ as well as the correction pressure p_{corr}^{n+1},

- either by the *segregated* approach:
  First compute the velocity predictor using equation (2.46), then solve the Poisson equation (2.58),
- or by the *coupled approach*:
  Solve the saddle point system (2.62).

In both cases, use the gradient of p_{corr}^{n+1} to correct the velocity using equation (2.59).

3. Compute the dynamic pressure p_{dyn}^{n+1} according to equation (2.52).

4. Update the temperature using equation (2.64).

After the velocity and pressure values are known, we use the velocity field to move the point cloud[11], re-organize it where necessary, see Section 2.3.5, and start the next time step. Note that we do not need to solve a non-linear equation in the whole solution process; the non-linearity is absorbed in the splitting of the pressure and the stress tensor. Instead, we need to solve three (segregated) or two (coupled) scalar Poisson-like equations as well as a three-dimensional system (segregated) or a four-dimensional saddle point system (coupled). In practice, the temperature equation as well as the vectorial velocity equation in the segregated approach can be solved easily using an one-level method like BiCGStab2 [132], see the experiment in Section 4.5.1.

2.3.4 Comments on the Coupled and the Segregated Approach in the FPM

In the previous section we have introduced two approaches to resolve the non-linearity of the Navier–Stokes equations (2.26) using a splitting of the stress tensor S. We also mentioned that the choice which approach to use depends on the Reynolds number of the problem. The coupled approach is much more accurate compared to the segregated approach, however we will see later that it is also much more expensive in terms of computation, which goes back to the saddle point structure of the arising coupled velocity-pressure system.

It is hard though to define a hard cut in terms of Reynolds numbers that allow for a sufficiently accurate simulation using the segregated approach versus Reynolds numbers that require the coupled approach to be used. In the end, this choice also depends on the required accuracy which may very well vary between different use cases. As a rule of thumb though, the coupled approach should always be used for Reynolds numbers smaller than 0.1 and should be strongly considered already with Reynolds numbers smaller than 1. Also note that the coupled approach will always give the more accurate results. At the same time, its computational cost is always higher. If run-time is not an issue, it is advisable to use the coupled approach.

[11] For details on the (second order) point cloud movement, see [142].

Numerical Experiment

In order to give an example which clearly shows that the segregated approach is not always sufficient, we look at a simple pipe with an inflow at one end and an outflow on the other end. At the inflow we set a constant velocity boundary condition of $v_0 = 1.0m/s$ and a Neumann boundary condition for the velocity at the outflow. With a Reynolds Number of 10^{-3}, the viscous forces are dominant in this laminar flow. As discussed in Section 2.3.3, the coupled approach in the FPM should be applied here. For this problem we know that for any cross section of the pipe that is far enough away from the inflow, we expect the maximum velocity $v_{\max} = 2v_0$ to occur in the center of the pipe with a quadratic fall-off to the sides.

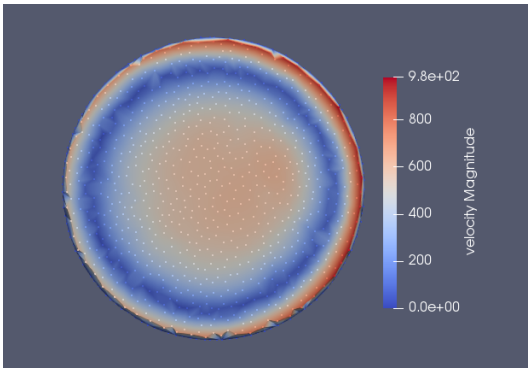


Figure 2.6: Cross section of a pipe, segregated approach, v_0=1.0, L=10mm, d=2mm, $\rho = 1000kg/m^3$, $\mu = 1000$kg/ms, $t = 0.0002$s.

After $t = 0.0002s$, using the segregated approach, our simulation has already produced velocities of $\approx 100m/s$ and does not show the parabolic profile in the cross sections, see Figure 2.6.

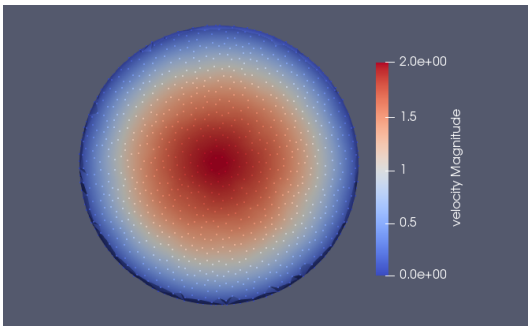


Figure 2.7: Cross section of a pipe, coupled approach, $\Delta t_{\text{virt}} = 0.1$, v_0=1.0, L=10mm, d=2mm, $\rho = 1000kg/m^3$, $\mu = 1000$kg/ms, $t = 0.0002$s.

With the coupled approach and a virtual time step size of $\Delta t_{\text{virt}} = 0.1\Delta t$, see Section 2.3.3, we get $v_{\max} = 2.0m/s$ both at $t = 0.0002s$ and $t = 0.1s$, showing that the scheme reaches the analytical steady state solution. With this approach, the method shows the parabolic velocity distribution in the cross sections as well, see Figure 2.7. In both cases, roughly 31,000 FPM points were used. As shown in Figure 2.8, the

	p_{hyd}	$\mathbf{v}$ (segregated)	$\mathbf{v}$ (coupled)	p_{corr}	p_{dyn}
Segregated	-	14	-	35	10
Coupled	-	-	70	-	8

Figure 2.8: Average number of BiCGStab2 iterations for the linear systems. Since we did not include gravity in this model the external body forces are 0 and so is the solid stress tensor S_{solid}. Therefore the hydrostatic pressure is also 0 meaning that we do not need to solve a linear system for the hydrostatic pressure.

number of BiCGStab2 iterations needed to solve the coupled velocity-pressure system is significantly higher than for the segregated velocity system, which is due to the properties of the saddle point structure of the linear system in the coupled case. Note that while all linear systems for the different pressures have as many equations as there are points in the point cloud, the segregated velocity system has three equations for every point (x-, y- and z-velocity components) and the coupled velocity system has four equations per point (x-, y- and z-velocity components plus one equation for the correction pressure). As explained in Section 2.4.3, the coupled velocity-pressure system also has about $160/120 = 4/3$ times as many entries per row. Therefore, neglecting architectural effects like caching and vectorization, the computational cost of one matrix-vector product for the coupled system is about $16/9 = 1.78$ times higher than for the segregated system. Taking the cost of the matrix-vector product as a baseline, in the segregated approach the cost for solving all three linear systems sums up to 59 units, whereas in the coupled approach the sum is 132.6, with 124.6 units from solving the coupled velocity-pressure system. This makes it clear that having an efficient solver that needs as little iterations as possible for solving this system is essential for the overall performance of the method. On the other hand, this experiment shows that in situations with small Reynolds Numbers the segregated approach cannot be used and the coupled approach is the only option giving realistic results.

2.3.5 Discretization of the Domain and Point Cloud Management

The key point in the FPM is that we can use Generalized Finite Difference Methods on point clouds as introduced in Section 2.2.4 to solve the discretized equations (2.51), (2.46), (2.58), (2.61), (2.52) and (2.64) from Section 2.3.3. To this end, we need an initial point cloud that is used in the first time step. The idea is to fill the entire fluid domain starting from the (triangulated) boundary. Onto the boundary, we can place regularly spaced boundary points. Each of those boundary points then introduces interior points in its neighborhood inside the domain, where attention needs to be paid to the user-prescribed values $h_{\max}(\mathcal{P})$ and $h_{\min}(\mathcal{P})$ from Definition 2.3. The newly generated interior points then continue the process, filling the entire fluid domain layer by layer.

Remark 2.14. With this strategy, the number of points $\mathcal{N}(\mathcal{P})$ cannot be prescribed a priori but is a property resulting from the choice of h in Definition 2.4.

Point Cloud Management

Since the points in the point cloud are moved in every time step according to the computed velocity field $\mathbf{v}$, there are multiple issues that can occur with respect to the point distribution:

- Holes can form in the point cloud, i.e. there may exist balls $B_r(x)$ of radius $r = H_{\max} = c_{\max} \cdot h$ around a particular location x within the fluid domain such that
$$\forall x_i \in \mathcal{P} : \; x_i \notin B_r(x). \tag{2.65}$$
- Points can come too close to each other, i.e.:
$$\exists x_i, x_j \in \mathcal{P} : \; \|x_i - x_j\| < H_{\min} = c_{\min} \cdot h. \tag{2.66}$$
Not only does this lead to a point cloud that has more points than necessary, but it also leads to numerical instabilities when solving the linear system, as the condition number increases.
- Points can leave the computational domain through an outflow boundary.

Therefore, a point cloud management phase is done at the beginning of every time step. If a hole is detected according to equation (2.65), it can simply be filled with new points. This is a very special property of the FPM and it is only possible to do so, because the points do not carry any mass. They are merely collocation points for discretizing the equations derived in Section 2.3.3. When filling holes in the point cloud, we need to distinguish between *numerical* holes that only occurred due to the movement of the points and *physical* holes that are a physical observation in the process being simulated. An example for such a process is the sloshing of a fluid in a tank: Here, plunging waves form which naturally lead to "holes" in the fluid, but those holes need to be captured by our method as well and must not be refilled with artificial points. The idea is to fill in more points only if the hole in the point cloud is larger than $H_{\max}$ but smaller than $H_{\text{hole}} = c_{\text{hole}} \cdot h$.

On the other hand, just as we can add points to the point cloud in order to fill holes, we can seamlessly delete points as well. This is necessary in order to control the numerical stability of the discretization. If two points are too close to each other, i.e. their distance is smaller than $H_{\min}$, we have to options: Deleting one of the two or deleting them both and inserting a new point half way between them. Both options are very similar, but the latter option has shown to produce slightly better results in some cases, see [126].

Remark 2.15. The values $c_{\max}$, $c_{\min}$ and c_{hole} are user parameters. Their value slightly depends on the particular problem, but common choices are $c_{\max} = 0.25$, $c_{\min} = 0.05$, $c_{\text{hole}} = 0.4$.

Lastly, points may leave the computational domain through an outflow boundary condition. These points can simply be deleted for the same reason as mentioned above. For more details on the point cloud management, see [35] and [143].

The lack of mass associated with a point furthermore allows the point cloud to be h-refined easily according to a number of different criteria. This is done by introducing more points in a particular area of interest. Areas of interest can either be defined manually in terms of geometric regions or by certain properties that are a result of the simulation itself, for example the current velocity gradient. For processes with large velocity gradients in some regions we can therefore use dense point clouds in those regions and coarser point clouds in other regions without explicitly defining those regions. We simply define a local smoothing length h depending on the velocity gradient, which will automatically trigger the necessary merging and addition of points in the point cloud management phase. This saves a lot of computational complexity compared to solving the problem with a dense point cloud in the full domain. For the purposes of this work, the refinement of the point cloud does not play a significant role. The interested reader is referred to Hagmann [55] for more details on the adaptive refinement of the point cloud.

In order to efficiently organize the point cloud, some geometrical computations need to be carried out very often. This includes, but is not limited to, calculating distances between points, calculating distances between points and triangles and determining neighborhoods. These computational challenges are not related to solving the linear systems and are therefore beyond the scope of this work. They are however important when implementing the FPM efficiently for industrial-sized applications.

Since industrial-sized simulations are usually carried out on a compute cluster with multiple cores using MPI, the point cloud organization stage also needs to take care of distributing the point cloud evenly across all processes. The metric for this in the industrial implementation of the FPM that we use here is solely based on the number of points that each individual core has to manage, which is targeted to be constant across the cores. Other options here would be to weigh certain types of points with a higher computational cost than others, thus reducing load balancing issues that can occur in our current approach. Again, this is beyond the scope of this thesis and will be the target of future work on the method itself.

2.3.6 Discretization of the Boundary

In meshfree methods, imposing boundary conditions is often quite involved. For FPM, the implementation of boundary conditions is not as involved, but has implications for the linear systems. As introduced in the previous section, the boundary is represented by a discrete set of points placed on the boundary. Boundary conditions are imposed by either setting up the stencil corresponding to the desired boundary operator (e.g. for Neumann boundary conditions for which the normals of the boundary need to be computed based on neighboring boundary points, see [141] and [114]) at the boundary points or — in case of Dirichlet boundary conditions — by setting the central stencil entry to 1 and the respective entry on the right-hand side to the desired value. Therefore, just like in FDMs, the central stencil values are

	FDM	GFDM
Symmetry	+	-
M-matrix	+	-
Essentially irreducible	+	(+)
Essentially diagonally dominant	+	-
Sparsity	+	(+)
Numeration of Variables	+	-
Elimination of Boundary Conditions	+	-
Static structure	+	-

Figure 2.9: Properties of FDM and GFDM matrices. (+) indicates partially fulfilled properties.

- $\mathcal{O}(1)$ at Dirichlet boundaries,
- $\mathcal{O}(h^{-1})$ at Neumann boundaries and
- $\mathcal{O}(h^{-2})$ in the interior.

Note that because the coupling structure of GFDMs compared to standard FDMs on regular grids is so dense, it is not possible to eliminate the boundary conditions from the linear system with reasonable computational effort, cf. Section 2.4.1.

2.4 Properties of the Linear Systems Arising in GFDMs

This section will deal with the linear systems $Au = f$[12] arising from GFDM discretizations on point clouds as introduced in Section 2.2.4. The observations we make here are true in particular for the linear systems that need to be solved in the FPM which we have introduced in Section 2.3. It is important to be aware of those properties because they determine which linear solvers can be employed to solve the linear systems and how efficient they will be.

The more complex the model to be simulated becomes or the more accurate the results of the simulations have to be, the more points we need in order to discretize the simulation domain. Therefore, the number of degrees of freedom in the linear systems that we need to solve increases. Solving the linear systems becomes a bottleneck as most linear solvers do not scale linearly with the number of degrees of freedom. Multigrid methods on the other hand scale linearly in the number of degrees of freedom n, their number of operations is $\mathcal{O}(n \log \varepsilon)$, i.e. their iteration number is $\mathcal{O}(\log \varepsilon)$, where ε is the desired relative residual reduction. Therefore, they are very well suited especially for large problems. However, not every linear system is suited to be solved using Multigrid methods. There are certain properties of the corresponding matrix A that have to be fulfilled in order to theoretically guarantee convergence. Some other

[12]Dropping the subscript h for readability, whenever it is not needed.

properties influence the efficiency of different linear solvers. Figure 2.9 shows some of these properties and compares matrices arising from classical FDMs to those from GFDMs with respect to these properties.

Note that theoretically verified AMG methods only exist for symmetric, positive-definite M-matrices (using Ruge-Stüben coarsening, [138]) or also non-symmetric M-matrices (using a special aggregative coarsening strategy, [104]). Lottes [89] recently constructed an AMG method that is targeted at *inherently* non-symmetric problems, for example convection-dominated equations, for which he successfully proved two-level convergence. In order to construct an efficient AMG method it is necessary to understand the properties of the linear systems that need to be solved. We will introduce Multigrid methods and the issues that come with them in the context of GFDMs in Section 3.3.

The outline of this section is as follows: First we will examine the properties of matrices arising in GFDMs for Poisson problems. This is a natural first step, as the linear systems (2.51), (2.58) and (2.52) arising in the FPM in particular are Poisson-like[13] systems. Many of the comments we make throughout this particular subsection will hold true for matrices resulting from the GFDM discretizations for other problems as well. Section 2.4.2 will briefly deal with the linear systems for the predicted velocity in the segregated approach. In Section 2.4.3 we will look at the linear systems from equation (2.61) in the coupled FPM approach in more detail. These systems are very different to the Poisson-like systems due to their saddle point structure. Finally, we will turn to the changing structure of the linear systems between the time steps.

2.4.1 Poisson-like Pressure Systems

Symmetry and M-Matrix Property

In GFDMs on point clouds, the stencil at every point is constructed based on local information only. Similarly, equation (2.22) shows that the stencil entries at x_i depend on the distances and the positions of the points in the neighborhood N_i only. Since any two distinct neighborhoods N_i and N_j will be different from each other, any two stencils will also be different from each other.

For this reason, there is no guarantee that $a_{ij} = a_{ji}$ in the assembled matrix A. In fact, because the FPM uses a maximum of only 40 or 20 neighbors per point (in 3D and 2D respectively) that are not mutually the same, it is also possible to have $a_{ij} \neq 0$ but $a_{ji} = 0$, which makes the matrix structurally non-symmetric.

Attempts have been made to construct GFDMs leading to symmetric linear systems for example by Chiu et al. [26] [27] and Robinson et al. [115]. As Suchde describes in [141], these methods have two major drawbacks: First, the linear systems that need to be solved in order to determine the discrete operators become very large[14]

[13] For models with constant density, which is the case in incompressible flow when we do not consider phase-change phenomena like melting or freezing, they actually are Poisson systems.

[14] In fact there is one *global* linear system that includes all the subsystems determining the local stencils plus a couple of rows that couple all the local subsystems. Keep in mind that the number of local subsystems is equal to the number of points, which makes the overall linear system very large.

compared to the system (2.22) that we need to solve in the classical Least Squares formulation. Secondly, for the discrete Laplacian in particular, the method by Chiu et al. implies $c_{ii}^{\Delta} = 0$, leading to an unstable operator. To the author's knowledge there is no method to obtain symmetric matrices in the general GFDM case, hence we will accept the non-symmetry for the rest of this thesis.
Not only does the Least Squares approach yield non-symmetric matrices, but it also does not guarantee non-positive stencil entries away from the central point, see Example 1 in [127]. In consequence, the resulting matrix is not necessarily an M-matrix. In his work, Seibold [127] [126] introduces means to overcome the mixed signs of the stencil coefficients by replacing the Least Squares problem by a Linear Minimization approach. It turns out that while these approaches lead to M-matrices, they do require extra care in the point cloud management and are only applicable to points that do not have any boundary points in their neighborhood. Also, Seibold states that the improvements he saw in the linear solver in terms of run-time were merely caused by his stencils being smaller (leading to a sparser matrix) than the Least Squares ones, and not by the M-matrix character.

Essentially Irreducible

When working with a fixed mesh with fixed neighborhood relationships like in standard FDMs, a proper meshing of the domain will make sure that every point of the mesh is connected to at least one Dirichlet point, assuming a Dirichlet boundary is available. Point x_i is *connected* to x_k if there exists a path between the two points in the *adjacency graph* corresponding to the mesh. Then, in the corresponding matrix A every row will be connected to at least one Dirichlet row, which makes A an *essentially irreducible* matrix. See Section 5.2 for more thorough definitions of those terms.
This idea also carries over to GFDMs on point clouds: Two points x_i and x_j are considered adjacent, if $a_{ij} \neq 0$ or $a_{ji} \neq 0$ in the corresponding matrix A. For a connected domain Ω and a sufficiently fine point cloud, GFDMs will also yield essentially irreducible matrices. However, one of the main advantages of methods like the FPM is that the point cloud is moving in every time step. Now consider a problem with free surfaces, for example a car driving through a body of water. In the course of the simulation, drops of water leave the main body of water and therefore the simulation domain is not connected any more. The full matrix A is then not essentially irreducible any more, however the block of A corresponding to the main body of water still is. One-level solvers are not sensitive to the full matrix A not being essentially irreducible, however Multigrid methods are, as we will see in Section 5.4.3. We therefore employ a parallel algorithm to find independent submatrices in A and then treat them separately, cf. Section 5.2.

Sparsity

Since checking for and ensuring geometric conditions on the point distribution in every neighborhood to guarantee a solution of the linear system (2.10) would be rather

involved (cf. Section 2.2.4), most methods including the FPM use more neighboring points to construct the stencils than would be necessary in the optimal case. Note that for the FPM there are methods to construct *minimal* stencils [127], which are not applicable to every situation, though. For the Poisson problem, which we need to solve in the FPM, there exist Finite Difference stencils of second order with five (2D) and seven (3D) non-zero entries per point. In the FPM on the other hand, at least 20 (2D) or 40 (3D) neighbors are used [141]. Usually, there are more than 20 or 40 neighbors within the radius h around each point. Using the 40 closest neighbors (after merging and adding points in the point cloud management phase, cf. Section 2.3.5) has been demonstrated to be sufficient in order to guarantee a stable discretization. In classical AMG, the Galerkin product RA_hP is used to construct the coarse level operator [138], which can lead to fairly dense coarse level operators unless alternative coarsening strategies are applied, cf. Section 5.4.1. Less sparsity means more computational work for the linear solver. In the Multigrid framework, this is especially important for the *smoother* (cf. [138]). It is therefore important to achieve a good balance between having enough neighbors to ensure that the differential operators can be discretized with the desired accuracy, but at the same time keep the number of neighbors as low as possible to minimize the computational effort needed in the linear solver. To this end, a nice property of the method we introduce in Section 3.3.2 is that it allows to control the number of neighbors on the coarse levels to some extent. Experiments with AMG and different neighborhood sizes can be found in Section 4.5.5.

Numeration of Variables

Since the initial point cloud is not created in a structured way like a mesh in classical CFD methods, the numeration of variables is different from these methods as well: At the start of the simulation, the points are introduced starting at the boundary of the domain, filling towards the interior. While this is somewhat structured, this structure breaks apart when the point cloud starts moving due to the time stepping. In addition to that, because of the point cloud management, points are merged and added.

When working with MPI, points need to be transferred from one MPI process to the other at some point, causing yet another source of a non-structured numeration of the variables. This leads to a large bandwidth of the resulting linear system. For the FPM, a typical matrix structure without any renumbering of the points is shown in Figure 2.10. This figure was taken from a linear system for the hydrostatic pressure with the point cloud in the initial time step. In Section 5.3 we will see how this particular matrix can be renumbered.

At this point we want to point out that all these characteristics of the FPM and other GFDMs do not only impact the structure of the resulting linear system, but have similar impact on all the other operations within the method itself, like the neighborhood detection for example. Therefore, some of the arguments we make regarding computational issues in the next paragraphs also hold true for the rest of the method and are not limited to the linear solver specifically. Discussing those impacts aside from the linear systems would be beyond the scope of the thesis, though, and could

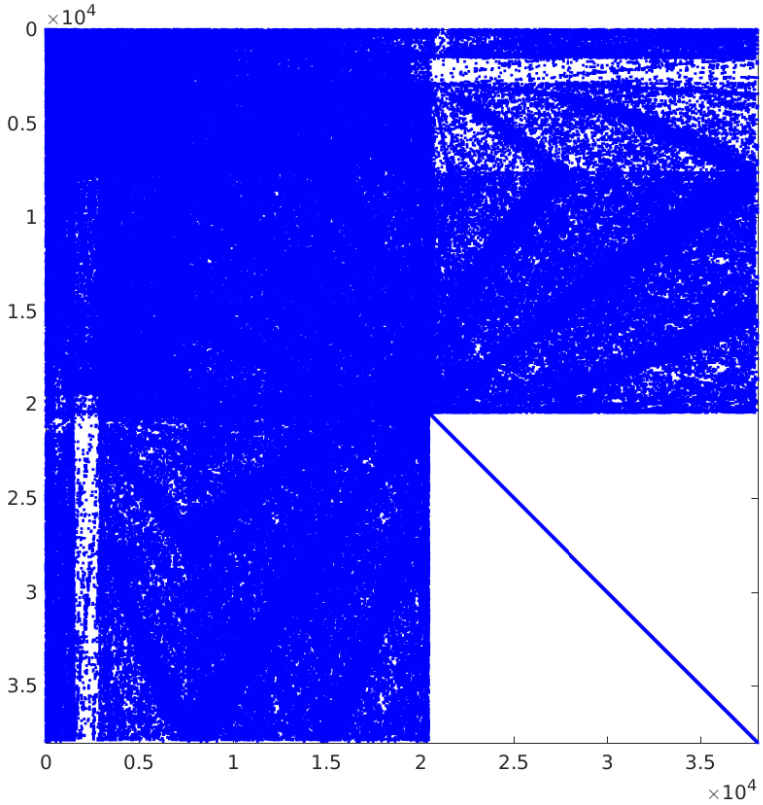


Figure 2.10: Matrix structure for the hydrostatic pressure system in the initial time step of the bifurcated tube model, cf. Section 4.5. The upper left block corresponds to couplings in the interior of the domain whereas the lower right block corresponds to boundary conditions. Note that in this case the ratio between inner points and boundary points is exaggerated.

be the topic of future research on GFDMs and the FPM in particular.

When focusing on the linear system, there are two main aspects that are impacted by the non-regular numbering: First, the computation aspect and secondly the algorithmical aspect. Algorithmically, let us note that most iterative linear solvers depend on the order of the rows in the matrix, which can be used to some extent in order to improve their convergence rates, see [54] for an example. It is not easy to determine what is a particularly good or bad order[15] but order definitely does influence the outcome of an iteration.

For Multigrid methods, we also note that the coarsening in the setup phase highly depends on the order of the matrix rows. Here, depending on the particular problem, certain numeration schemes can be particularly bad, especially in the MPI case[16].

Computational wise, a high-bandwidth matrix with no specific pattern – the 40 coefficients in each row are in different locations for each row – leads to inefficient prefetching [154] and vectorization [32]. Therefore, we can expect some improvement from renumbering the rows of the matrix. Thinking about the MPI case again, we see that depending on the way we distribute the point cloud across processes, the amount of communication required between those processes varies. The goal of an efficient renumbering strategy should be to minimize the necessary communication, as MPI communication is typically much more expensive compared to the actual computation.

So the issue of the almost random numeration of variables is twofold: First, we have effects like the inefficient prefetching that are affected by the order – and therefore the bandwidth – of the matrix within each process. Secondly, some effects are due to the distribution of the points across processes, for example the amount of communication required. Both phenomena can be dealt with by using renumbering strategies.

Section 5.4.4 will examine the possible benefits of using the Reverse Cuthill-McKee algorithm [87] and the PT-SCOTCH algorithm [25] for these purposes.

Scale of Boundary Conditions

In matrices arising from FDMs, Neumann and Dirichlet boundary conditions are usually eliminated from the matrix and only affect the right-hand side of the linear system. Due to the nature of GFDMs on point clouds, this is not as easy with the matrices we are looking at. Let A_{II} be the submatrix of couplings from interior to interior points, A_{IB} and A_{BI} the submatrices of couplings from interior to boundary and boundary to interior points respectively, and finally let A_{BB} be the submatrix of couplings from boundary to boundary points. With a suitable renumbering of the points we then

[15] Except for in some special cases.

[16] This is because our Multigrid method assumes that couplings across process boundaries are weak couplings. If the process boundaries are placed in such a way that they cut exactly those couplings that are supposed to be strong, the choice of coarse level rows leads to a relatively bad coarse grid correction. See Section 5.1 for details.

have to solve

$$\begin{pmatrix} A_{II} & A_{IB} \\ A_{BI} & A_{BB} \end{pmatrix} u = \begin{pmatrix} f_I \\ f_B \end{pmatrix} . \tag{2.67}$$

Note that $(A_{IB})^T \neq A_{BI}$. Also note that, independently of the boundary conditions, the coefficients in A_{II} are of order $\mathcal{O}(h^{-2})$, $A_{II} \in \mathcal{O}(h^{-2})$ for short. On the other hand, the scaling of the entries in A_{BB} depends on the boundary conditions used. For pure Dirichlet boundary conditions for example, $A_{BB} \in \mathcal{O}(1)$. Neumann conditions on every boundary would lead to $A_{BB} \in \mathcal{O}(h^{-1})$. This obviously has a very negative impact on the range of eigenvalues – and also the condition number – of the linear system (see below). In addition to that, the different scales of the interior part versus the boundary part of the matrix start to mix on the first coarse level when using AMG, because the coarse level operator is determined by using the Galerkin product $A_H = RA_hP$. On the coarse level, the *strong couplings* then cannot be determined by looking at the off-diagonal coefficients any more, because the influence of the boundary conditions is not taken into account correctly.
The multigrid approach presented in Section 3.3.2 does not suffer from this situation as it constructs the coarse grid operator based on a coarser point cloud rather than on the fine level matrix.
It is impractical to use a Schur complement ansatz, see [172] Chapter 1 for example, to solve

$$\begin{pmatrix} A_{II} - A_{IB}A_{BB}^{-1}A_{BI} & 0 \\ A_{BI} & A_{BB} \end{pmatrix} u = \begin{pmatrix} f_I - A_{IB}A_{BB}^{-1}f_B \\ f_B \end{pmatrix} , \tag{2.68}$$

which would decouple the interior part from the boundary part, because for most applications, A_{BB} is too big to be inverted directly and it does not have any specific properties that would allow a robust application of any iterative method to approximate the inverse, which would still be extremely expensive.

Even if A_{BB}^{-1} is available or easy to compute, then it is also unclear what properties $A_{II} - A_{IB}A_{BB}^{-1}A_{BI}$ has, which would then make it difficult to solve

$$\left(A_{II} - A_{IB}A_{BB}^{-1}A_{BI}\right) u = \left(f_I - A_{IB}A_{BB}^{-1}f_B\right) \tag{2.69}$$

in the next step. For example, the Schur complement does not even have to be sparse, nor does it fulfill any properties regarding symmetry, M-matrix character and the like. This is mainly because it is not derived from a partial differential equation any more, but is merely a purely algebraic operator.

Scaling the original matrix A row-wise so that $a_{ii} = 1$ for every i, i.e. using

$$\hat{A} = \begin{pmatrix} a_{11} & & \\ & \ddots & \\ & & a_{nn} \end{pmatrix}^{-1} A \tag{2.70}$$

instead of A, yields a system with a smaller range of eigenvalues (see Figure 2.11) and improves the convergence of BiCGStab (see Section 3.3.2 Figures 3.5, 3.6 and

3.7). In the context of AMG, a row-wise scaling is usually not advisable as it makes a symmetric matrix non-symmetric and destroys the symmetry of the couplings a_{ij} and a_{ji}, which is used in the coarsening process. In many cases, extreme care must be taken in order to preserve a certain scaling for AMG or it is even necessary to re-scale a discretized problem so that AMG is applicable to that problem, see for example [48].

In our case however, we have a non-symmetric matrix to begin with and the notion of strong couplings in our matrix is non-symmetric from the beginning. The scaling (2.70) therefore does not sacrifice any properties of the matrix but has a rather positive impact, see Section 4.5.2.

As mentioned before, the considerations of varying scales only play a role when one-level methods like BiCGStab or AMG methods are employed. The former suffer because their convergence mainly depends on the range of eigenvalues (by absolute value) of A, see [119] Section 6.11.3, and the latter suffer because of the mixing of scales on the coarse level. Both arguments do not apply to the geometrically motivated idea in Section 3.3.2.

Eigenvalues

The eigenvalues of A are important to us as they influence the convergence of most linear solvers. BiCGStab for example is known to show convergence rates that depend on the absolute range of eigenvalues, although this has not been proven theoretically. Note that when we talk about "smallest" and "largest" eigenvalues, we mean "smallest" or "largest" by absolute value, as the matrices we are dealing with are generally non-symmetric.
For the purpose of this analysis we will restrict ourselves to the Poisson equation

$$\begin{cases} -\Delta u &= f \text{ in } \Omega \\ u &= g \text{ on } \partial\Omega \end{cases} \tag{2.71}$$

with $\Omega = [0, L_1] \times [0, L_2]$. The point clouds we are working with are generated by a uniform distribution across Ω, but we make sure that the minimum distance between any two points is $0.1h$.

First, we want to refine the point cloud and observe the changes in the eigenvalues of the resulting matrices. Since the FPM is often applied to problems with fairly detailed geometries, a sufficiently dense point cloud is required to correctly discretize the domain. Consider the Poisson equation (2.71) on the unit square, i.e. $L_1 = L_2 = 1$, with a varying smoothing length h.

Figure 2.11 shows the smallest and largest eigenvalues of the matrix A discretizing the Laplacian in (2.71) with and without the normalization. While in both cases the range of eigenvalues increases when the point cloud becomes finer, the original matrices show the larger range with four to six orders of magnitude and the normalized matrices show the smaller range with two to four orders. We therefore expect BiCGStab to

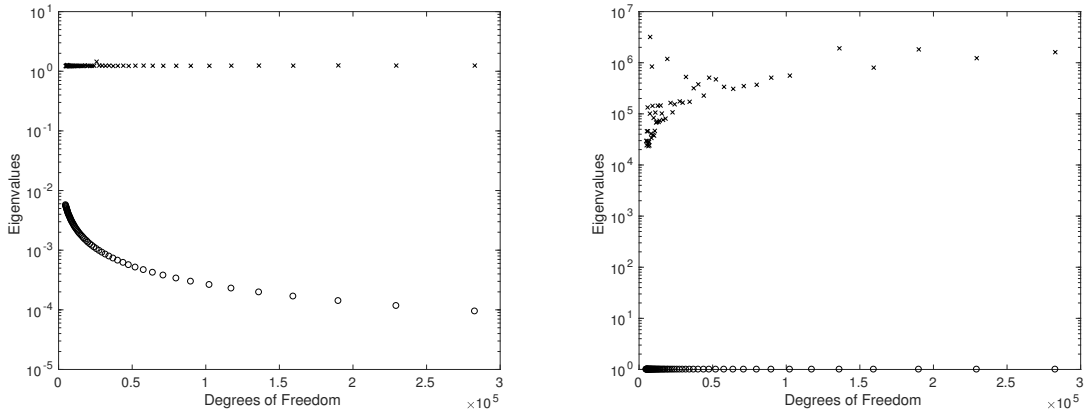


Figure 2.11: Smallest and largest eigenvalues (by absolute value) of the normalized matrix $\hat{A}$ (left) and the original matrix A (right) when refining the point cloud. Note the different scales.

converge faster when using the normalized matrices and the results in Section 3.3.2 confirm that.

Secondly, we are also interested in the eigenvalues for stretched domains. Here, we keep the smoothing length $h = 0.03$ fixed and focus on the normalized matrix, as this is the matrix with the smaller range of eigenvalues and we therefore expect it to yield better convergence rates.

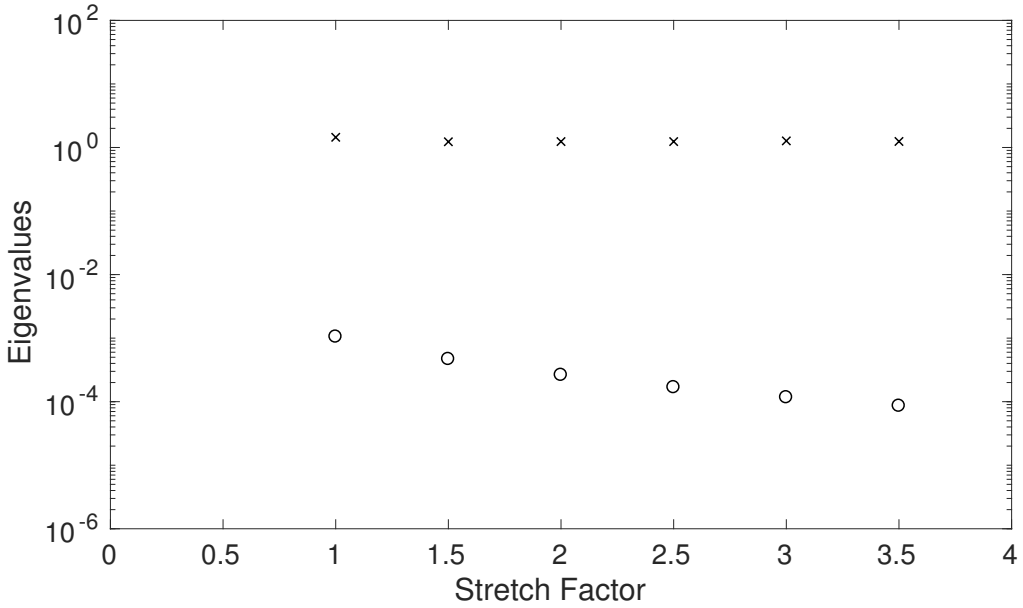


Figure 2.12: Smallest and largest eigenvalues (by absolute value) of the normalized matrix $\hat{A}$ when stretching the domain in both directions.

Our experiments for a domain stretched by a factor of 3.5, i.e. for $\Omega = [0, 3.5]^2$, yield a minimal eigenvalue that is about an order lower than for the original problem on $\Omega = [0, 1]^2$, see Figure 2.12. In Section 3.3.2, we will come back to this observation.

Diagonal Dominance

Assume that A is an essentially irreducible matrix, i.e. it has at least one row corresponding to a Dirichlet boundary condition and does not split into independent subsystems. For a physically well posed problem, this can be enforced by extracting an irreducible submatrix by using the method described in Section 5.2. In this case, the Poisson-like systems discretized by GFDMs are weakly diagonally dominant, i.e.

$$|a_{jj}| = \sum_{i=1, i\neq j}^{n} |a_{ij}|, \tag{2.72}$$

in those rows corresponding to interior points, if all the stencil values except for the central one, are negative: Because of the consistency conditions we apply for the discretization of the Laplace operator, we have a matrix row with a row sum of 0. If every coefficient except for the one on the diagonal is negative, then this implies weak diagonal dominance. For rows with mixed sign off-diagonal values, the diagonal dominance is violated. Again, this is because the row sum needs to be 0.

Those rows corresponding to Dirichlet boundary conditions are strictly diagonally dominant as all their off-diagonal entries are zero.

Diagonal Dominance is a desired property of the linear system as it drastically improves the convergence of linear solvers. It is therefore worth considering strategies that improve the diagonal dominance of the system. We already noted that the diagonal dominance in GFDMs is connected to the number of positive off-diagonal coefficients and therefore to the number of positive stencil values. Seibold [127] introduced a method that instead of using a circular neighborhood, used a minimization approach in order to select neighbors that ensured negative off-diagonal coefficients where possible[17]. This way, Seibold was able to obtain M-matrices for many problems. The drawback of his method though is that it requires additional effort in the point cloud organization phase and is not applicable to points that have boundary points in their neighborhood.
Another option, that is described by Suchde in [141], is to modify the least squares problem that defines the stencil for a given (circular) neighborhood. For example for the Laplacian, we can modify equation (2.10) by adding an additional condition for the central stencil value, obtaining

$$K_i^T \mathfrak{s}_i^\Delta = b^\Delta \tag{2.73}$$

$$c_{ii}^\Delta = \zeta. \tag{2.74}$$

[17] Note that Seibold works with negative diagonal coefficients and positive off-diagonal coefficients, which is why he uses the term “positive stencil”.

Changing ζ will not only change the central stencil value and therefore the coefficient on the diagonal, but also the other coefficients. Since the consistency condition for the derivation of constant functions implies that the sum of all coefficients in a stencil must be 0, a large positive value for ζ promotes negative off-diagonal coefficients. The optimal choice of ζ is situation-specific and Suchde ends up defining a stencil of the form

$$\mathfrak{s}_i^\Delta = \sigma_i^\Delta + \alpha^\Delta d_i^\Delta, \tag{2.75}$$

where σ_i^Δ satisfies the consistency conditions and d_i^Δ is in their null space. By using an optimized value for α^Δ, the number of weakly diagonally dominant rows can be increased significantly, see Section 4.5.4. It does not guarantee negative off-diagonal coefficients and therefore weak diagonal dominance though, as there are neighborhood configurations that do not permit a consistent discretization of the Laplacian without positive off-diagonal coefficients [127].

2.4.2 Linear Systems for the Velocity in the Segregated Approach

If the segregated approach in the FPM is used, the saddle point system can be avoided altogether at the cost of the drawbacks discussed in Section 2.3.3 and Section 2.3.4. In the segregated approach, the linear system of equations determining the predicted velocity is

$$\left(I - \frac{\Delta t}{\rho}\Psi_{\hat{\mu}}\right)(\tilde{\mathbf{v}}^{n+1}) = \mathbf{v}^n - \frac{\Delta t}{\rho}\tilde{\nabla}\tilde{p} + \Delta t \cdot \hat{g}, \tag{2.76}$$

see equation (2.46). Note that this is a system of equations for all three velocity components, therefore the matrix associated with this system has three times as many rows as the pressure systems for the same point cloud and time step. Depending on $\Psi_{\hat{\mu}}$ the three velocity components may or may not be coupled to each other. In the special case of an incompressible flow with constant viscosity, we have

$$\Psi_{\hat{\mu}}(\mathbf{v}) = \left(\operatorname{div}\left(\hat{\mu}\nabla\mathbf{v}^T\right)\right)^T, \tag{2.77}$$

and therefore we do not have any coupling between the velocity components in the interior of the domain. Depending on the type of boundary condition, the velocity components may still be coupled at the boundary. It is important to know if those couplings exist, because if they do it might be necessary to employ an *Alternate-Block-Factorization* to the matrix, see Section 3.3.5. If these couplings do not exist though, this computational effort is unnecessary and should be avoided.
For most applications in which the segregated approach is used, this system is diagonally dominant because Δt is small compared to $\hat{\mu}$. In applications involving water with free surfaces and splashes, for example, we have $\hat{\mu} \approx 1$ and $\Delta t \approx 10^{-4}$. Since this property means that those systems can be solved easily even with very basic iterative one-level methods, even if they have similar properties like the Poisson-like systems (Sections 2.4.1–2.4.1), we will not focus on the segregated velocity systems in this thesis.

2.4.3 Saddle Point Systems in the FPM

In Section 2.4.1, we examined the properties of the Poisson-like linear systems arising from the discrete versions of the pressure equations (2.51), (2.58) and (2.52). In Section 2.3.3 we also introduced a *coupled* approach though that gives rise to a coupled velocity-pressure system (2.61). These systems are saddle point systems [17] and in addition to that they still have the properties discussed in Section 2.4.1. Let us write the saddle point system as

$$\underbrace{\begin{pmatrix} \mathbf{A} & B \\ C & -D \end{pmatrix}}_{=:S} \begin{pmatrix} \mathbf{v} \\ p \end{pmatrix} = \begin{pmatrix} \mathbf{g} \\ f \end{pmatrix}, \tag{2.78}$$

where A represents the velocity-velocity couplings, $-D$ represents the pressure-pressure couplings and B and C represent those couplings linking the velocity and the pressure. In classical saddle point systems we would expect $B = C^T$, but the FPM yields $B \neq C^T$, see the saddle point system (2.61). The coupled approach in the FPM does not yield systems with $D = 0$ but rather $D \to 0$ as $\Delta t_{\text{virt}} \to 0$. As explained in Section 2.3.3 the ideal case would be $\Delta t_{\text{virt}} = 0$ which would imply $D = 0$. In practice we use $\Delta t_{\text{virt}} > 0$ to avoid numerical instabilities. Although $D \neq 0$ for the FPM case, the scales in A and D are still very different especially as $\Delta t_{\text{virt}} \to 0$. Note that the velocity-velocity subsystem A and the pressure-pressure subsystem $-D$ carry over their respective properties from the segregated approach. In particular, the three velocity components in A may or may not be coupled to each other depending on whether the viscosity is constant or not and on the type of boundary conditions. This will be important in Section 4.3.

Classical iterative schemes and also standard AMG approaches are not very efficient for saddle point systems because these systems are neither definite nor diagonally dominant. Multiple specialized methods for solving such systems have been developed by Benzi et al. [16] [17], Murphy et al. [100] and Bank et al. [15]. Since our main focus are AMG methods, we are especially interested in Uzawa-type methods [160]. The integration of the Uzawa method as a smoother in AMG has been studied for example by Adams [2] and later by Webster [168]. Our experiments in this thesis use the method proposed by Metsch [98] which will be described in more detail in Section 3.3.5.

Sparsity

Note that because the coupled velocity-pressure system includes equations for all three velocity components as well as for the correction pressure, there can be up to 160 non-zero coefficients in each row, when using 40 neighbors for each point in the point cloud. In the segregated case, there can only be up to 120 non-zero coefficients. In order to ensure the best possible computational efficiency, it is necessary to only include those coefficients in the matrix that are non-zero. In the case of incompressible flow with constant viscosity for example, we know that there are no couplings between the three

velocity components and therefore the corresponding coefficients in the matrix should not be present at all, rather than assigning them to 0.0[18].

2.4.4 Transient Structure of the Linear Systems

Apart from the static properties that each individual linear system in GFDMs has, there are some observations regarding the change of the linear systems within a time step and between time steps. Some of these observations are specific to the FPM, others may as well apply to other GFDMs.
Recall that in the FPM we either solve three pressure systems and one velocity system or two pressure systems and a coupled velocity-pressure system, depending on the approach that we choose to use. In the segregated approach the three pressure systems in a time step are identical, as long as the boundary conditions are consistent for all three systems (cf. Section 2.3.3). In the coupled approach, the same holds for the two remaining pressure systems. On the other hand, between solving any linear systems from two different time steps, there is a point cloud management phase which can, and in most cases will, change two important aspects influencing the structure of the linear system:

1. Points can be deleted or added to the point cloud. This means, that between two time steps the number of matrix rows does not have to be the same. It may increase or decrease.

2. Points move and so their neighborhoods can change. Therefore, points that were not adjacent to each other in time step T may be adjacent in time step $T + 1$. That means a coefficient in the matrix may change from $a_{ij} = 0$ to $a_{ij} \neq 0$ and vice versa, hence the couplings between two rows are not only variable in size across different time steps but they may also appear or disappear, changing the logical structure of the matrix.

Taking into account that different boundary conditions can be prescribed for the three or two pressure systems within the same time step (cf. Section 2.3.3), we must note that this change can also lead to a different matrix structure: If, for example, part of the boundary is changed from being a Dirichlet boundary condition for the pressure to being a mixed boundary condition, then the neighborhood of a point on that boundary will change from size one, just the point itself, to size 40 in the mixed case[19]. Again, this leads to a change in the logical matrix structure, which is especially important in Section 5.2.

The logical structure of the matrix to be solved as well as its size is important from the perspective of an AMG method, as AMG methods comprise two phases: A setup phase and a solution phase, cf. Section 3.3.3. One key point of using AMG effectively in the context of a transient simulation is to re-use the setup phase as often as possible, i.e. avoiding to construct a new setup for every matrix to be solved. This

[18] This description aims at the CSR matrix format [150] which we used in our implementation. In other data structures, these 0-coefficients may be unavoidable or less of a problem.
[19] In order to discretize the first order derivatives needed to represent Neumann boundary conditions.

becomes even more important when the solution phase is not much more expensive than the setup phase. As we will see in Section 4.5.6, the ratio between setup and solution phase in the FPM can be close to 50:50, meaning that by re-using one single setup for each linear system that needs to be solved, we could improve the overall speed of the linear solver by 2x. Of course, this would most likely not work even in classical discretization methods, as the matrix changes too much in order to use the same setup over and over again. In the FPM however almost the exact opposite is the case: Since even the logical structure of the matrix changes from time step to time step, we cannot re-use any setup between two time steps. What we can do though is to re-use the setup for hydrostatic pressure system in the other one (coupled approach) or two (segregated approach) pressure systems, as long as the boundary conditions are unchanged. Possible strategies for a setup re-use across time steps will be discussed in Section 4.4.

Remark 2.16. Although here and in the rest of the thesis we focus on GFDMs, we note that many of the properties discussed here are also true in other meshfree methods. For example, both the *Implicit Incompressible SPH* method [64] and the (also implicit) *SPH projection* approach [31] lead to non-symmetry matrices, if the density is non-constant (like in multi-phase simulation) or the particle distribution is non-uniform. Similarly to the issue of randomly enumerated points in the FPM point cloud carries over to the particles in SPH methods. This suggests that the new AMG method we introduce in Chapter 4 and the following chapters is also an efficient linear solver for such SPH methods.

Chapter 3

Applying State of the Art Linear Solvers to GFDM Matrices

Chapter 2 gave an overview of linear systems that arise from GFDMs. Since this thesis focuses on the industrial application of the FPM, many of the simulations we are dealing with use a large number of points, leading to large linear systems that need to be solved. These large systems pose a problem to classical linear solvers as their run-time would take up a large amount of the overall simulation run-time, therefore making the simulation unfeasible.

There are two main classes of methods to solve linear systems of equations: iterative methods and direct methods. Direct methods are often the simplest to use linear solvers, but their application is limited to fairly small problems, because they need around $\mathcal{O}(n^3)$ operations, where n is the number of rows in the linear system. This means that for almost all industrial applications the computational cost for solving a linear system using a direct solver is too high.

Direct solvers compute a solution that satisfies the linear system within machine precision. Iterative linear solvers on the other hand compute a series of approximations to the solution. This gives us the freedom to stop the process when the current approximation is *good enough* for our needs. Saad [119] gives a comprehensive overview of the wide class of iterative methods. A common observation for many iterative solvers, namely *one-level iterative linear solvers*, is that their convergence behavior depends on the discretization size h: This means that the number of iterations needed to reach the same level of accuracy of the approximation increases when refining the discretization. Because the computational effort per iteration also increases with the refinement, this means that the overall cost for these solvers increases with $\mathcal{O}(n^k \log \epsilon)$, where $k > 1$ and ϵ is the desired accuracy of the approximation. Again, we expect industrial applications to be quite refined, at least in certain areas of interest. Therefore, the one-level iterative schemes are not optimal in this context. Additionally, saddle point problems as they arise in the coupled velocity-pressure systems in the FPM require special treatment and not all one-level iterative solvers can cope with these systems.

In order to overcome the h-dependency of the one-level solvers, Multigrid methods have been developed. They provide an h-independent convergence rate, making the overall method scale linearly with the number of matrix rows, i.e. the computational

cost is $\mathcal{O}(n \log \epsilon)$. Multigrid methods achieve this by a carefully tuned interplay between what is called *smoothing* and *coarse grid correction*. Note that they are usually used as a preconditioner for a classical one-level iterative solver rather than standalone.
The first Multigrid methods were based on *geometrically* finding coarser discretizations based on the fine level discretization. Those coarser discretizations were referred to as *grids*, hence the name *Geometric Multigrid* method. We will give a geometrically motivated Multigrid method in Section 3.3.2 of this chapter that shows how the idea of Geometric Multigrid can be carried over to GFDMs.

Although Geometric Multigrid methods already show desirable properties in terms of their scaling behavior, they introduce the need to create and deal with multiple *grids*. We therefore take a look at so-called *Algebraic Multigrid methods* which construct a hierarchy of *levels* based on the initial linear system only. These methods have the same linear scaling property as their geometric counterparts, but do not require any additional geometric information. We introduce Algebraic Multigrid methods in Section 3.3.3 and their modifications used to solve saddle point systems. It is however not clear how these methods can be applied efficiently to the matrices arising in GFDMs. We will point out some of the difficulties in Section 3.3.4.
The chapter concludes with some numerical experiments showing that these methods work on some of the model problems that we introduced in Chapter 2.

Method	# operations in 2D
Gaussian Elimination [137]	$\mathcal{O}(N^3)$
Gaussian Elimination (band version) [109]	$\mathcal{O}(N^2)$
Jacobi / Gauss-Seidel iteration [119] [42]	$\mathcal{O}(N^2 \log \epsilon)$
Conjugate Gradient [58]	$\mathcal{O}(N^{3/2} \log \epsilon)$
Multigrid	$\mathcal{O}(N \log \epsilon)$

Figure 3.1: Some classical linear solvers. N is the number of rows in the matrix and ϵ is the desired accuracy of the solution. Excerpt from [156]. The methods listed here are very well known so the references given are just commonly used examples.

Figure 3.1 gives an overview of the methods discussed here together with their asymptotic complexities for a 2D Poisson problem[1] . A more extensive version of this table can be found in [156]. Figure 3.2 shows run-times for some linear solvers based on the FPM's hydrostatic pressure system from a 3D model of a flow through a unit cube. The importance of an efficient linear solver becomes clear when looking

[1]Hierarchical matrices ($\mathcal{H}$-matrices) as introduced by Hackbusch in [53] are not considered in this thesis, although they can be used to solve certain types of linear systems as well by approximating the inverse of system matrices [19] [38]. In general, the computational cost for this approximation is $\mathcal{O}(N \log^2 N k^2)$ [19], where k is a parameter influencing the quality of the approximation. They can also be used as a preconditioner (for a Krylov subspace method for example) by computing an $\mathcal{H} - LU$ decomposition [53]. The question how and how well the inverses of the matrices in this thesis can be approximated by $\mathcal{H}$-matrices is beyond the scope of this thesis.

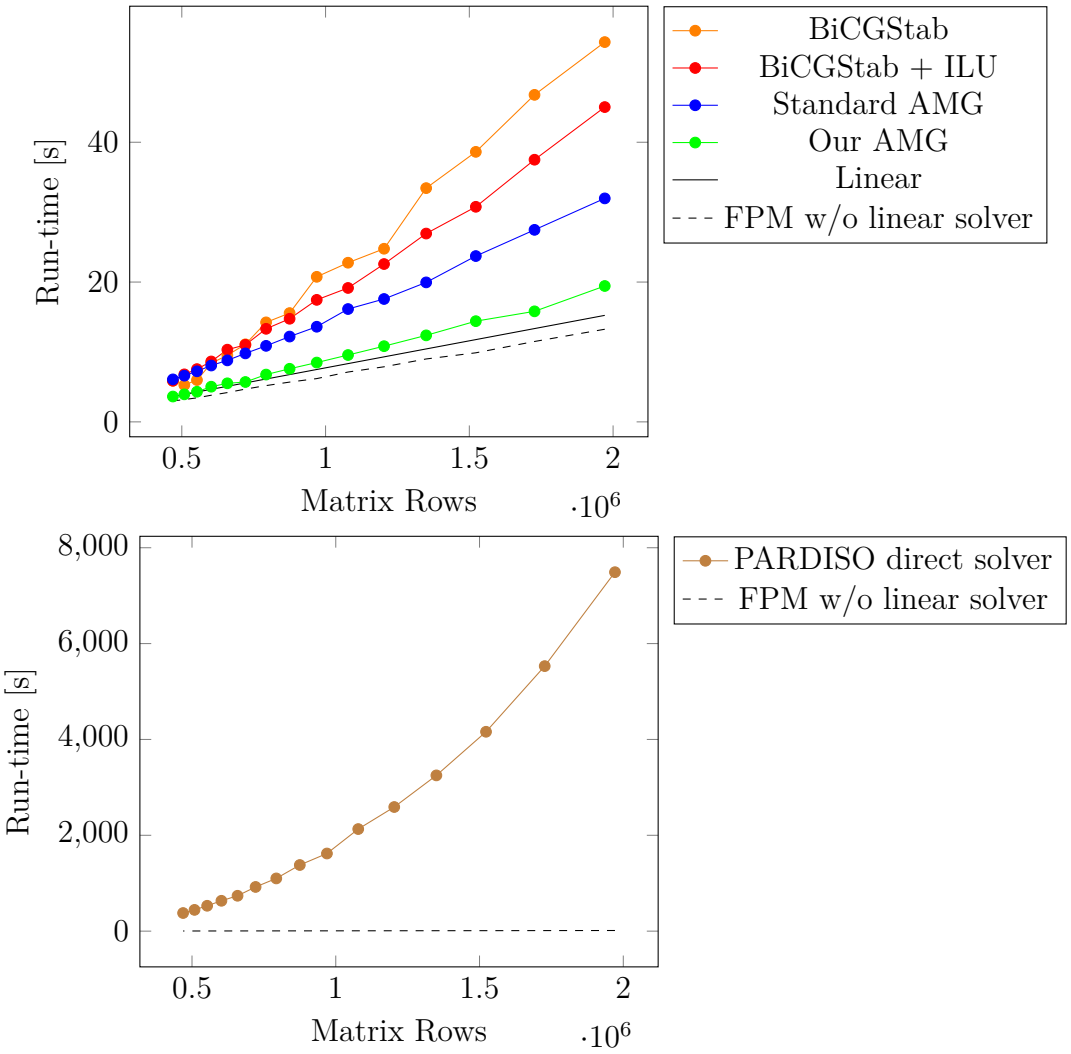


Figure 3.2: Run-times for different solvers for a 3D Poisson problem on a unit cube using the FPM with different numbers of points. "Standard AMG" refers to standard Algebraic Multigrid as introduced in [138] and "Our AMG" is the Algebraic Multigrid method we develop in Chapter 4. For comparison, the run-time of the FPM without the linear solver is plotted as a dashed line.

at the solver run-times for those simulations with many degrees of freedom, i.e. small smoothing length. Our AMG method, which we will explain in Chapter 4, is the only method that scales almost perfectly linear with the degrees of freedom. It is therefore the only usable option for industrial size applications, where run-time matters. Even the standard AMG method does not perform very well, which is why our newly developed method is needed. For comparison, Figure 3.2 also gives the FPM run-time outside the linear solver. Observe how the linear solver is the main bottleneck in this simulation.

With the insight gained in this chapter we will then build a new efficient Algebraic Multigrid method for GFDMs in Chapter 4. In Chapter 5, we will look into its parallelization and robustness.

3.1 Direct Linear Solvers

Direct linear solvers are seldom used nowadays, but we want to point out some of their features as they will also be one of the components that form the AMG algorithm. Unlike iterative solvers, direct solvers do not approximate the solution of a linear system over multiple iterations, but they find a solution in one single step[2].
They also have the advantage of being almost independent of any specific matrix structure or any specific matrix properties: They can be applied to almost any linear system. The system does not even have to be sparse, although sparsity, and more importantly a small band-width, can lead to shorter run-times.
The main drawbacks of direct solvers are computational effort and memory usage though. Gauss Elimination for example has a computational complexity o $\mathcal{O}(n^3)$ and uses $\mathcal{O}(n^2)$ memory, as it also needs to store zeros because they might become non-zeros later in the algorithm. This means that the full $n \times n$ matrix needs to be stored. Note however that the computational complexity can be reduced to $\mathcal{O}(n^{\log_2 7})$ by using a more sophisticated formulation of the Gauss Elimination algorithm [137]. In general, even modern direct linear solvers have a computational complexity of around $\mathcal{O}(n^3)$ making them rather unfeasible to use as a stand-alone linear solver for industrial size application, where $n >> 100000$.
Modern implementations of direct solvers exhibit the computer architecture quite well though and try to minimize the constant factor hidden in their $\mathcal{O}(n^3)$ complexity. One example of such implementations is Intel's PARDISO which is part of Intel's MKL [167].
Another example for a direct solver that we will encounter later in this thesis is the LU-decomposition [13] which is equivalent to Gauss Elimination but where the factorization $A = LU$ is stored which reduces solving the linear system $Ax = f$ to solving

$$Ly = f \tag{3.1}$$

and then

$$Ux = y. \tag{3.2}$$

[2]Up to machine precision.

Since L and U are triangular matrices, these two equations can be solved easily by forward and backward substitution. The main computational work, which is again $\mathcal{O}(n^3)$, is in finding the decomposition $A = LU$[3]. Since the decomposition is independent of the right-hand side f though, we can use the same decomposition to solve $Ax = f$ for multiple right-hand sides, which is a feature that we will be able to make use of in our AMG method, see Chapter 4.

3.2 Iterative One-Level Linear Solvers

This section will very briefly familiarize the reader with some commonly used non-hierarchical (*one-level*) iterative methods for the solution of sparse linear systems. First we will introduce the very basic relaxation schemes and make some comments regarding their convergence. These schemes will also be one part of the multigrid algorithms that we will use throughout this thesis.
Secondly, we will focus on Krylov subspace methods. These methods, when combined with a *preconditioning method*, are often found in software tools to solve the linear systems. Again, we make some comments about their convergence. The Multigrid methods that we will introduce in the next section are usually employed as a preconditioning method for Krylov subspace methods, so Krylov subspace methods also play a role in the multigrid context.

3.2.1 Relaxation Schemes

Two relaxation schemes are important in the context of Multigrid methods: The *Jacobi* method and the *Gauss-Seidel* method. Both methods are well-known and can be found in [119], for example. We will not introduce these methods in detail here. We do want to note though that the Gauss-Seidel method cannot be parallelized without deviating from the original formulation. Adams et al. [1] studied the Gauss-Seidel algorithm in parallel in the context of Multigrid methods.

Convergence

As pointed out in Section 2.4.1 the matrices of interest for this thesis are not necessarily diagonally dominant, but because of the modifications to the least squares problem described by Suchde [141], many of the rows are at least weakly diagonally dominant, cf. Section 4.5.4. Diagonal dominance is advantageous for the convergence of many iterative linear solvers and also for the relaxation schemes discussed here. It is also well-known though that the convergence rate of relaxation methods highly depends on the smallest eigenvalue of the matrix A, which in turn depends on the discretization size h, see for example [80] Section 4.2.1. This leads to an increased amount of iterations needed to solve a linear system corresponding to a finer discretization for a fixed accuracy. Since the cost of each iteration increases linearly with the size of the matrix, simply because the computations need to be carried out for a larger number of rows,

[3] In practice it is more common to find a decomposition $PA = LU$ where P is a permutation matrix.

the cost of the overall method increases more than linearly.
For this very reason, relaxation schemes are usually not employed as a stand-alone solver but as a *smoother* in the context of Multigrid methods or as a preconditioning method in the context of Krylov subspace methods.

3.2.2 Krylov Subspace Methods

Krylov subspace methods are among the most important techniques for solving large, sparse linear systems of equations [119]. They are based on projections onto so called *Krylov subspaces* which are spanned by vectors $p(A)v$, where p is a polynomial. They are more sophisticated to implement compared to the relaxation schemes in the previous section, but they offer better convergence behavior in most cases. There is a variety of different Krylov subspace methods for different purposes and different types of matrices, for example non-symmetric matrices. Saad [119] gives an extensive overview.
Note that Krylov subspace methods are usually used together with a preconditioning method, i.e. instead of solving $Au = f$, the modified system

$$M^{-1}Au = M^{-1}f \tag{3.3}$$

is solved. The preconditioning operator M^{-1} should be easy to compute and easy to apply to a given vector and at the same time it should approximate the exact inverse of A:

$$M^{-1}A \approx I. \tag{3.4}$$

The goal is to decrease the number of Krylov subspace iterations needed to solve the preconditioned system (3.3) compared to the original system.
Our main focus is the family of conjugate gradient algorithms as they are commonly used with multigrid preconditioning [73] and also because the BiCGStab2 algorithm served as our initial benchmark when evaluating the performance of Algebraic Multigrid in the context of GFDMs. The basic conjugate gradient (CG) algorithm is described in [119] Section 6.7 but is only applicable for symmetric matrices, which is not the case in GFDM matrices, as discussed earlier. Hence the BiCGStab algorithm was developed by Van der Vorst [161], which was later generalized by Gutknecht [49]. The most general form is BiCGStab(l) [132] which is equivalent to BiCGStab and BiCGStab2 for $l = 1$ and $l = 2$ respectively, given that all operations are carried out in exact arithmetic.

Convergence

For the CG method for symmetric matrices, there exists a convergence estimation of the form

$$\|u^* - u^m\|_A \leq 2\left[\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right]^m \|u^* - u^0\|_A, \tag{3.5}$$

where m is the current iteration and

$$\kappa = \kappa(A) = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)} \tag{3.6}$$

is the condition number of A. Here we assume A to be symmetric which is why the second equality holds.
For non-symmetric matrices, we need to employ the BiCGStab(l) algorithms for which no theoretical convergence estimates are known. Since the BiCGStab algorithm is built based on two CG steps and an additional stabilization and the BiCGStab(l) algorithm essentially enhances this stabilization, specifically for problems with large complex eigenpairs [132], it is not surprising though that experimentally it can still be seen that $\kappa(A)$ is the key indicator also for the convergence of BiCGStab(l). For our GFDM matrices that means when refining the discretization, we still expect an increase in computational cost for solving the linear systems that is more than linear. The experiments in Section 3.3.2 confirm this expectation. Although preconditioning can improve the convergence dramatically, the dependency on the discretization size persists.

Preconditioning by Incomplete LU-Decomposition (ILU)

The incomplete LU-decomposition was first mentioned by Varga [165] and examined by Meijerink et al. [96] and is based on the LU-decomposition mentioned in Section 3.1. For the ILU method, we do not find an exact splitting $A = LU$ but only an approximation

$$A \approx LU. \tag{3.7}$$

There are multiple variants for defining this approximation, for example ILU(p) [72] and ILUT [118]. Using ILU as a preconditioning technique for CG methods is computationally more expensive per iteration compared to Jacobi or Gauss-Seidel preconditioning, but it can have a much better impact on the convergence of the CG method. ILU methods are also used as a smoother in the context of Multigrid methods [73] [140], although their properties are quite different from those of the Gauss-Seidel or Jacobi method. We will be using ILU smoothing in our Algebraic Multigrid method for the coupled velocity-pressure systems as well, at least in certain situations.

3.2.3 Application to the Segregated Velocity Systems

As already mentioned in Section 2.4.2, the segregated velocity systems are diagonally dominant, which makes them solvable for both relaxation schemes and (preconditioned) Krylov subspace methods in typically very few iterations. In both cases, the convergence rate is mainly dominated by the smallest eigenvalue $\lambda_{\min}(A)$ which, for diagonally dominant matrices A, can be bounded away from zero using Gershgorin discs [43].
In Section 4.5.1 we show that owing to the small number of BiCGStab2 iterations needed to solve these linear systems with a simple Jacobi preconditioning, it is not advisable to employ a sophisticated multigrid preconditioning in terms of overall computational cost.

3.3 Multigrid Solvers

3.3.1 Geometric Multigrid

In this section we will give the basic multigrid ideas and specifically go into the Geometric Multigrid algorithm that will serve as our basis for the algebraic version in the next section. Some ideas of Geometric Multigrid have also been incorporated in the new method introduced in Section 3.3.2.
The observation that leads to multigrid methods is that classical iteration schemes like Gauss-Seidel reduce certain components of the error quite well, namely the high frequency ones, but have trouble solving for some others, the lower frequency ones [138]. Multigrid methods were originally developed for sparse linear systems arising from the discretization of elliptic PDEs. The basic idea of any multigrid algorithm is to use a hierarchy of linear systems whereas the original problem is the finest level in the hierarchy, and therefore the one with the largest associated matrix, and the coarsest level in the hierarchy is small enough to be solved with a direct solver[4]. The key here is that the low frequency error components on each level are well represented on the next coarser level but are high frequency components there. Therefore, on the coarser level, they can be solved by a standard iterative scheme. This idea can be applied recursively, until the problem is small enough to be solved with a direct solver. Two main components need to work together in a Multigrid method: The *smoothing*, i.e. the reduction of the high frequency error components, and the *coarse grid correction*, which refers to transferring the low frequency error components down to coarser levels, solving for them there and interpolating a correction back to the fine level. These two components can be coupled in multiple fashions: either in the form of a V- or W-Cycle [138] or, in special applications, as a k-Cycle, for example [106].
These ideas were first developed by Hackbusch [50] and Brandt [21] and we refer to the ample literature in this field for further details. Trottenberg et al. [156] give a quite comprehensive overview.

Unlike one-level iterative solvers or direct solvers, Multigrid methods scale linearly in the number of matrix rows, i.e. their computational complexity is $\mathcal{O}(n \log \epsilon)$, where ϵ is the desired relative residual reduction. This means that when refining a discretization for a physical problem and, say, doubling the number of degrees of freedom (and therefore rows in the matrix), the time needed to solve this linear system also doubles. This is because in Multigrid methods the number of iterations needed to solve the linear system stays constant, unlike in one-level methods. The extra computational complexity purely originates in the extra operations that are needed in every iteration and in the setup phase in the algebraic case.
Due to the hierarchical approach, the memory consumption is around two times the size of the original matrix, if the hierarchy is created in a suitable fashion. That means that storing the hierarchy, which is a *series* of smaller matrices, needs about as much memory as the original matrix.
Because of their linear scalability, Multigrid methods can be used to solve large prob-

[4] Although iterative methods can also be employed on the coarsest level, the original idea is to use a direct solver here.

lems as well. In practice their most common use is as a preconditioner for CG methods, which reduces the number of iterations needed to reach the desired accuracy.

In the case of Geometric Multigrid methods, we assume that the hierarchy of linear systems is provided externally, for example through providing a series of grids which go from a finest grid to a coarsest grid on which the same discretized differential operator can be applied, leading to matrices of different sizes that all discretize the same problem. This is a problem, as for non-standard discretizations it is not always clear what a coarser discretization should be that is suitable for the Geometric Multigrid hierarchy. Note that when those methods were developed, Finite Difference Methods on regular grids were the most commonly used discretization techniques and in this case, a coarser discretization can be generated naturally by moving from a discretization size h to $2h$ or even $4h$ for example.

3.3.2 A Geometric Multicloud Approach

This section introduces a new iterative solver that is based on a hierarchy of point clouds and is related to Geometric Multigrid methods. The method is fairly limited in terms of the variety of problems it can solve and it is implemented in MATLAB only, but it may well serve as a means of motivating the use of Algebraic Multigrid methods for GFDMs.

We describe the construction of all of the multigrid operators and finish the section with some results produced by the method, comparing them to results with our AMG method (cf. Chapters 4 and 5) on the same matrices. Lastly, we discuss the drawbacks of this method, motivating the use of AMG.

Motivation

The simple structure of a point cloud with no inherent neighboring relationships presents itself to using a geometric idea of creating coarse levels, namely by using coarser point clouds, rather than using the matrix entries. To generate coarse point clouds, there are two options:

- Generate multiple, independent point clouds with different smoothing lengths h or
- start with an existing, fine, point cloud and select a subset of the points to form a coarse point cloud.

The required interpolation and restriction operators between the levels are easier to determine when the coarse levels are subsets of the finer ones, so for our method we chose the latter option. The coarsening can be done based on geometric neighborhood information only and no information on the stencils computed on the fine point cloud is needed. Therefore, the information on which we are basing our coarsening, namely the distance between two points, is symmetric, as opposed to the size of the matrix entries a_{ij} which are not necessarily symmetric in our case.

Remark 3.1. In other meshfree methods, various strategies have been found in order to incorporate the idea of (non-algebraic) Multigrid methods: The coarsening idea presented here is similar to the idea of a *Tree Partition of Unity Method* presented in [123], but works the opposite way: Instead of starting with a coarse point cloud and *refining* the point cloud successively, we are starting off with a fine point cloud that we will *coarsen*. Takahashi et al. [147] use a background mesh in order to use a Multigrid method in their implicit SPH method. Katz and Jameson [71] focus on the aspect of constructing a "meshless coarse level operator" using radial basis functions for non-regular grids in an Eulerian framework. Their idea of coarsening a point cloud is very similar to the Multicloud approach presented here.

Coarsening Strategies Using Geometric Information

The coarsening strategy of the method we are proposing is based on the assumption that the solution of the linear system has similar values at points that are geometrically close to each other, which is the case especially in the pressure systems that need to be solved in the FPM as long as the density ρ is constant or at least does not have any jumps.

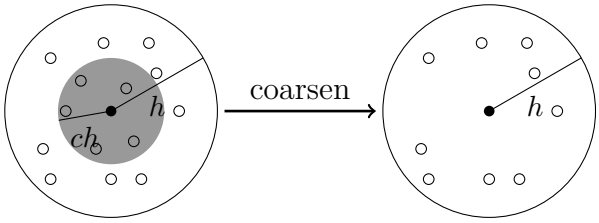


Figure 3.3: Local view of the geometric coarsening strategy: Points that are close to the central point (black) are not used to form the coarse point cloud.

For a given fine point cloud $\mathcal{P}_h$ we start by choosing one point x_i as a *coarse level point*. Then all points that are within a distance of ch ($c < 1$) of x_i become *fine level points*, meaning that they will not be present in the coarse point cloud $\mathcal{P}_H$, see Figure 3.3. Note that we do not need to recompute any distances here, as they are already known from the operator setup, see Section 2.2.4.

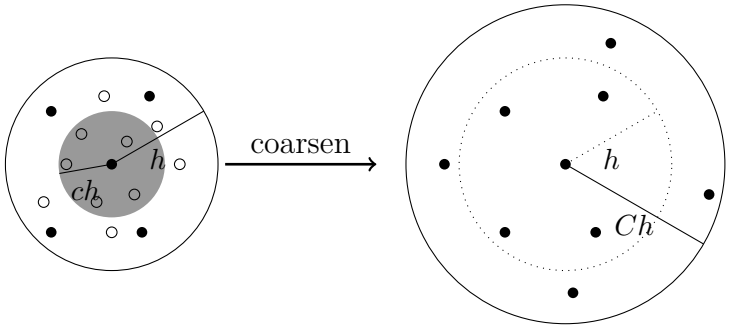


Figure 3.4: After applying the coarsening strategy at every point in the fine level point cloud, the points labeled *coarse* (black) form the coarse level point cloud.

We repeat this procedure with every point that has not been assigned a label *coarse* or *fine*, yet. All points that have been assigned the *coarse* label form our coarse point cloud. Figure 3.4 shows how this procedure affects the neighborhood of a central point x_i. The resulting point cloud depends on the order in which we assign the *coarse* label to the points in the fine point cloud.
However, because we want to use the coarse point cloud to construct the coarse operator — see Section 3.3.2 — we need to make sure that the coarse point cloud is a proper discretization of the original problem. In particular, this means that there need to be boundary points in the coarse point cloud as well. The easiest way to make sure that not all of the boundary points are marked as fine level points is to start by applying the coarsening algorithm at the boundary points and only start marking interior points as coarse level points when all of the boundary points have been marked as fine or coarse level points. Figure 3.4 also shows that we need to reconsider what the coarse level neighborhood of the central point x_i is. Because we do not take the points near the central point to the coarser level, there are less neighbors within the old neighborhood radius h. If we would define the neighborhood on the coarse level with the same radius h that we use on the fine level, the situation depicted in Figure 3.4 would cause issues: The way the coarse level operator is constructed (see Section 3.3.2) makes it mandatory for the coarse point cloud to allow the construction of an operator by the same method that was used to construct the fine level operator. In the situation depicted here, however, the central point x_i would not have a neighbor to the right, which would make the construction of a second order stencil impossible. Recall from Section 2.2.4 and also from Section 2.4.1 that

- making sure the points are distributed in such a way that the construction of a second order stencil is always possible is not straightforward and would also be too expensive in terms of run-time and that
- The FPM assumes that with 20 (2D) or 40 (3D) neighbors there is usually enough neighbors to construct second order stencils.

We stick with this assumption and therefore need to make sure that the number of neighbors in the coarse point cloud is about the same as in the fine point cloud. By using $H = Ch$ $(C > 1)$ as our new neighborhood radius and assuming that the points are uniformly distributed in the domain and assuming that the order in which we assign the *coarse* label does not affect this uniformness,

$$C = \sqrt{\frac{n_{\text{coarse}} + n_{\text{fine}}}{n_{\text{coarse}}}} \tag{3.8}$$

is sufficient to keep the average number of neighbors constant. Here, n_{coarse} and n_{fine} is the global number of coarse level and fine level points, respectively. Note that these are two disjoint sets and their union form the fine level point cloud. The parameter $c < 1$ can be used to determine the coarsening rate of the method. A value close to one means that more points in the neighborhood of a coarse level point will become fine level points, therefore there will be less points on the coarse level. A small value of c on the other hand will mean that less points will be declared fine level points.

Coarse Grid Operator

After having coarsened the point cloud, we can now construct a coarse grid operator on this point cloud. Instead of using a Galerkin operator like in AMG, we construct the coarse grid operator by applying the same scheme from Section 2.2.4 that was used on the fine point cloud to the coarse point cloud. This gives us a coarse grid operator discretizing the same continuous problem as the original one, but with a coarser mesh size and therefore a smaller matrix. As in all Multigrid methods, we can either solve the coarse grid problem directly or use the same idea of coarsening recursively.

Restriction

Restriction is done based on the spatial distance between two points. The idea is that every coarse level point x_i has a coarse level residual R_i that is a weighted sum of the fine level residual r_i at the coarse point itself and the fine level residual at the fine level points in its neighborhood. Here, the weights 1/4 and 3/4 are loosely based on the idea of *full weighting* in Geometric Multigrid methods [156]:

$$R_i = \frac{1}{4} r_i + \frac{3}{4} \sum_{j \in N_i^F} \alpha_{ij} r_j, \tag{3.9}$$

where N_i^F is the subset of neighbors of x_i that are fine level points and

$$\alpha_{ij} = \frac{\hat{d}_{ji}}{\sum_{l \in N_j^C} \hat{d}_{jl}} \quad \text{with} \quad \hat{d}_{ij} = ||x_i - x_j||^{-1} \tag{3.10}$$

are weights. Note that the definition of α_{ij} depends on the *coarse* level neighbors N_j^C of x_i rather than on the neighbors of x_j. It ensures that all weights going out from a fine level point sum up to 1.
Special considerations are needed at points that are neighbors of boundary points. Because of the different scale at boundary points (cf. Section 2.3.6), we cannot restrict the residual to the boundary as easily. Therefore, while we do take boundary points into account when computing the denominator of the weights in (3.10), we reset all weights to 0 that refer to a boundary point, i.e. we have

$$\alpha_{ij} = \begin{cases} 0 & \text{if } x_j \in \partial\Omega \\ \frac{\hat{d}_{ji}}{\sum_{l \in N_j^C} \hat{d}_{jl}} & \text{else} \end{cases}. \tag{3.11}$$

We then make sure that the sum of all weights coming in to a specific coarse level point x_i is one. Otherwise, the correction we get from our coarse grid correction has a wrong scale. This is achieved by simply row-scaling the restriction operator so that all row sums equal 1.

Interpolation

The setup of the interpolation operator is very similar to that of the restriction operator. For every fine level point x_k the interpolated correction from the coarse level

is

$$e_k = \sum_{i \in N_k^C} \gamma_{ik} E_i \quad \text{with} \quad \gamma_{ik} = \frac{\hat{d}_{ik}}{\sum_{j \in N_k^C} \hat{d}_{jk}}, \tag{3.12}$$

where N_k^C denotes the subset of neighbors of x_k that are coarse level points and E_i is the correction computed on the coarse level at x_i. As opposed to the restriction, the interpolation weights γ_{ik} depend on the neighborhood of x_k only. For coarse level points x_k we set $e_k = E_k$.
Similarly to what we do in the restriction, at the boundary we reset all weights that refer to interpolations from the boundary to the interior of the domain to 0. As we are also cutting off the restriction to the boundary, this means that we have decoupled the interior from the boundary in both transfer operators. Since we construct the coarse operator by computing new stencils on the coarse point cloud though, there are still couplings between the interior and the boundary in the coarse operator. Therefore, the correction that is computed using the coarse operator will take effects of the boundary conditions into account. The results in Section 3.3.2 show that our approach works well in the case of Dirichlet boundary conditions. However, note that these results are limited to Dirichlet boundary conditions as they can be solved for using the smoother on the finest level.

Numerical Experiments

Let us evaluate different linear solvers using the Poisson model problem (2.71) with

$$f(x, y) = -8\pi^2 \sin(2\pi x) \sin(2\pi y) \,, \tag{3.13}$$

$$g(x, y) = 0. \tag{3.14}$$

The stopping criterion for all linear solvers is a relative reduction of the initial residual by ten orders of magnitude. For both AMG and our Multicloud approach we use Gauss-Seidel relaxation. In the Multicloud approach, we use $c = 0.4$ as the coarsening parameter which leads to an average coarsening rate of about 1/10. The Multicloud approach is applied to the original matrix A, rather than a scaled version, in all experiments, because scaling does not matter in this approach. The AMG implementation we are comparing our Multicloud approach against is based on the work of Stüben [138], cf. Appendix B. Like in Section 2.4.1, the point clouds are generated with a normal distribution, additionally making sure that the minimal distance between any two points is greater than $0.1h$.

Refining the Point Cloud. In the first experiment, we keep the domain $\Omega = [0, 1]^2$ fixed and use point clouds of different density, i.e. we vary the smoothing length h. The resulting point clouds have between 4876 and 913,520 points. Figure 3.5 shows how the number of iterations the one-level solver BiCGStab needs in order to converge increases with the problem size. This behavior is typical for a one-level method. Our Multicloud approach on the other hand is a Multigrid method and benefits from using the hierarchy of multiple point clouds. As we would expect from a Multigrid method, the approach needs an almost constant amount of iterations for all problems: The number of iterations only varies between 10 and 13 for the Multicloud approach but

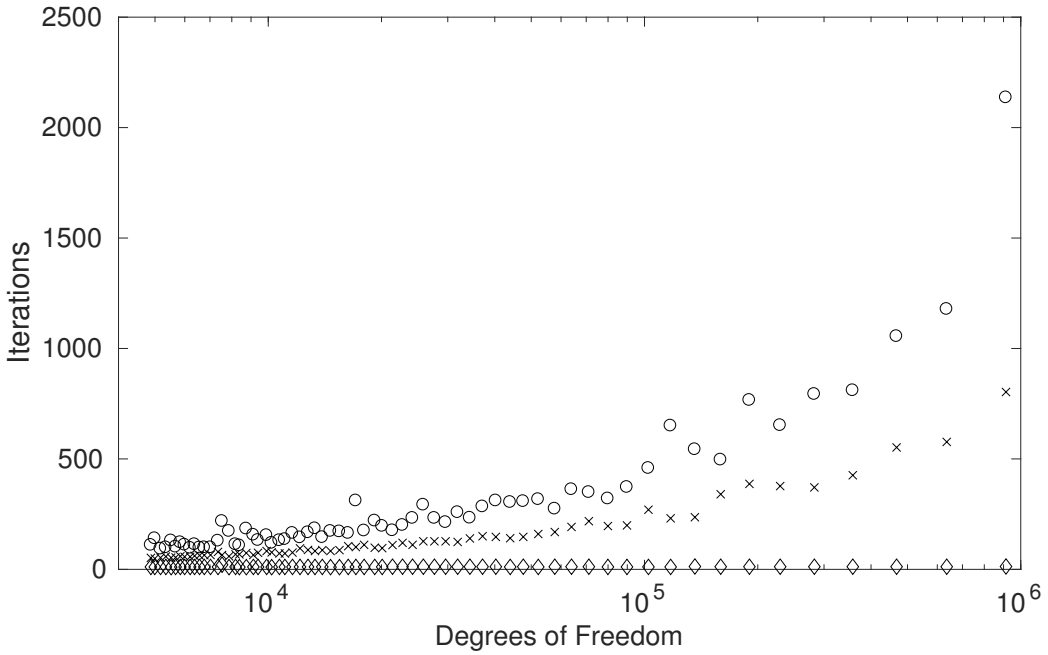


Figure 3.5: Number of iterations needed to converge when decreasing h and therefore increasing the number of degrees of freedom. $\circ$: BiCGStab on the original matrix; $\times$: BiCGStab on the normalized matrix; $\diamond$: Geometric Multicloud approach.

between 93 and 2,136 for BiCGStab on the original matrix. When normalizing the matrix, which decreases the range of eigenvalues as Section 2.4.1 shows, BiCGStab needs less iterations, however the number of iterations needed still increases significantly when refining the point cloud.

Stretching the Domain. Next, we consider the model problem (2.71) with a fixed smoothing length $h = 0.03$ on a varying domain, namely $\Omega = [0, L_1] \times [0, 1]$, $L_1 = 1, \ldots, 20$.

This stretching of the domain does not have a significant influence on the number of iterations needed by BiCGStab or our Multicloud method, as is shown in Figure 3.6. If we stretch the domain in both directions, i.e. consider the domain $\Omega = [0, L_1] \times [0, L_2]$, $L_1, L_2 = 1, \ldots, 5$, this observation changes. Figure 3.7 shows that in this case the number of iterations needed by the one-level BiCGStab method increases significantly when the domain gets larger. With 11 and 12 iterations the Multicloud approach again has an almost constant amount of iterations. The increase in iterations of the BiCGStab solver is more significant than the increased range in eigenvalues in Figure 2.12 would indicate, which shows that the range of eigenvalues is not the only factor that determines the convergence rate of BiCGStab. Heuristically we have found that generally BiCGStab converges more slowly for long and thin domains. More specifically, the longer the shortest path between any two Dirichlet boundary points along neighboring points in the point cloud becomes (cf. Section 5.2, Definition 5.3), the worse the convergence rate of any one-level method gets. Normalizing the matrix

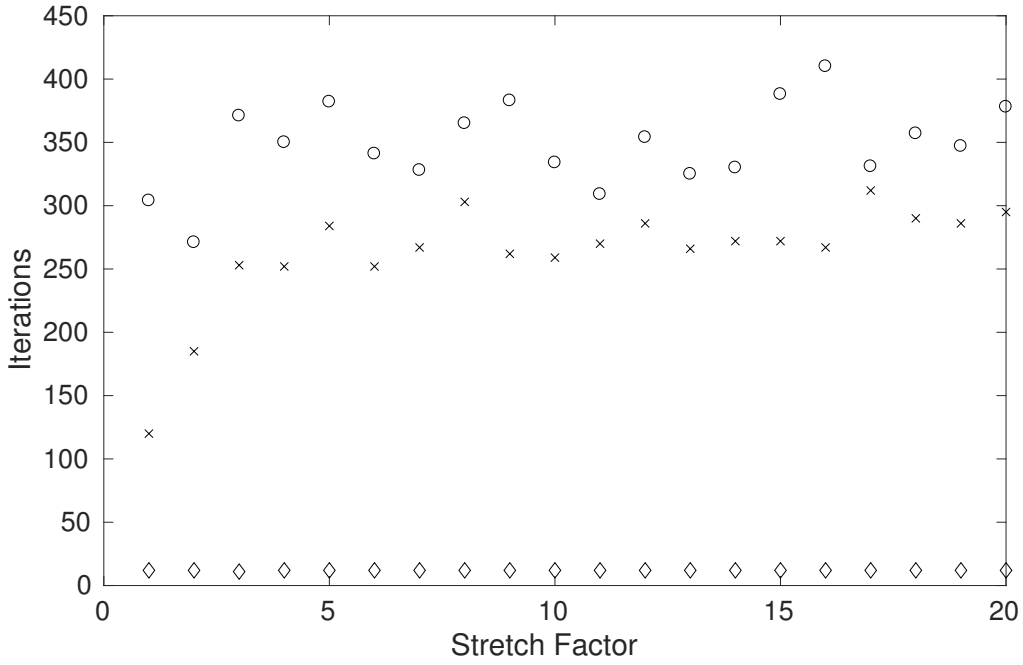


Figure 3.6: Number of iterations needed to converge when stretching the domain in the x-direction only. ○: BiCGStab on the original matrix; ×: BiCGStab on the normalized matrix; ◇: Geometric Multicloud approach. $h = 0.03$.

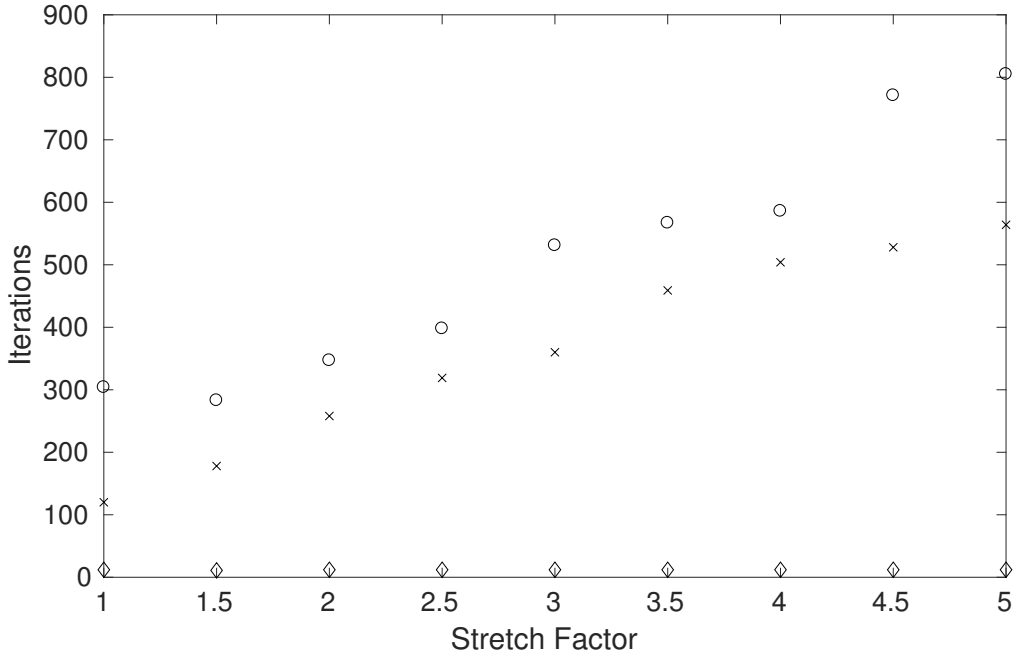


Figure 3.7: Number of iterations needed to converge when stretching the domain in both x- and y-direction. ○: BiCGStab on the original matrix; ×: BiCGStab on the normalized matrix; ◇: Geometric Multicloud approach. $h = 0.03$.

before applying BiCGStab helps again, but, as before, does not solve the problem of increasing iteration numbers.

Comparing AMG to the Multicloud Approach. AMG methods are known to be very efficient linear solvers for elliptic PDEs – like the Poisson equation – so they are a good benchmark to compare our method against. Note that in order to separate effects, we use both AMG and also our Multicloud approach without any acceleration, i.e. we do not use them as preconditioners for a Krylov method but in a stand-alone manner. For this experiment, we use the original matrix A for our Multicloud approach and the normalized matrix for the standard AMG method. Our AMG method regards positive couplings as weak couplings (cf. [138]). The experiment in Figure 3.8 shows

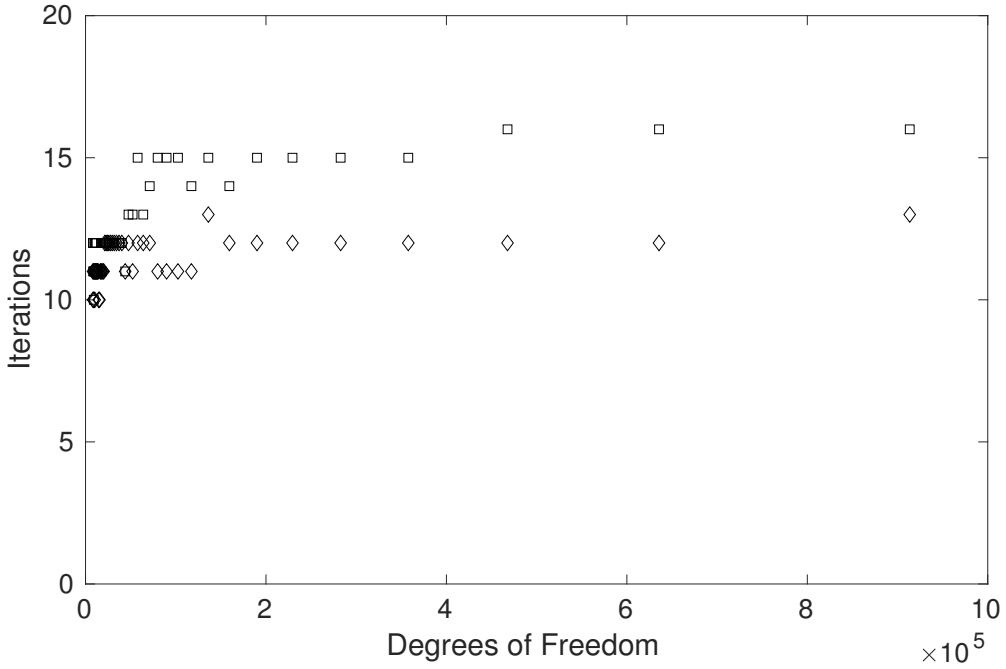


Figure 3.8: Number of iterations needed to converge when changing h and therefore the number of degrees of freedom. □: AMG; ◇: Geometric Multicloud approach.

that our method needs slightly less iterations compared to AMG, but both methods need an almost constant amount of iterations across the various refinement levels. We can therefore conclude that the Multicloud approach successfully makes use of the hierarchical structure of the problem by using coarse point clouds.

Drawbacks of the Geometric Approach

The geometric Multicloud approach presented here has shown that the general idea of Multigrid methods also work in the context of GFDMs. It does have a few drawbacks though: It is constructed for the pressure systems only and it is not clear how it

would translate to the coupled velocity-pressure systems because of their saddle point structure, which usually requires a special coarsening strategy, see Section 3.3.5. It is also not straight forward how to deal with boundary conditions other than Dirichlet conditions, where we employed special restriction and interpolation operators here. Lastly, this geometric method would have difficulties to deal with jumping coefficients. All these drawbacks can be addressed by using an Algebraic Multigrid method, which is why we turn to those methods in the following.

3.3.3 Introduction to Algebraic Multigrid (AMG)

The key idea in *Algebraic* Multigrid method is to automatically construct the multigrid hierarchy, that is, a hierarchy of coarse grids Ω_l, interpolation operators P_l, restriction operators R_l and coarse grid operators A_l based on the fine grid matrix $A_1 = A$ and the initial "grid" (index set) $\Omega_1 = \{1, \dots, N\}$.

These methods were introduced by Brandt [23] [22], Ruge and Stüben [117] and further developed in the industrial context by Stüben [138]. This development led to a number of different software libraries that implement various forms of Algebraic Multigrid, among them BoomerAMG [170] [56], SAMG [139] [112], various implementations in different languages like Python and even more implementations within other software packages like PETSc [9] [10] [11] or Trilinos [57].
Apart from those differences in implementation, there are also some AMG techniques that are tailored to certain specialized problems, leading to *subclasses* of the aforementioned *standard* AMG method. Examples include the smoothed aggregation AMG techniques by Vanek et al. [164] and more recently Lean AMG by Livne et al. [88]. Note that some of the software libraries are focused more on one or the other AMG techniques and some techniques have their own, exclusive, code base.

In AMG, like in any multigrid method, the coarse grid correction must reduce those error components that are not efficiently damped by the smoother M_l. Common choices for the smoother are Jacobi, Gauss-Seidel, or (S)SOR relaxation. For symmetric positive definite M-matrices A_l, the smooth error (i.e. the error components e that remain after a few iterations, $M_l e \approx e$) can be characterized using the matrix A_l: A smooth error only varies slightly along large off-diagonal negative couplings a_{ij}. We hence define, for each index $i \in \Omega_l$, the set $\mathcal{S}_i$ of *strong couplings* by identifying all i for which the negative respective matrix coupling a_{ij} exceeds a certain threshold,

$$\mathcal{S}_i = \{j \neq i : \ -a_{ij} \geq \alpha \max_{k \neq i} -a_{ik}\}. \tag{3.15}$$

A standard choice for α, to which we stick in the pressure systems but not in the coupled system, is $\alpha = 0.25$.

Remark 3.2. The basic convergence theory for standard AMG requires A to be symmetric and positive definite, see [138]. This is in contrast to the properties of the GFDM matrices in Chapter 2. Also, if A is an M-matrix, we can identify the couplings among which the error is smooth as large negative couplings. In GFDM matrices, which do not necessarily have the M-matrix property, positive couplings exist

and it is not clear if those couplings are always *weak couplings*. Their treatment is discussed and experiments are carried out in Chapter 4.

The strong couplings define the edges of a graph whose nodes are given by Ω_l. We determine a maximal independent set (*the coarse grid points*) $\mathcal{C}_l \subset \Omega_l$ such that each *fine grid point* $i \in \mathcal{F}_l = \Omega_l \setminus \mathcal{C}_l$ is strongly connected to at least one coarse grid point $j \in \mathcal{C}_l$. The splitting $\Omega_l = \mathcal{C}_l \cup \mathcal{F}_l$ is referred to as the *C/F-splitting* on level l.
Then, we build the *interpolation operator* I_l row by row using standard interpolation [138], so that each value at point $i \in \mathcal{F}_l$ is interpolated from the values at (directly or indirectly) strongly connected coarse grid points $\mathcal{C}_l$. The set C_l serves as coarse mesh Ω_{l+1}, while the coarse level matrix is computed by the Galerkin product,

$$A_{l+1} = I_l^T A_l I_l, \tag{3.16}$$

with I_l^T being the *restriction* operator.
We apply this procedure recursively until the size of the matrix is reasonable small for direct solution. Then, we can start the usual V-cycle:

1. Smooth the error by applying ν_1 iterations of a *relaxation* operator M_l to the current approximation u_l^n:
$$\tilde{x}_l^n = M_l^{\nu_1} u_l^n$$
2. Compute and restrict the residual to the coarse level:
$$r_{l+1} = I_l^T \left(f_l - A_l \tilde{x}_l^n\right)$$
3. Solve the coarse level equation either recursively or directly:
$$e_{l+1} = A_{l+1}^{-1} r_{l+1}$$
4. Interpolate the computed correction back to the fine level:
$$e_l = I_l e_{l+1}$$
5. Apply the correction and another ν_2 iterations of the relaxation:
$$u_l^{n+1} = u_l^{\nu_2} \left(\tilde{x}_l^n + e_l\right)$$
6. Continue with step 1 until the approximation u_l^{n+1} fulfills a specified termination criterion.

3.3.4 AMG Assumptions in the Context of GFDM Matrices

The matrices we need to solve in GFDMs are generally non-symmetric, non-M-matrices although they represent a Poisson-like problem. But because the underlying problem is an elliptic Poisson-like problem, the application of AMG methods can be heuristically motivated.
Here, we want to point out which properties that are required for the AMG *theory* are lacking in the GFDM matrices and why this poses a problem. We will go into the *practical* treatment of those issues in Chapter 4.

- *Symmetry:* The lack of symmetry means that standard AMG convergence theory cannot be applied, because the matrix A does not define an inner product. There have been different approaches to establish AMG convergence theories for different kinds of problems that do not require symmetry. We will briefly discuss some of those later.
  Since we normally do not use AMG as a stand-alone solver but merely as a preconditioner, the lack of symmetry means that only some Krylov methods that can deal with non-symmetric matrices, like BiCGStab and flexible-GMRES, can be considered.

- *M-matrix property:* In the classical AMG theory, the M-matrix property is required to establish the *smoothing property* of the Gauss-Seidel relaxation. This in turn leads to the notion of an *algebraically smooth error* that dictates the couplings along which the coarsening can take place. For a symmetric, positive-definite M-matrix, the error is *smooth* along strong couplings (cf. [138] Section 3.3.1). Note though that in our GFDM matrices, there may also exist positive couplings, and therefore those matrices do not have the M-matrix property.

- *Irreducibility and Singularity:* Usually, a matrix A arising from FDM discretizations is irreducible, i.e. there does not exist any permutation matrix P of the matrix rows and columns such that
  $$PA = \begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}. \tag{3.17}$$
  This means that A does not decompose into two independent subsystems. Because of geometric features or an insufficient discretizations size, this situation can occur though in the FPM, which is why the arising matrix can sometimes be reducible and even singular, see Section 5.2.
  If both A_1 and A_2 are regular matrices themselves, applying AMG to A will essentially yield the same result as applying AMG to A_1 and A_2 separately. In case one of the two blocks is singular, A becomes singular as well. In this case of course, AMG cannot be applied to A. However, if for example A_1 is singular, AMG still can be applied to A_2. This is precisely what we discuss in Section 5.2.

- *Elimination of Boundary Conditions / Scaling:* Algebraic Multigrid methods assume that the scale of the matrix coefficients is somehow related to the physical problem, i.e. the underlying PDE. Jumping coefficients usually indicate jumps in the material properties. This is crucial for the notion of a smooth error and therefore the definition of strong couplings. The different scales between the interior points and the boundary however are of purely algebraic nature and therefore the coefficient sizes near the boundary do not necessarily indicate the direction of the smooth error components there. Since the elimination of boundary conditions is hardly possible in GFDMs, we counter this issue by normalizing the matrix.

- *Diagonal Dominance:* Weak diagonal dominance is important for the theory of AMG because it ensures at least some – however slow – convergence of the

standard Gauss-Seidel smoothing scheme. This is violated by GFDM matrices because the solution of the least squares problem may yield mixed-sign stencils that still preserve a row sum of 0, meaning that in these cases the weak diagonal dominance must be violated. It is therefore advisable to improve the diagonal dominance, as Section 4.5.4 shows. In the FPM in particular, the coupled velocity-pressure system is highly non-diagonally-dominant because of the couplings between pressure and velocity.

Different AMG strategies have been developed addressing the seemingly biggest issue here which is the non-symmetry. They still make assumptions though, that do not fit for the matrices under consideration here: Notay et al. for example examine non-symmetric M-matrices using aggregation-based techniques [104] [101] [105]. Both Southworth et al. [135] [134] [136] and Lottes [89] work on *inherently* non-symmetric problems like convection dominated problems for example. This is opposed to the applications of GFDMs that we are considering here where the (Poisson-like) problem is symmetric but the discretization yields a non-symmetric matrix.

3.3.5 AMG for Saddle Point Systems

When using the coupled approach in the FPM, we need to solve a linear system which couples the velocity components and the correction pressure. Recall that the discretized equation (2.61) yields a saddle point system of the form

$$\underbrace{\begin{pmatrix} \mathbf{A} & B \\ C & -D \end{pmatrix}}_{=:\mathcal{K}} \begin{pmatrix} \mathbf{v} \\ p \end{pmatrix} = \begin{pmatrix} \mathbf{g} \\ f \end{pmatrix}, \tag{3.18}$$

with $D \to 0$ for $\Delta t_{\text{virt}} \to 0$. We will see later that the value of Δt_{virt} determines the AMG strategy that we will use to solve the system.
The linear system (3.18) is difficult to solve for standard AMG methods, as it is not diagonally dominant, because the submatrix D includes a factor of Δt_{virt}, whereas C is independent from that parameter. For the same reason, the system is not definite. Secondly, it is not clear how such a system could be coarsened: If treated as one single matrix, all the couplings within the D block would potentially be treated as weak couplings, because the coefficients in C outweigh them for sufficiently small Δt_{virt}. Similarly, the couplings in A are not comparable to those in B.
The idea here is to use an unknown-based AMG approach [117], i.e. treating the *unknowns* x-velocity, y-velocity, z-velocity and pressure separately, thus determining interpolation operators I_u, I_v, I_w, I_p for each individual unknown and stitching them together to construct the full interpolation operator

$$\mathcal{I} = \begin{pmatrix} I_u & & & \\ & I_v & & \\ & & I_w & \\ & & & I_p \end{pmatrix}. \tag{3.19}$$

Uzawa-smoothing

Because the linear system (3.18) is not diagonally dominant, we cannot employ standard smoothers like Jacobi or Gauss-Seidel. Instead, we employ an inexact Uzawa scheme [122] of the form

$$\mathbf{v}^* \leftarrow \mathbf{v}^{it} + \hat{\mathbf{A}}^{-1}\left(\mathbf{f} - \mathbf{A}\mathbf{v}^{it} - Bp^{it}\right), \tag{3.20}$$

$$p^{it+1} \leftarrow p^{it} + \hat{S}^{-1}\left(g - C\mathbf{v}^* + Dp^{it}\right), \tag{3.21}$$

$$\mathbf{v}^{it+1} \leftarrow \mathbf{v}^{it} + \hat{\mathbf{A}}^{-1}\left(\mathbf{f} - \mathbf{A}\mathbf{v}^{it} - Bp^{it+1}\right). \tag{3.22}$$

Here, $\hat{\mathbf{A}}$ denotes the diagonal of $\mathbf{A}$ scaled such that $\hat{\mathbf{A}} - \mathbf{A}$ is positive definite. The diagonal matrix $\hat{S}$ is formed such that $\hat{S} - C\hat{\mathbf{A}}^{-1}B - D$ is positive definite. Details on the choice of $\hat{\mathbf{A}}$ can be found in [98].

Coarsening the Pressure

We now look at the propagation of the respective errors $\mathbf{e_v}$ and e_p of velocity and pressure during each iteration step (3.20)–(3.22),

$$\begin{aligned}\begin{pmatrix}\mathbf{e_v}\\ e_p\end{pmatrix}^{it+1} = &\begin{pmatrix} I & -\hat{\mathbf{A}}^{-1}B \\ 0 & I\end{pmatrix}\cdot\begin{pmatrix} I & 0 \\ \hat{S}^{-1}C & I\end{pmatrix}\\ &\cdot\begin{pmatrix} I-\hat{\mathbf{A}}^{-1}\mathbf{A} & 0 \\ 0 & I-\hat{S}^{-1}\left(C\hat{\mathbf{A}}^{-1}B+D\right)\end{pmatrix}\cdot\begin{pmatrix}\mathbf{e_v}\\ e_P\end{pmatrix}^{it}.\end{aligned} \tag{3.23}$$

An efficient AMG method needs to reduce the error components that cannot be quickly damped by the smoother. The rightmost factor in equation (3.23) suggests that the error propagation of the Uzawa method is, at least partly, described by a Jacobi-like iteration using the matrix

$$\begin{pmatrix}\mathbf{A} & 0 \\ 0 & C\hat{\mathbf{A}}^{-1}B + D\end{pmatrix}. \tag{3.24}$$

This heuristically motivates us to use the matrices $\mathbf{A}$ and $C\hat{\mathbf{A}}^{-1}B + D$ to build the coarse spaces and interpolation operators for velocity and pressure, respectively. We do so by using the aforementioned unknown-based AMG ideas [117] creating an interpolation operator of the form (3.19). Clees showed in [29] that AMG techniques can be applied for the velocity matrix $\mathbf{A}$. The numerical experiments in [99] show that for virtual time step sizes of $\approx 0.1\Delta t$ it is sufficient to find a pressure coarsening based on the pressure matrix D alone.
In cases with smaller virtual time step sizes though it is necessary to use the ideas described in [98] and to compute the Schur-Complement

$$C\hat{\mathbf{A}}^{-1}B + D. \tag{3.25}$$

The coarsening as well as the transfer operators are then built based on the Schur-Complement, rather than D. Since this introduces a substantial computational overhead though, we try to stick to using D only when coarsening the pressure whenever

possible, which means that the difficulties introduced by the saddle point structure of the problem is dealt with by the Uzawa smoother, the *Alternate-Block-Factorization* technique [14] described in the next section and the appropriate scaling of the system which we will discuss in Section 4.3.

Alternate-Block-Factorization

In the case of non-constant viscosity η, the off-diagonal velocity matrix blocks A_{xy}, $x \neq y$ can also contain significant non-zero entries, cf. Section 2.4.3. In this case, simple unknown-based AMG might not be sufficient and a pre-processing of the matrix on the finest level is needed: We use the *Alternate-Block-Factorization* idea [14]. To this end, let us renumber the matrix $\mathcal{K}$ by the discretization points,

$$\mathcal{K} = \begin{pmatrix} \tilde{K}_{11} & \tilde{K}_{12} & \dots & \tilde{K}_{1N} \\ \tilde{K}_{21} & \tilde{K}_{22} & \dots & \tilde{K}_{2N} \\ \vdots & & \ddots & \vdots \\ \tilde{K}_{N1} & \tilde{K}_{N2} & \dots & \tilde{K}_{NN} \end{pmatrix}, \tag{3.26}$$

where each small matrix $\tilde{K} \in \mathbb{R}^{4\times 4}$ represents the couplings between point i and point j,

$$\tilde{K}_{ij} = \begin{pmatrix} \tilde{\mathbf{A}}^{(i,j)} & B^{(i,j)} \\ C^{(i,j)} & -D^{(i,j)} \end{pmatrix} = \begin{pmatrix} a_{uu}^{(i,j)} & a_{uv}^{(i,j)} & a_{uw}^{(i,j)} & b_u^{(i,j)} \\ a_{vu}^{(i,j)} & a_{vv}^{(i,j)} & a_{vw}^{(i,j)} & b_v^{(i,j)} \\ a_{wu}^{(i,j)} & a_{wv}^{(i,j)} & a_{ww}^{(i,j)} & b_w^{(i,j)} \\ c_u^{(i,j)} & c_v^{(i,j)} & c_w^{(i,j)} & -d^{(i,j)} \end{pmatrix}. \tag{3.27}$$

We scale $\mathcal{K}$ from the left using a block-diagonal matrix, where each diagonal block takes the form

$$\begin{pmatrix} \left(\tilde{\mathbf{A}}^{(i,i)}\right)^{-1} & \\ & I \end{pmatrix}. \tag{3.28}$$

Hence, all cross-velocity couplings inside a point are eliminated. We obtain (now reordering the system back by unknowns) the linear system

$$\bar{\mathcal{K}} = \begin{pmatrix} \bar{A}_{uu} & \bar{A}_{uv} & \bar{A}_{uw} & \bar{B}_u \\ \bar{A}_{vu} & \bar{A}_{vv} & \bar{A}_{vw} & \bar{B}_v \\ \bar{A}_{wu} & \bar{A}_{wv} & \bar{A}_{ww} & \bar{B}_w \\ C_u & C_v & C_w & -D \end{pmatrix}. \tag{3.29}$$

Now, we apply unknown-based AMG to the block-scaled matrix $\bar{\mathcal{K}}$.

Stabilizing the Interpolation

We can optionally enhance the quality of the interpolation by an additional stabilization factor. To this end, let us sort the velocity variables (regardless whether they belong to u,, v, or w) by coarse and fine grid points

$$\mathbf{v} = \begin{pmatrix} \mathbf{v}_F \\ \mathbf{v}_C \end{pmatrix}, \tag{3.30}$$

and, correspondingly, re-write $\mathcal{K}$,

$$\mathcal{K} = \begin{pmatrix} \mathbf{A}_{FF} & \mathbf{A}_{FC} & B_F \\ \mathbf{A}_{CF} & \mathbf{A}_{CC} & B_C \\ C_F & C_C & -D \end{pmatrix}, \tag{3.31}$$

as well as the interpolation operator in equation (3.19),

$$\mathcal{I} = \begin{pmatrix} I_F & 0 \\ I_C & 0 \\ 0 & I_p \end{pmatrix}. \tag{3.32}$$

Now, we compute the *F-stabilized* interpolation by [98]

$$\hat{\mathcal{I}} = \begin{pmatrix} 1_{FF} & 0 & -\hat{A}_{FF}^{-1} B_F^T \\ 0 & 1_{CC} & 0 \\ 0 & 0 & I \end{pmatrix} \begin{pmatrix} I_F & 0 \\ I_C & 0 \\ 0 & I_p \end{pmatrix}. \tag{3.33}$$

This interpolation operator is no longer block-diagonal and requires more memory than the interpolation operator from equation (3.19). Its advantage is the increased stability, i.e. the coarse grid operator

$$\left(\mathcal{K}_H = \hat{\mathcal{I}}^T \mathcal{K} \hat{\mathcal{I}}.\right) \tag{3.34}$$

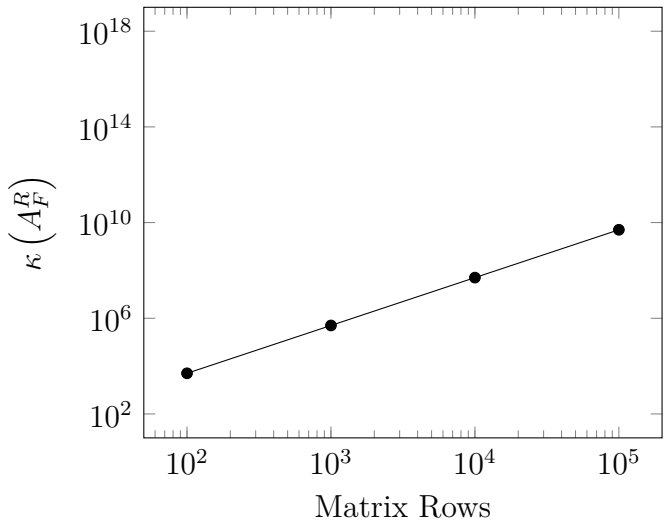
satisfies an inf-sup-condition if the fine grid operator $\mathcal{K}$ satisfies one, see Lemma 4.6 in [98] for details. In contrast, if we just use the block-diagonal interpolation operator from equation (3.19), the coarse grid matrix may even become singular depending on the interplay between the coarse velocity and pressure spaces [98].

3.4 The Problem of (Singular) Components

Later in this thesis, in Section 5.2, we will introduce the notions of *components*, which represent independent subsystems within the linear system. Due to the nature of the FPM, these components can become singular in some cases, for example in the model of a valve, see Section 5.4.3. While some one-level methods are not affected by these singular components because both the solution vector and the right-hand side vector are already 0 in the corresponding places, the convergence of an AMG method can decrease dramatically or even diverge. This is one of the reasons, other than performance considerations, why we cannot apply AMG in a black-box fashion to GFDM matrices. Section 5.2 will explain in detail how the linear system has to be *pre-processed* in order to avoid such problems.

3.5 Numerical Experiments

Let us investigate the applicability of some of the linear solvers introduced in this chapter to some of the model problems introduced in Chapter 2. We have already seen in Figure 3.2, that none of the current solvers performs well for the systems arising in the FPM. The experiments here shall motivate our idea to pursue the development of a new AMG method in the following two chapters.



Matrix Rows	100	1,000	10,000	100,000
AMG iterations	5	4	4	4

Figure 3.9: Top: The condition number of A_F^R increases with $1/h^2$. Bottom: The number of AMG iterations (relative residual reduction of $\epsilon_{\text{AMG}} = 10^{-8}$) on the other hand is almost constant.

3.5.1 AMG on Fornberg Matrix

We start by examining the behavior of a standard AMG method on the 1D Laplace problem on the interval $[0, 1]$ discretized using Fornberg's method, see Section 2.2.2. There are three cases that we want to consider: Using a regular grid, using a random set of nodes generated by a uniform distribution and a set of nodes generated in a similar manner as in the FPM.
For the matrix A_F^R resulting from the regular discretization, we find that it is identical to a standard Finite Difference matrix for this problem:

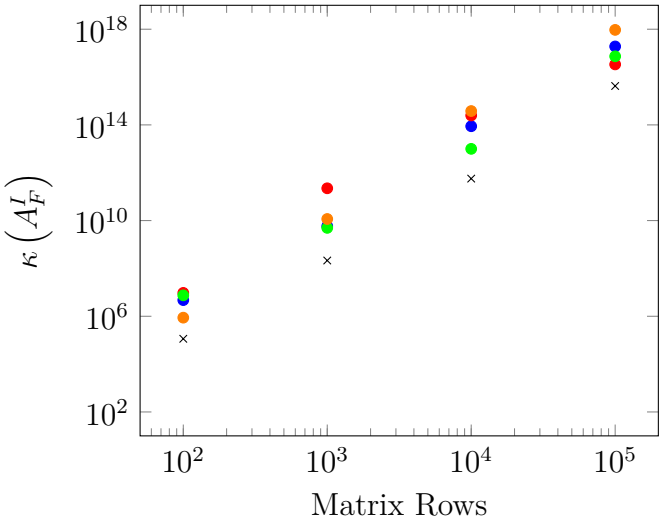
$$A_F^R = \frac{1}{h^2}\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}. \tag{3.35}$$

As we would expert, standard AMG shows a linear scaling behavior for these matrices for different values of h. Note that the condition number of A_F^R is proportional to $1/h^2$ for $h \to 0$, see Figure 3.9. This is to be expected as the eigenvalues for these matrices are (cf. [80])

$$\lambda_p\left(A_F^R\right) = -\frac{2}{h^2}\left(\cos(p\pi h) - 1\right),\ p = 1, 2, \ldots, \frac{1}{h} + 1. \tag{3.36}$$

The smallest eigenvalue is therefore

$$\lambda_1\left(A_F^R\right) = -\frac{2}{h^2}(\cos(\pi h) - 1) = \pi^2 + \mathcal{O}(h^2) \tag{3.37}$$



Matrix Rows	100	1,000	10,000	100,000
AMG iterations	17	diverges	diverges	diverges
AMG iterations (normalized matrix)	21	67	diverges	diverges

Figure 3.10: Top: The condition number of A_F^I for three different randomly generated grids with a different number of points (circles) and the condition numbers for the normalized matrices based on the matrices that were used to plot the orange circles (black crosses). Bottom: The number of AMG iterations ($\epsilon_{\text{AMG}} = 10^{-8}$).

and the largest eigenvalue is

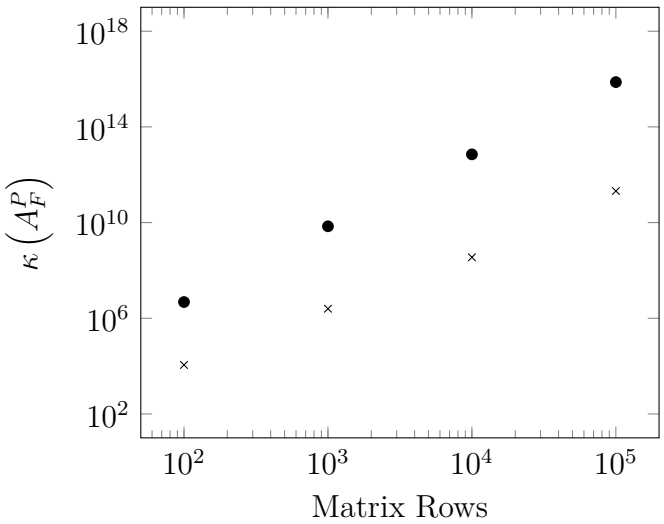
$$\lambda_{1/h}\left(A_F^R\right) = -\frac{2}{h^2}(\cos(\pi) - 1) = \frac{4}{h^2}. \tag{3.38}$$

Since A_F^R is symmetric, we have

$$\kappa\left(A_F^R\right) = \frac{\left|\lambda_{\max}\left(A_F^R\right)\right|}{\left|\lambda_{\min}\left(A_F^R\right)\right|}, \tag{3.39}$$

showing that the condition number is indeed proportional to $1/h^2$.

Next, we turn to A_F^I which is the matrix generated by Fornberg's method on a completely random set of nodes in $[0, 1]$ drawn from a uniform distribution. In this case, there is no prescribed minimum distance between any two nodes. This leads to numerical instabilities, resulting in a much higher condition number compared to the regular discretization for the same number of nodes, cf. Figure 3.10. Additionally, the matrix becomes non-symmetric (see Section 2.2.2). Due to the high condition number, just like any other numerical method, standard AMG does not work very well on these matrices. In order to decrease the condition number, we can employ preconditioning techniques similar to those described in Section 2.4.1, where we divided every row by its diagonal entry. In this case however, the normalization does not lead to an improvement in terms of AMG iterations needed to solve the linear system.

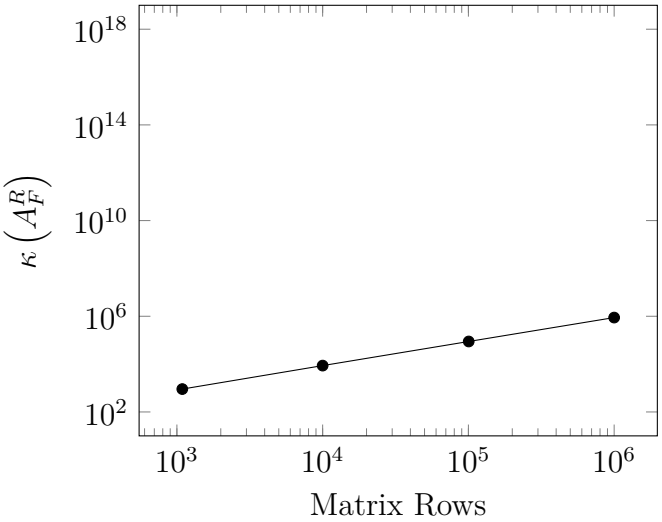


Matrix Rows	100	1,000	10,000	100,000
AMG iterations	58	diverges	diverges	diverges
AMG iterations (normalized matrix)	6	6	7	10

Figure 3.11: Top: The condition number of A_F^P for grids generated using a basic *point cloud management*. Circles for the non-preconditioned matrix and crosses after normalizing the diagonal entries to 1. Bottom: The number of AMG iterations ($\epsilon_{\text{AMG}} = 10^{-8}$) for these matrices.

Trying to mimic the point cloud management in the FPM, we can move away from the idea to introduce the nodes in a fully random fashion. Instead, we start by adding two nodes at the boundary and then fill the interior of our domain $[0, 1]$ by adding nodes next to the node we have added before. Whenever we add a node, we make sure that its distance to the previous node is not smaller than $(1-\epsilon)h$ and not bigger than $(1+\epsilon)h$. That way, we get a non-regular, yet not fully random, discretization of $[0, 1]$. The advantage over the previously discussed method is that by changing ϵ we can control the irregularity of the discretization and therefore also have some control over the condition number of the resulting matrix A_F^P. The matrix is again non-symmetric and we can improve its condition number by dividing every row by its diagonal element. Condition numbers for $\epsilon = 1/2$ and the corresponding numbers of AMG iterations are shown in Figure 3.11. We can see that those numbers are comparable to the regular case and much better than in the fully random case.

This leads to the conclusion that the non-symmetry of the matrix is less of a problem for a standard AMG method. The *point cloud management* that we discussed in Section 2.3.5 is rather important though, as a lack of such a preprocessing of the point cloud leads to numerical instabilities that also show in the linear solver, as we would have to expect. Lastly, we have also seen that the preconditioning of the matrix by dividing every row by its diagonal has a positive effect on both the condition number and the performance of AMG.



Matrix Rows	1000	10,000	100,000	1,000,000
AMG iterations	6	6	6	6

Figure 3.12: Top: Condition numbers for the mixed stencils problem. Bottom: The number of AMG iterations for these matrices (relative residual reduction of $\epsilon_{\text{AMG}} = 10^{-8}$) is constant.

3.5.2 AMG on the Mixed Stencils Matrix

The mixed stencils discretization described in Section 2.2.3 does not pose any problems to a standard AMG method[5], see Figure 3.12. Again, this is a heuristic hint that the non-symmetry of the matrix that we are encountering in the matrices arising in GFDMs is not a major issue, at least for the practical application of AMG. In this particular case, there is no difference in terms of AMG iterations compared to the standard stencil that leads to a symmetric matrix, when using BiCGStab as the Krylov method for the symmetric matrix as well. When using CG, as one would normally do for a symmetric matrix, the number of iterations increases slightly, but the overall runtime is better. Both observations are what we would expect for a symmetric matrix. For the matrices we are dealing with in this thesis, we need to stick to BiCGStab as our Krylov method though because of the non-symmetry of the matrix.

[5]Standard Ruge-Stüben coarsening, Gauss-Seidel smoothing, a direct coarse level solver and used as a preconditioner for BiCGStab.

Chapter 4

Solving the Linear Systems Arising in the FPM

In Chapter 3 we saw that AMG methods are a promising approach for the linear systems we are dealing with. We also saw in Figure 3.2, that the standard AMG method from [138] does not work well for these systems. Additionally, there is the problem of singular components.

In this chapter we will establish a new AMG method that is focused on GFDMs and the FPM in particular. The linear systems arising from the FPM are different from those in classical, mesh-based discretization methods, therefore special AMG techniques are required. Also, the transient character of the method needs to be taken into account to achieve optimal efficiency.

In order to solve the linear systems arising in the FPM as efficiently as possible, we need to design an AMG method that is tailored to the specifics of the FPM. For example, the density of the matrix and the possibly positive couplings need to be taken into account. We also need to scale the linear system in a fashion that allows us to design a Saddle Point AMG approach for solving the coupled velocity-pressure systems.

When thinking about the FPM we need to keep in mind that we are dealing with a transient simulation in which we have to solve many linear systems in different time steps. The AMG method we design takes this into account. Not only do we make sure that our method is as efficient as possible when solving one single system, but by re-using the same AMG hierarchy for different linear systems within a time step and even across time steps, we can again lower the overall cost of our linear solver.

We give some numerical examples showing the efficiency of our linear solver and the effect of re-using the hierarchy. The latter also has some numerical and performance effects on the FPM as a method itself. In this section, we will examine a model simulating the flow through a bifurcated tube. Depending on the Reynolds Number of the flow we will use both the segregated and the coupled approach in the FPM and show the performance of our linear solver for all three types of linear systems. In this chapter, all experiments are carried out on a single core. The next chapter will then deal with the parallelization of the method, with a focus on how robustness can be preserved in cases where the point cloud is not connected any more.

4.1 Velocity Systems

As mentioned in Section 2.4.2, the linear systems for the velocity in the segregated approach are highly diagonally dominant. Therefore we do not apply any specialized AMG strategies for those systems, but stick with a simple BiCGStab2 scheme, that has been proven to be sufficiently fast and stable, see Section 4.5.1. This section will also give some examples regarding the diagonal dominance of those systems and the computational effort to solve them by using BiCGStab2 versus using a sophisticated AMG method. It turns out that in most cases the overhead that comes with using an AMG method does not outweigh the lower iteration count.

4.2 Pressure Systems

The pressure systems that occur in the FPM are

- the hydrostatic pressure system (2.51),
- the correction pressure system (2.58),
- the dynamic pressure system (2.52).

In the segregated approach, they are solved in this very order and the velocity predictor is solved for after solving for the hydrostatic pressure. In the coupled approach, the velocity corrector is solved for in one linear system together with the correction pressure. Hence, in the coupled approach, there are only two pure pressure systems to be solved. As we discussed in Section 2.4.1, the pressure systems arising in the FPM are essentially Poisson-like systems.

4.2.1 Normalization of the Linear System

In Section 2.4.1 we saw that because it is not possible to eliminate the boundary conditions from the linear system and because these boundary conditions are scaled differently from the rest of the domain, we can improve the condition of the linear system by normalizing the diagonals to 1, i.e. by solving

$$DAu = Df, \tag{4.1}$$

with

$$D = \begin{pmatrix} 1/a_{11} & & \\ & \ddots & \\ & & 1/a_{nn} \end{pmatrix}. \tag{4.2}$$

Scaling the system like this is usually advised against in the context of AMG as DA is not necessarily symmetric, even if A is symmetric. In our case however, A is not symmetric to begin with so there is not much benefit in trying to preserve any symmetry. The numerical results in Section 4.5.2 and Section 3.5.1 show that normalizing the system improves the convergence rate noticeably while not introducing much computational effort.

4.2.2 Coarsening Strategies

Regarding the coarsening strategy, we recall from Section 2.4.1 that compared to standard discretizations of Poisson's equation the FPM matrices A are fairly dense. While with a standard Finite Difference method in 3D the Laplace stencil would include 7 entries, in the FPM we have 40 entries per row. When constructing the coarse level Galerkin operator

$$A_H = RAP, \tag{4.3}$$

this increased density becomes even more exaggerated, cf. Section 5.4.1. This means that

- the coarse level operator A_H consumes more memory than in comparable, mesh-based systems and
- since A_H has a lot of entries, the smoothing steps on the coarse level are more expensive, again compared to mesh-based systems.

To counter this effect, we coarsen more aggressively compared to what would be the standard AMG approach [138]: Where in the basic AMG algorithm, for a variable that we choose to be a coarse level variable (C-variable), all its strongly connected, *direct* neighbors become fine level variable (F-variable), in the so-called (l, k)-*aggressive coarsening* strategy this definition is extended to all variables that are either strongly connected direct neighbors or to which at least l paths of length k along strong connections exist. In our case, we used a $(1, 2)$-aggressive coarsening strategy, meaning that for every C-point i_C, all its strongly connected neighbors j_k as well as the points that are strongly connected to those j_k become F-points. This leads to a lot less C-points on the coarse level and therefore to an overall smaller number of non-zero entries. When using an aggressive coarsening strategy, we need to employ the so-called multi-pass interpolation [138] which proceeds in several passes, using the direct interpolation [138] where possible and then use interpolation formulas at neighboring F-variables for those F-variables that are not connected to any C-variable. For the aggressive coarsening strategy we are using here, this process will need at most four passes before every F-variable has been assigned an interpolation formula [138]. We use this aggressive coarsening strategy to create the second level in our hierarchy. The other levels – if needed – are created in the standard Ruge-Stüben fashion. Comparisons between the standard and the aggressive AMG coarsening can be found in Section 5.4.1.

Positive couplings

AMG methods have originally been developed with symmetric, positive definite M-matrices in mind. Stüben [138] shows how for these matrices the error of a given approximation, after applying some iterations of a relaxation scheme, varies smoothly along large negative couplings. In an M-matrix, all off-diagonal entries, i.e. couplings, are negative. This does not hold true for the GFDM matrices. But because of the measures being taken to make the matrix weakly diagonally dominant where possible, cf. Section 2.4.1, most of the couplings are in fact negative, cf. Section 4.5.4. In

addition to that, since there are 40 neighbors in almost every neighborhood[1], we expect to have at least one large negative coupling in every spatial direction, avoiding possible one-sided interpolations. Therefore, in our AMG algorithm we regard positive couplings as weak couplings, i.e. we neither coarsen nor interpolate along them.

4.2.3 Interpolation and Restriction

We employ a standard interpolation as described in [138] and let the restriction be the transpose of that interpolation. The idea to use the transpose of the interpolation as the restriction originates from the case of *symmetric* matrices but has shown to work sufficiently well in our case, too. We did conduct experiments with the modified *lAIR* restriction operator described by Southworth [135] but did not notice any significant improvements, possibly because Southworth's work focuses on naturally non-symmetric problems like convection dominated phenomena. Another reason is that with our preprocessing of the matrix and the AMG techniques we use the convergence rates of our new methods are already fairly competitive (as the numerical experiments show) and could not be improved much, if at all, by the modified restriction operator.

4.2.4 Smoothing Schemes

Although the pressure matrices arising in the FPM do not have some of the properties that most FD matrices have, they are still weakly diagonally dominant for the most part, cf. Section 4.5.4. Also, the pressure systems do not have any special block structure with scaling issues like the saddle point problems in the coupled approach have. Hence we can apply a standard smoother. In our case this is the Gauss-Seidel relaxation scheme applied in forward order in the pre-smoothing step and in reverse order in the post-smoothing step. We perform one iteration in the pre- and post-smoothing step each.

4.2.5 Accelerator

As explained in Chapter 3, Multigrid methods are usually used as a preconditioning method for Krylov subspace methods. Then, in AMG terminology, the Krylov subspace method being used is called an *accelerator* for the AMG method.
A standard accelerator choice for a Poisson-like problem discretized using Finite Difference Methods would be the CG method. Since we are dealing with non-symmetric matrices in the case of a GFDM discretization, we use the BiCGStab method instead, cf. Section 3.2.2.

4.2.6 Coarse Level Solvers

In most situations, we employ a direct solver on the coarsest level Ω_H of our multigrid hierarchy. The only exception to that is if the coarsest level is diagonally dominant

[1] Except for at the boundary.

with

$$\max_{i\in\Omega_H}\delta_i = \max_i \frac{\sum_{i\neq j}|a_{ij}|}{|a_{ii}|} \leq 0.9, \tag{4.4}$$

in which case we use a simple one-level relaxation method, the default being Gauss-Seidel. Regarding the direct solver, we have two main options: Intel's PARDISO and a standard LU-decomposition[2]. In the former case, the linear system on the coarsest level is collected on one single process, solved there and the solution is distributed back to all other processes. In the latter case, all processes involved in the AMG algorithm solve the coarse level system as well. Which of these methods gives a faster overall algorithm depends on the problem at hand, see Section 5.4.2. Both are direct methods though, so they do not influence the convergence of the overall AMG method. For the relaxation scheme in case of diagonally dominant coarse levels, we iterate until a relative residual reduction of 0.01 or a maximum number of 200 iterations is reached. The latter case would be considered as an error though but does not occur when the coarse level matrix is diagonally dominant as it is the case here.

4.2.7 Setup Re-use Within a Time Step

Multigrid methods always consist of two separate phases: The setup phase and the solution phase. In the setup phase, the coarse levels are created, along with the corresponding interpolation and restriction operators. Since the coarse level system is also known after these steps, any decomposition or other form of setup for the coarse level solver can also be counted towards the setup phase. Then, in the solution phase, the actual iterations are carried out. This phase consists of the smoothing, applying the transfer operators, solving on the coarsest level and applying the accelerator, see Section 3.3.3. The important observation here is that if we have to solve two linear systems

$$Au = f, \tag{4.5}$$
$$Au = g \tag{4.6}$$

in sequence, then we only need to compute the setup phase once and can re-use all the computed operators to also solve the second system.

Remark 4.1. Even if we had to solve

$$Au = f, \tag{4.7}$$
$$Bu = g \tag{4.8}$$

with $B \approx A$, $A, B \in \mathbb{R}^{n\times n}$, we still have a number of options in order to avoid having to compute the full setup for B:

- We can keep the full setup for A when solving B, including all transfer and coarse level operators. Note that when applying the smoother and the Krylov iteration on the finest level, we still use B. But on all the coarse levels, the operators that were originally constructed for solving $Au = f$ are used. In this method, we do

[2]BLAS implementation [79] [34] [18].

not need to compute anything in the setup phase, but the convergence of the overall method may not be very good, if A and B are too different.

- Another option is to use the C/F-Splitting (cf. Section 3.3.3) from A but re-compute all the transfer and coarse level operators using B. This way we save the cost of finding a C/F-Splitting for B, but we do need to re-compute all the operators. This gives an algorithm that is much better suited for solving $Bu = g$ but the setup phase is not as cheap as in the first option.

Unfortunately it is very hard to tell for any given A and B if one of the two options above or a completely new setup gives the fastest overall solution method. In some cases it can be a good option to perform a certain number of test cycles when solving $Bu = g$ and then either continue with the iterations if the convergence rate is similar to the one that was observed when solving $Au = f$ or to stop iterating and to create a new setup. Note that this approach may introduce some overhead and its success in terms of trying to save run-time is highly dependent on the application and also the specific model in some cases.

In the case of the FPM at least we know that the hydrostatic pressure system (2.51) and the correction pressure system (2.58) both yield the same matrix A and the only difference is in the corresponding right-hand sides. Therefore we only need to compute one setup in order to solve both of these systems.
For the dynamic pressure system (2.52) the same holds in the default case. However, it is possible to impose different boundary conditions on the dynamic pressure systems for reasons of stability of the Chorin projection approach, see Section 2.3.3. In this case, the matrix B corresponding to the dynamic pressure system changes in those rows that correspond to boundary conditions that are being changed. Not only can this lead to a decreased convergence rate when re-using the setup from the hydrostatic system but these changes may also introduce or remove independent subsystems within the linear system which is important for our AMG method in the parallel case, see Section 5.2. In this case, the setup cannot be re-used because the additional couplings introduced or removed by the change in the boundary conditions that are responsible for merging or creating independent subsystems also have an impact on the communication pattern of our AMG method. Therefore, the setup cannot be re-used without substantial computational effort in the update of the communication patterns which is why in those cases we opt to compute a completely new setup. For these reasons, we only re-use the setup from the hydrostatic pressure system for the dynamic pressure system if the boundary conditions did not change.

4.3 Coupled Systems

Due to their saddle point structure, the coupled velocity-pressure systems that arise when using the coupled approach in the FPM cannot be solved using the same AMG techniques as in the pressure systems.
On the other hand though, Section 2.3.4 showed that in some situations the segregated approach is not sufficient and we need to use the coupled approach. It is therefore

essential to have an efficient linear solver for those coupled systems as well.

Since some of the strategies that we presented in Section 3.3.5 are computationally intensive though, this section describes when we should use which of those techniques.

As opposed to the pressure systems, we do not normalize the linear system before passing it into the solver. This is because the techniques described in [98] assume that the saddle point system has the form

$$\mathcal{K} = \begin{pmatrix} \mathbf{A} & B \\ C & -D \end{pmatrix}, \tag{4.9}$$

with $C = B^T$ which is not the case in the FPM, see equation (2.61). We therefore introduce a *native* scaling:
Our goal here is to make sure that although $C \neq B^T$, the entries in C and B are at least of the same scale. To this end, we find that the entries in B are of order

$$\mathfrak{p} = \frac{\Delta t}{\rho} C_G, \tag{4.10}$$

whereas the entries in C are of order

$$\mathfrak{v} = C_G. \tag{4.11}$$

In both cases, let C_G denote the order of magnitude for the coefficients of the discretized gradient operator. This means that by scaling the matrix from the left

$$\begin{pmatrix} \mathbb{1} & 0 \\ 0 & \frac{\Delta t}{\rho} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{A} & B \\ C & -D \end{pmatrix}, \tag{4.12}$$

we get a modified matrix

$$\bar{\mathcal{K}} = \begin{pmatrix} \bar{\mathbf{A}} & \bar{B} \\ \bar{C} & -\bar{D} \end{pmatrix}, \tag{4.13}$$

in which $\bar{B} \approx \bar{C}$, entry wise. This does not take into account the difference in scale between boundary conditions and interior points. The numerical experiments at the end of this chapter, in Chapter 6 and also [99] have shown though that this does not seem to be necessary in order to achieve good convergence. The reason for this is that the saddle point character of the system has a much greater impact on the convergence compared to the different scaling between interior and boundary and it is much more important to be able to deal with the saddle point character than it is to reduce the condition number by scaling the boundary conditions appropriately. It would also be possible to combine these two scalings, but we did not find that to be necessary.

Secondly, we need to deal with the velocity-velocity couplings within the single points, i.e. the off-diagonal coefficients in $\bar{A}_{uu}$, $\bar{A}_{vv}$ and $\bar{A}_{ww}$ in equation (3.29). This is necessary because both the unknown-based AMG and the Saddle Point AMG approaches that we use assume that in the velocity system $\bar{\mathbf{A}}$ each of the three velocity

components u, v, w is mostly independent of the other two components and therefore AMG could be applied to each of the components individually. We resolve the velocity-velocity by using the Alternate-Block-Factorization described in Section 3.3.5, so that the pre-processed system that our solver needs to solve does not have any velocity-velocity couplings within any given point. The computational effort for solving those couplings is noticeable, but not substantial. So in order to gain the best performance, we only activate this pre-processing step in case of a non-constant viscosity, as in the case of constant viscosity these couplings are not present in the interior of the domain, cf. Section 2.4.3. Velocity-velocity couplings within boundary points are not as significant in this context and can be included in the linear system.

After having pre-processed the linear system accordingly, we have a couple of different options to solve it. In all these options, we use a threshold of $\alpha = 0.5$ to identify strong couplings (cf. equation (3.15)) as opposed to $\alpha = 0.25$ in the case of the pressure systems.

4.3.1 Unknown-based AMG with ILU-Smoothing

One option we have is to use the unknown-based AMG techniques described by Ruge [117] that we already touched on in Section 3.3.5, i.e. we find a coarsening and the respective interpolation operators separately for every one of the four unknowns x-, y-, z-velocity and pressure, thereby not taking into account the inter-unknown couplings. This however does not solve the issue of needing a smoother on the finest level that is capable of dealing with highly non-diagonally dominant, non-definite systems. As Section 3.3.5 describes, Uzawa schemes are useful in this context, however they introduce a number of new parameters and need additional computational effort for finding the matrices $\hat{\mathbf{A}}$ and $\hat{S}$.
A computational wise cheaper alternative is to use ILU-smoothing [169]. In many situations this is sufficient, if the virtual time step size Δt_{virt} is not too small. Section 4.5.7 includes an experiment that shows the usability of this approach.

4.3.2 Saddle Point AMG

For simulations where the unknown-based AMG method with ILU-smoothing is not sufficient, we need to use the Saddle Point AMG approaches from [98] as described in Section 3.3.5. As discussed in [99], the Saddle Point AMG approach with Uzawa-smoothing and a pressure coarsening based on D can be used with

$$\frac{\Delta t_{\text{virt}}}{\Delta t} \geq 0.01 \tag{4.14}$$

in most situations, whereas for smaller virtual time steps the pressure coarsening based on $C\hat{\mathbf{A}}^{-1}B + D$ is needed [99].

In the Uzawa-smoothing, see equations (3.20)–(3.22), we use

$$\hat{\mathbf{A}} = \tau \cdot \operatorname{diag}(\mathbf{A}), \tag{4.15}$$

where $2 \leq \tau \leq 10$ is usually sufficient in order to obtain a converging Uzawa method. For $\hat{S}$ (cf. Section 3.3.5) we want to use

$$\hat{S} = \omega \cdot \operatorname{diag}\left(C\hat{\mathbf{A}}^{-1}B + D\right). \tag{4.16}$$

Here, ω is a user parameter but can usually be left at its default value of $\omega = 0.7$. In the case of $\Delta t_{\text{virt}}/\Delta t \geq 0.01$ where we do not want to explicitly compute $C\hat{\mathbf{A}}^{-1}B + D$, we can use different means to estimate the diagonal of $S = C\hat{\mathbf{A}}^{-1}B + D$: The default choice in the new AMG method presented here is

$$s_{ii} \approx \sum_k \sum_j \left| \frac{d_{ij} b_{jk}}{a_{jj}} \right|. \tag{4.17}$$

Other choices include

$$s_{ii} \approx \sum_j |d_{ij} bji / a_{jj}|, \; s_{ii} \approx \sum_k \left| \sum_j d_{ij} b_{jk} / a_{jj} \right| \text{ and } s_{ii} \approx \left| \sum_j d_{ij} b_{ji} / a_{jj} \right|. \tag{4.18}$$

The results of the different options are similar, however the first option shows slightly better convergence rates in many models [116]. The estimation for $\hat{S}$ can also serve as an indicator as to how good the coarsening based on the pressure-pressure matrix D is, or if the more sophisticated coarsening based on the Schur-Complement $C\hat{\mathbf{A}}^{-1}B + D$ is needed [98].

Only in very rare cases (mostly for very small virtual time step sizes Δt_{virt}) do we need the stabilized interpolation (3.33). One of these cases can be found in [99]. Again, we refer to Section 4.5.7 for an experiment comparing the different techniques.

4.4 Setup Re-Use Across Time Steps

So far, we have only discussed re-using the AMG setup within one time step. This is only relevant for the pressure systems since there is no more than one velocity or coupled velocity-pressure system per time step. Therefore, with the arguments we have given so far, the AMG setup for the coupled velocity-pressure system cannot be re-used anywhere.
In order to re-use the AMG setup from those systems and also to increase the number of systems for which we can re-use the setup for the pressure systems, we need to think about re-using a particular setup across different time steps.
However, compared to mesh-based discretization methods, GFDMs add more challenges to the re-use of AMG setups: Due to the point cloud management that is carried out in every time step, the size of the point cloud as well as the neighborhood relations between points may change. This in turn means that for example two hydrostatic pressure systems from two consecutive time steps

- do not have to be of the same size and

- can have logically different connectivity patterns.

This gives rise to a number of questions:

- If a point is deleted that was associated to a coarse level point in time step t, this has an immediate impact on the full AMG hierarchy in time step $t+1$.
- Does a point that has been added in time step $t+1$ necessarily become an F-point or can it become a C-point in certain situations? If so, do we need to re-built the full AMG hierarchy for that time step?
- What happens if due to the changing neighborhoods, a system becomes reducible in time step $t+1$? A good example for this is when the fluid forms droplets over time: Those necessarily lead to independent subsystems in the linear systems and therefore the full system becomes reducible. This may then influence the convergence dramatically in certain situations which is why we employ means to find those subsystems, see Section 5.2. Therefore, if the neighborhood structure (potentially) changes, we need to re-run our algorithm which means we also have to compute a new AMG setup.

These problems do not exist for discretizations with static meshes. The only variable to consider there is how closely related the two matrices A and B are in terms of coefficient size.

Options to overcome the obstacles for re-using the AMG setup in GFDMs are:

- Keep two separate point clouds: A static one and a moving one. The dynamic one is moved in every time step, just like described above. It can then be used to discretize the continuity equations, but before solving the linear systems, the coefficients need to be projected onto the static point cloud. The idea is similar to what the Material Point Method (MPM) [145] does, where all the physical quantities are mapped back and forth between a background mesh and a moving *point cloud*. This step introduces an error that would need to be accounted for. It would also be necessary to move the static point cloud at least in some time steps in order to keep track with moving geometries for example. In fact, this idea introduces difficulties that are similar to those of mesh-based methods and the error introduced by the projection between the point clouds would need to be investigated carefully. For these reasons, this idea is not further pursued.
- Secondly, one may drop the idea of a moving point cloud all together and return back to an Eulerian approach. This is entirely possible with GFDMs, but this approach drops a lot of the strengths of these methods. It does have applications though when looking at transport problems [128].
- An idea that would not interfere with the FPM at all is to solve the problems arising due to the movement of the point cloud completely on the AMG side, especially in the setup. The insertion and deletion of points, and therefore rows in the matrix, could possibly be dealt with using locally adopted AMG

setups. This would reduce the computational effort in the setup phase, because only certain parts of the C/F-Splitting and the coarse level operators had to be rebuilt. To the best of the author's knowledge though, approaches like this have not been considered so far.

- Another idea that does not alter the FPM algorithm at all would be to use aggregation based coarsening in AMG. In this coarsening technique, instead of finding a maximal independent set of coarse level points [138], so called aggregates are formed that consist of multiple points. The interpolation from the coarse level to the fine level within each aggregate is constant. This makes it easy to add or remove single points (rows) from an aggregate, as long as at least one point of the aggregate is still present.
  It is unclear though how to handle the changing neighborhoods in this setting as well. In addition to that, convergence rates of aggregative AMG methods are often not as good as for standard coarsening approaches. Although the setup phases for aggregative methods are usually cheaper, we found that in our case this is different and the additional loss in the convergence rate makes aggregation based AMG not usable in our context, cf. Section 5.4.1. Also, since the point cloud is moving, the convergence rate of an AMG solution phase with a re-used setup can be expected to be even worse.
  Because of that, aggregation based AMG can be ruled out as well.

- The final idea, which we found to be the most useful one, is to skip the point cloud organization phase for a predefined[3] number of time steps. If the point cloud management phase is skipped, then no new points are introduced or deleted and no neighborhood relationships between points are changed. The only change is the position of the points and therefore the coefficients in the stencil. So the two (hydrostatic pressure, for example) matrices A from time step t and B from time step $t+1$ are structurally identical and have the same size, but include different coefficients. In this situation we can use standard setup re-use strategies from AMG, like re-using the full setup from time step t in time step $t+1$. Not only does this strategy save time in the AMG setup, but it also saves time in the FPM algorithm itself, as the point cloud organization is skipped. On the other hand, the point cloud management is an essential step in the FPM algorithm and cannot be skipped for too many time steps. How many time steps can be carried out without managing the point cloud depends on the particular problem being solved and also on the question the simulation aims to answer. Generally speaking, the slower the movement of the point cloud or the smaller the time step, the more likely it is that skipping the point cloud management has a negligible impact on the quality of the solution.
  Section 4.5.6 compares the benefits of skipping the point cloud management and re-using the AMG setup to the loss in quality of the solution.

[3] Adaptive strategies are also possible.

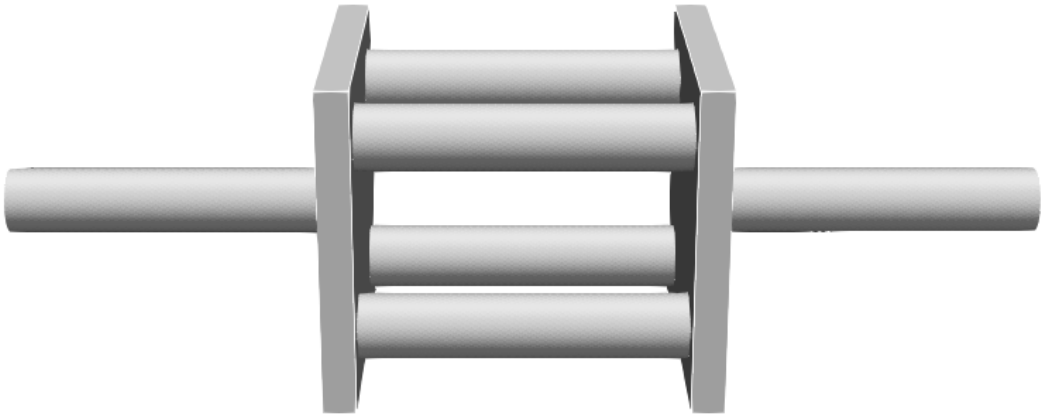


Figure 4.1: Geometry for the bifurcated tube model.

4.5 Numerical Experiments

In this section we will study a 3D model problem using the complete FPM scheme, i.e. we will deal with all three different types of linear systems and will consider various types of solvers and the AMG strategies discussed in this chapter to solve these systems.

For this purpose, we use a model of a bifurcated tube with an inflow boundary condition at one end and an outflow boundary condition at the other end. No-slip boundary conditions are imposed at the tube walls. The geometry for this model is depicted in Figure 4.1. The initial velocity of the fluid and also the velocity of the fluid at the inflow is $v_0 = 1.0m/s$. We also consider gravity as an external force in this model, which means that we have to solve the hydrostatic pressure system in every time step[4]. In the original version of the model, we choose the viscosity μ so that the Reynolds Number $Re = 1000$ which means that we can use the segregated approach to begin with. In Section 4.5.7 we will examine the same model with a much more viscous flow though in order to examine our methods for the coupled approach as well. The smoothing length is set to $h = 0.06$ in the default case, but we will vary it in various experiments here.

Remark 4.2. A major advantage of GFDMs over other meshfree methods is that imposing boundary conditions is fairly simple: At boundary points, we do not use a stencil for discretizing the PDE like we do at interior points, but we use stencils representing the desired boundary conditions. Various types of boundary conditions are discussed throughout the GFDM literature. The most important ones (Dirichlet, Neumann and free surface) are discussed in [141], Section 2.5.4.
Apart from the influence on the linear system through their different scale that we previously discussed, neither of the boundary conditions we tested had a significant

[4]For an incompressible flow model without any external forces and $S_{\text{solid}} = 0$, the right-hand side for the hydrostatic pressure system is always 0 and therefore the solution to that system does not have to be computed.

influence on the linear solver, which is why in the following we will not conduct any experiments comparing the linear solver for different boundary conditions.

Remark 4.3. In matrix-based experiments throughout the numerical results in this thesis, we use a relative residual reduction of 10^{-8} as our stopping criterion. For experiments involving full FPM simulations, a different criterion is used: The iteration is stopped when

$$r^{(n)} \cdot \epsilon_{\text{local}} \leq 1, \tag{4.19}$$

where ϵ_{local} is a weighting vector that weighs the residual at every point and $r^{(n)}$ is the new residual vector. The application of this special criterion does not differ too much from using a relative residual reduction of 10^{-8} in most situations, though.

4.5.1 Segregated Velocity Systems

Defining a measure for the diagonal dominance of a row i as

$$\delta_i = \frac{\sum_{i \neq j} |a_{ij}|}{|a_{ii}|}, \tag{4.20}$$

for the segregated velocity system in the high Reynolds Number case we find that, as has to be expected from the theoretical considerations in Section 2.4.2, the average diagonal dominance

$$\bar{\delta} = \frac{\sum_i \delta_i}{N}, \tag{4.21}$$

N being the number of rows, is $\bar{\delta} \approx 10^{-2}$ in this case meaning that on average, the diagonal is 100 times larger compared to the sum of the absolute off-diagonal values. Also, the diagonal dominance is only violated in some rows with mixed positive and negative off-diagonals, which only make up about 1.8% of all rows.

Because of this diagonal dominance, the system can be solved in only two Gauss-Seidel iterations, two BiCGStab iterations or a single BiCGStab2 iteration with a relative reduction of the residual of 10^{-14}. In this test, we used a random vector r to generate a right-hand side $f = Ar$ and an initial solution $u_0 = 0$.

Since these iterations are substantially faster than AMG iterations (cf. Remark 4.6), even neglecting the setup cost of AMG, we generally do not consider AMG methods for the segregated velocity systems in the FPM. For these systems, it is far more important to have an efficient implementation of a one-level method that performs well even on larger systems.

In this thesis, especially in the next sections, we will mention the segregated velocity systems only when they show a different behavior to what we have seen here. Usually, we stick to the aforementioned one-level methods, though.

4.5.2 Normalization

The experiments in Chapter 3 have already indicated that for the non-symmetric discretizations of the Laplace operator that we are looking at in this thesis, normalizing

h	.03	.04	.05	.06	.07	.08	.09	.1
Matrix Rows	366k	166k	93k	63k	41k	27k	22k	17k
Original Scaling	92	99	114	122	117	103	110	103
Normalization	30	28	32	35	32	33	30	31

Figure 4.2: Sum of AMG iterations across all three pressure systems in the 100th time step of the bifurcated tube model.

the matrix improves the convergence of linear solvers, including AMG. Therefore, we do the same for our new AMG method. Figure 4.2 shows the improvement in the overall iteration number across all three pressure systems that we see with this technique for the 100th time step of the bifurcated tube model. The cost per iteration as well as the setup costs are the same in both cases. Given that normalizing the matrix has about the same computational cost as one Jacobi iteration, which is negligible compared to the reduction in iterations the normalization yields, we will always work with the normalized matrix for the pressure systems in the following. Note that for the coupled velocity-pressure systems, we keep using the scaling described in Section 4.3.

4.5.3 Refining and Stretching the Domain

With this experiment, we will motivate why using classic one-level methods sometimes is not an option even in the segregated approach. We have already seen that they work very well for the velocity systems. Here however, we will turn to the pressure systems which are, by the nature of the Laplace operator, only weakly diagonally dominant in the best case[5].

For the purpose of this section, we will examine the matrix from the hydrostatic pressure system from the 100th time step of the bifurcated tube model. With a random vector r we generate a right-hand side $f = Ar$ and we choose $u_0 = 0$ as our initial guess for all three one-level methods under consideration here: BiCGStab, BiCGStab2 and flexible-GMRES. For comparison, in all the plots we will also show the number of iterations needed with our new AMG method used as a preconditioner for BiCGStab. The stopping criterion for every method was a relative residual reduction of 10^{-8}. At the moment, we are only interested in the convergence of the methods and only give some rough performance impressions for completeness.

Refining

One parameter that has significant influence on the convergence rate of one-level methods is the discretization size. The discretization size is obviously important as it is one of the key ingredients to obtain precise simulation results. In the industrial context, a high accuracy of the simulation can be crucial for the success of the simulation workflow. For increasingly small values of h, the convergence rate of one-level solvers

[5]I.e. if there exist no positive off-diagonal couplings.

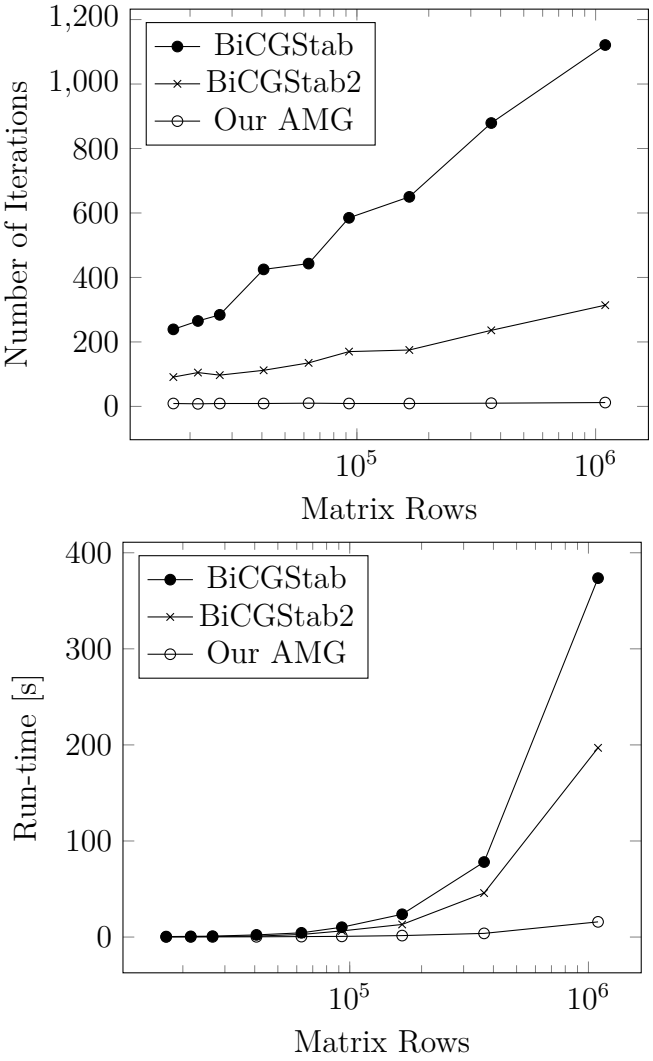


Figure 4.3: Number of iterations and run-time needed to reduce the residual by a factor of 10^{-8} for different methods and different matrix sizes for the same problem. None of the systems converged with a plain Gauss-Seidel scheme. Flexible-GMRES did not achieve the requested residual reduction within 10,000 iterations for the three finest discretizations and produced very high iteration counts already for the third system, so for the sake of readability, flexible-GMRES is not plotted here.

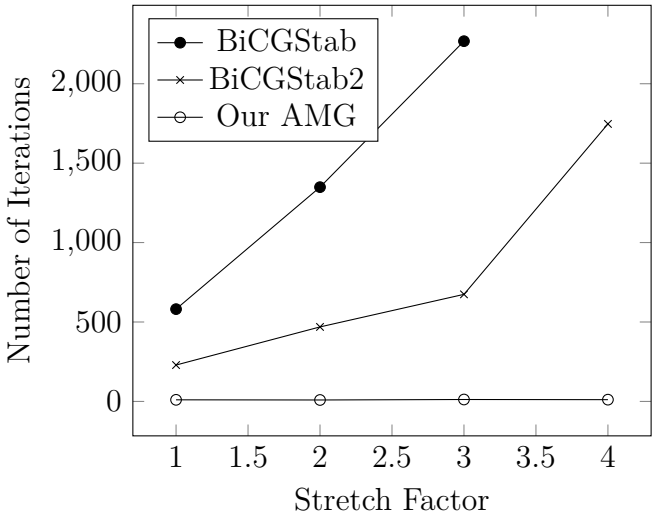


Figure 4.4: Number of iterations needed to reduce the residual by a factor of 10^{-8} for different methods and different stretch factors. None of the systems converged with a plain Gauss-Seidel scheme. Flexible-GMRES only reached the requested residual reduction within 10,000 iterations for the non-stretched problem. BiCGStab did not reach the required residual reduction for the system with stretch factor 4 within 10,000 iterations.

to tends towards 1. This phenomenon is not restricted to GFDMs, it can be found in all forms of discretization methods, which makes AMG such a successful method. AMG addresses this issue by introducing the coarse grid correction. Figure 4.3 shows how the discretization size, or *smoothing length* as we call it in the FPM, influences the convergence rates of the different linear solvers. The classical one-level methods need increasingly more iterations with every refinement step as we are refining the point cloud from $h = 0.1$ to $h = 0.02$. While refining the point cloud, the size of the matrix increases at the same rate, as every point induces one row in the pressure matrix. Therefore, every single iteration of the linear solver becomes more expensive computationally. Together with the increasing number of iterations needed to solve the system up to the (fixed) desired accuracy, this means that the cost for solving the system increases more than linearly when refining the point cloud. For our AMG method, the number of iterations needed stays constant which means that it fulfills the most basic property of a Multigrid method, cf. Section 3.3.2. Since the computational cost per iteration increases linearly with the matrix size, we can say that our method scales linearly with the matrix size as well.

Stretching

By stretching the domain in the direction of its major axis (going from left to right in Figure 4.1), we increase the distance[6] between two Dirichlet boundaries for the pressure. Just like we have seen in Section 3.3.2, Figure 4.4 shows that by doing so

[6] cf. Section 3.3.2.

the convergence rate of the one-level solvers decreases severely. The smoothing length that we used for this experiment was $h = 0.06$, leading to $67,000$ points for the non-stretched case (stretch factor 1). Regarding the number of points, the point cloud that is stretched by a factor of 3 is comparable to the one for the original model but with a smoothing length of $h = 0.04$. Both have around $160,000$ points. For the stretched model, BiCGStab needed 2268 iterations, while for the refined model it only needed 906. Similarly, the number of iterations needed by BiCGStab2 increases from 291 to 674. In all cases, our AMG method needs between 9 and 12 iterations, independently of the stretch factor or the level of refinement.

Remark 4.4. Therefore, in the context of GFDMs, AMG methods are especially useful for both very fine point clouds and thin geometries.

4.5.4 The Effect of Improving the Diagonal Dominance on Positive Couplings and AMG Convergence

As described in Section 2.4.1, the diagonal dominance of the linear systems can be improved by modifying the least squares problems that define the discrete differential operators.
Recall our measure of diagonal dominance from equation (4.20):

$$\delta_i = \frac{\sum_{i \neq j} |a_{ij}|}{|a_{ii}|}. \tag{4.22}$$

If $\delta_i < 1$, the i-th row of the matrix is diagonally dominant. A weakly diagonally dominant row has $\delta_i = 1$ and other rows have $\delta_i > 1$. In order to assess the diagonal dominance of a matrix, we can look at minima, maxima and averages of δ_i across different subsets of rows.
For the bifurcated tube model we find that without using the improvement of the diagonal dominance as described in Section 2.4.1, the average diagonal dominance in the hydrostatic pressure system is

$$\delta_{\text{avg}} = 1.361. \tag{4.23}$$

This indicates that the matrix is not diagonally dominant, which is what we expected because of the positive off-diagonal entries. The matrix has 62719 rows and 2508756 couplings, 456972 (or 18.22%) of which are positive. Out of these 62719 rows, 26412 (42.11%) have both positive and negative off-diagonals and the other rows have negative off-diagonals only. There are no rows with only positive off-diagonals, as this would contradict the consistency condition for the constant function. In Section 2.4.1 we already said that if all non-central stencil coefficients are negative, then the corresponding row must be weakly diagonally dominant. Our experiments are consistent with this claim, as for these rows we find

$$\delta^-_{\text{avg}} = \delta^-_{\text{min}} = \delta^-_{\text{max}} = 1.0, \tag{4.24}$$

where the superscript $^-$ indicates we are taking into account those rows that only have negative off-diagonal coefficients. The diagonal dominance is violated only in those

rows where positive off-diagonals occur. For those rows, our measures are

$$\delta^+_{\text{avg}} = 1.86, \tag{4.25}$$
$$\delta^+_{\text{min}} = 1.00, \tag{4.26}$$
$$\delta^+_{\text{max}} = 3.06, \tag{4.27}$$

where the superscript $^+$ indicates that only rows with at least one positive off-diagonal coefficient are taken into account.

Through the modifications to the least squares problem that increase the absolute value of the central stencil coefficient, we can decrease the number of positive off-diagonal entries in the matrix to 33378 (1.33%). At the same time, the number of rows with both positive and negative couplings decreases to 9642 (15%). This improves the average diagonal dominance to

$$\hat{\delta}_{\text{avg}} = 1.13. \tag{4.28}$$

The diagonal dominance measures in the rows with only negative couplings obviously do not change, but we find that $\hat{\delta}^+_{\text{max}}$ improves to 2.20

By increasing the diagonal dominance in that manner, we reduce the number of iterations needed in the hydrostatic pressure system from an average of 7.6 down to 6.8 in the first 100 time steps of the bifurcated tube model. In the correction pressure system, the number of iterations decreased from 13.5 to 10.8 and for the dynamic pressure the difference was marginal (11.81 vs. 11.76). In the latter case the convergence rate did improve from 0.261 to 0.211, but this improved convergence rate did not always lead to less iterations.

The modified least squares problem that gives discrete operators with a better diagonal dominance requires a little more computational effort compared to the base version. In the 100 time steps run in this experiment, the average time per point and time step[7] needed to set up the discrete operators was $0.211129 \cdot 10^{-6}$ s in the base version and $0.212817 \cdot 10^{-6}$ s in the enhanced version, which is a difference of $< 1\%$ and is therefore negligible.
The differences in the run-time of our AMG solver were also negligible for the hydrostatic and the dynamic pressure system. For the hydrostatic pressure system this is because the slight improvement in the number of iterations does not pay off as much because the hydrostatic pressure system involves the AMG setup. For the dynamic pressure system we did not observe much of an improvement in the iteration numbers in the first place, which is why we also cannot expect to gain any benefit in the run-time.
On the other hand, the correction pressure system had the most notable improvements regarding the number of iterations, which also gives the best improvement in performance: The average time for solving the correction pressure system per point and

[7] We are reporting per point timings here as the number of points between the two simulations differed by about 10%.

Neighbors	40	30	20
BiCGStab	317 / 519 / 610	302 / 537 / 590	310 / 710 / 827
BiCGStab2	145 / 194 / 204	183 / 224 / 186	166 / 250 / 258
AMG	4 / 9 / 12	4 / 8 / 10	6 / 8 / 11

Figure 4.5: Iteration counts (hydrostatic pressure system / correction pressure system / dynamic pressure system) for different solvers and neighborhood sizes. The smoothing length was fixed at $h = 0.06$.

time step decreased from $0.5180 \cdot 10^{-5}$ s to $0.4412 \cdot 10^{-5}$ s, which is an improvement of 1.31x.

4.5.5 Different Neighborhood Sizes

When thinking about the performance of a linear solver in terms of run-time, it is not only important how many iterations are needed to solve the linear system to the desired accuracy, but it is also important how much run-time is needed for one single iteration. One very important aspect for this run-time is how many non-zero entries there are in a row, or in the matrix as a whole.
In FPM, this number is typically fixed to 40 entries in a row, since every point is limited to 40 neighbors, see Section 2.2.4. At and next to the boundary, this value can be slightly lower, but the majority of points will have 40 neighbors. Recall that this value was chosen in order to heuristically enable every point to have enough neighbors in each direction so that a second order discretization for the Laplacian can be found. Note that even with 40 neighbors, it is not guaranteed that this is possible. But 40 neighbors have been found to be sufficient in most situations [126].
We may argue though that there are situations where we can assume that the point cloud does not degenerate too much so that less than 40 neighbors are sufficient. In our bifurcated tube model for example the points are flowing in one direction without any free surfaces or other possibly generated geometric features. Figure 4.5 shows how the iteration counts of the linear solvers for the three pressure systems (hydrostatic, correction, dynamic) are affected by having less neighbors available. Note that the two BiCGStab variants are affected quite heavily, especially in the dynamic pressure system. Our AMG method proves to be much more stable in this experiment.

Remark 4.5. Although the matrices for the linear systems for each of the pressure systems are identical if the boundary conditions for the dynamic pressure systems are unchanged, the number of iterations needed to converge varies. This is mostly because the initial residual is different, as the right-hand sides and initial guesses differ. Take the hydrostatic pressure system for example: Here, the only external force influencing the right-hand side is the gravity, which is constant across all time steps. Therefore, the solution to the hydrostatic pressure system from time step t will be a good initial guess for the hydrostatic pressure system in time step $t + 1$. On the other hand, both the right-hand sides for the correction pressure and the dynamic pressure depend on the velocity in time step $t + 1$, therefore the right-hand side changes more compared to the hydrostatic system and so does the solution.

Neighbors	40	30	20
BiCGStab	3.202 / 6.211 / 6.976	2.374 / 4.296 / 4.782	1.872 / 4.453 / 4.870
BiCGStab2	3.390 / 4.490 / 4.209	2.551 / 3.199 / 2.931	1.695 / 2.634 / 2.784
AMG	0.336 / 0.422 / 0.500	0.242 / 0.297 / 0.320	0.188 / 0.180 / 0.219
AMG setup	0.234 / 0.219 / 0.219	0.164 / 0.156 / 0.156	0.117 / 0.102 / 0.094

Figure 4.6: Run-time in seconds (hydrostatic pressure system / correction pressure system / dynamic pressure system) for different solvers and neighborhood sizes with smoothing length $h = 0.06$. The AMG timings in the first part include the setup times. In the second part, the setup times are given separately.

Looking at the run-times in Figure 4.6 we find that the solver cost decreases proportionally to the number of neighbors. In most cases with the two BiCGStab solvers, this decrease in cost per iteration more than makes up for the amount of additional iterations needed in the cases with fewer neighbors. The AMG method also benefits from having less non-zero entries in the matrices.

Remark 4.6. From the run-times for our AMG method we can see that

1. One BiCGStab2 iteration is twice as expensive as one BiCGStab iteration, which is in line with what can be expected from looking at the algorithms.

2. For the pressure systems, which typically need very few AMG iterations only, the setup cost makes up for about half the overall run-time cost.

3. The AMG setup is about 20 times as expensive as one BiCGStab iteration.

4. Without the setup cost, one AMG iteration is about 2-4 times as expensive as one BiCGStab iteration.

Therefore, it is worth thinking about how to re-use the AMG setup, see Section 4.5.6.

Remark 4.7. Using less neighbors per point would not only lower the cost per iteration for the linear solver, but it would also be beneficial for other parts of the code, for example setting up the differential operators would also become cheaper as the corresponding least squares problems would be smaller.

If the run-time of the linear solver was the only variable we would have to keep in mind, the conclusion would be to keep the number of neighbors as low as possible. However, there are other, more complicated models where 20 neighbors per point are not enough to ensure a second order discretization everywhere in the domain. There are two options that can be considered regarding the neighborhood size:

- Starting with a certain number of neighbors, say 40, and if the model is stable, decrease the number of neighbors. The number of points where a second order discretization is not possible can easily be determined after solving the least squares systems. If for some neighborhood size this number gets too high, the size needs to be increased again. This trial-and-error method would be part

of the overall simulation workflow. For certain types of applications, the most appropriate setting can then be determined from experience.

- A more flexible option is to dynamically increase or decrease the number of neighbors *locally* for every point. Since after solving the least squares system at a point x_i it is easy to determine whether a second order discretization has been found or not, we can increase the number of neighbors in case the discretization is not of second order. This means though that it would be necessary to compute the possible neighbors again and choose a (larger) subset of neighbors to then solve the least squares problem for the second time. It might happen that this again does not lead to a second order discretization. Therefore, this process must be limited to a certain number of *neighborhood extensions*.

While the second option seems much more convenient and can also lead to a much more efficient overall method, as not all the points would need 40 neighbors, this thesis is limited to a fixed number of neighbors and if not stated otherwise, 40 neighbors have been used in the experiments. Keep in mind though, that our AMG method also to use less than 40 neighbors without any negative impact on the solver performance, which is not always the case with BiCGStab and BiCGStab2.

4.5.6 Setup Re-use

All AMG methods inherently consist of two phases: The setup phase and the solution phase. Depending on the specific AMG strategy and the type of linear system to be solved, the ration in terms of computational cost between these two phases varies. As we have seen in the previous section, for a Poisson-like pressure system with suitable AMG techniques, the number of iterations needed to converge is pretty small and therefore the setup phase is about as computationally intensive as the actual solution phase.

Typically, this observation is amplified when working with multiple cores: The AMG setup phase does not scale as well with the number of processes p as the solution phase, which mainly consists of matrix-vector products, does.

For other types of linear systems, the ratio can be different, mainly when the convergence rate is not as good as in those Poisson-like applications and therefore the number of iterations needed increases. This is the case for example in many elasticity applications, see for example [7].

It is obvious though that in any case the number of setups in a transient simulation should be kept as low as possible. As discussed earlier in this chapter, re-using the setup comes at the cost of a potentially deviating convergence rate in the linear system for which we re-used the setup of a previous system. The task is therefore to balance the benefits of not having to compute a new setup versus the increasing number of iterations needed to solve the system and the impact on the quality of the numerical solution of the simulation. The answer to this highly depends on the types of linear

systems and the application behind them, so we will examine this in the case of the bifurcated tube model with the FPM in this section.

Our considerations on this matter are twofold: First, we will discuss how the setup for the hydrostatic pressure system can be re-used within the same time step. Because there is only one velocity system (or a coupled velocity-pressure system in the coupled approach), the velocity (or velocity-pressure) setup cannot be re-used within a time step. Secondly, we will have a look at how it is possible to re-use the setup across time steps. In Section 4.4 we discussed that our strategy to do so is to skip the point cloud management phase for a given number of time steps. Here, we will examine the numerical effects of this strategy with respect to the quality of the solution of the overall method and the impact on the convergence of AMG when re-using the setup and also the possible computational benefits.

Re-using the Setup Within a Time Step

In order to decide whether or not it is possible to re-use the setup within a time step we first need to comment on the coherence of the three pressure systems that are being solved. Observe that in the three pressure equations

$$\operatorname{div}\left(\frac{1}{\rho}\nabla p_{\text{hyd}}^{n+1}\right) = \operatorname{div}\hat{g}, \tag{4.29}$$

$$\operatorname{div}\left(\frac{\Delta t}{\rho}\nabla p_{\text{corr}}^{n+1}\right) = \operatorname{div}\tilde{\mathbf{v}}^{n+1}, \tag{4.30}$$

and

$$\operatorname{div}\left(\frac{1}{\rho}\nabla p_{\text{dyn}}^{n+1}\right) = \operatorname{div}\tilde{g}, \tag{4.31}$$

the left-hand sides are identical, if we carry Δt in equation (4.30) over to the right-hand side.
This holds true as long as all three systems obey to the same boundary conditions. For the reasons given in Remark 2.13 we may encounter different boundary conditions in the dynamic pressure system which then leads to a different matrix.

For now, we want to assume that all three systems have the same boundary conditions. Then, according to equations (4.29)–(4.31), the three linear systems

$$A_{\text{hyd}}p_{\text{hyd}} = f_{\text{hyd}},\ A_{\text{corr}}p_{\text{corr}} = f_{\text{corr}},\ A_{\text{dyn}}p_{\text{dyn}} = f_{\text{dyn}} \tag{4.32}$$

all share a common matrix

$$A = A_{\text{hyd}} = A_{\text{corr}} = A_{\text{dyn}}. \tag{4.33}$$

Since the AMG setup phase only depends on the matrix and not on the right-hand side or the initial guess, the setup that has been computed for the hydrostatic pressure

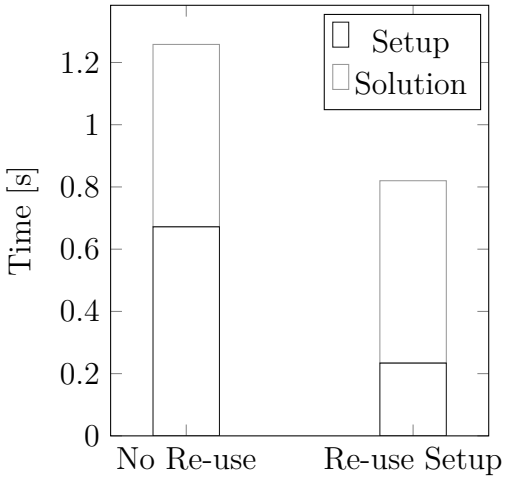


Figure 4.7: Overall run-time of the AMG method in one time step with and without re-using the setup from the hydrostatic pressure system.

system (4.29) can trivially be re-used for the other two systems.

For the bifurcated tube model with 40 neighbors per point as discussed in Section 4.5.5, Figure 4.7 shows the gain in run-time when re-using the setup from the hydrostatic pressure system for the other two pressure systems. Here, the numbers from Figure 4.6 were used. By computing only one setup instead of three, we can save 35% of the overall linear solver run-time in this case. This observation is typical for the setup re-use in the pressure systems within a time step, because the cost of a setup and a solution phase are approximately the same and one time step consists of three solution phases and three setups, two of which can be re-used. Therefore, the approximate theoretical decrease in computational cost is 33%.

The situation becomes more involved if for the reasons given in Remark 2.13 the boundary conditions in the dynamic pressure system differ from the ones in the other two systems. The most common situation is that the boundary conditions in the dynamic pressure system are changed from a Dirichlet or Neumann boundary condition to a mixed boundary condition or no boundary condition (cf. Remark 2.13). In this case we need to re-run the detection algorithm for independent subsystems (cf. Section 5.2) and we are forced to compute another setup just for the dynamic pressure system. The benefit of re-using the hydrostatic pressure setup then reduces to re-using the setup in the correction pressure system, meaning that instead of saving $\approx 33\%$ of the linear solver run-time, the setup re-use only saves $\approx 16.5\%$.

Re-using the setup across time steps

As discussed in Section 4.4, in order to re-use the setup across two or more time steps, we skip the point cloud management for this number of time steps. In this section, the first two time steps always include point cloud organization and the setup phase for

the hydrostatic system. Then we skip the point cloud organization and also the AMG setup for a predefined number of time steps. With "Skip n" we denote that the point cloud organization only takes place in every n-th time step, i.e. "Skip 1" means that the point cloud organization is not skipped at all, whereas "Skip 3" means it takes place in every third time step.

Figure 4.8 shows that the number of AMG iterations needed to solve each of the pressure system changes when the setup is skipped for multiple time steps. Although the iteration number slightly changes in all cases, especially the "Skip 4" and the "Skip 5" case shows a noticeable increase in the overall iterations needed in the 10 time steps under consideration here. Recall that this is because although the point cloud management is deactivated, the differential operators are still constructed again in every time step. While the neighborhood is fixed because the point cloud management has been skipped, the least squares problem will still yield a slightly different operator at every point as the points have moved. Therefore, the pressure matrices A^{t+1} are not perfectly identical to the ones from the previous time step. But because we want to re-use the setup, we use the interpolation, restriction and coarse level operators constructed for A^t to solve the system $A^{t+1}p = f$. A decrease in the convergence rate is therefore something we would expect.

In Figure 4.9 we see that from a performance point of view, re-using the setup for the pressure across time steps does not yield much of a benefit, as the re-use within the time step already decreased the ratio of setup cost versus solution cost by quite a bit. The first bar shows that with the inner time step setup re-use, the setup phases only make up about 25% of the overall linear solver run-time. In addition to that, the slightly increasing number of iterations needed to converge makes the solution phase more expensive when re-using the setup. Therefore, re-using the setup across time steps when using the segregated approach often does not pay off, especially considering the numerical effects the skipping of the point cloud management has, see below. Section 4.5.7 will show though that this can be different in the coupled approach.

However, skipping the point cloud management not only allows to re-use the AMG setup when solving the linear systems, but it also saves some computational effort in the FPM itself. Figure 4.10 shows that the benefit of skipping the point cloud management phase has a very noticeable effect on the "organize" part, which includes moving the point cloud and the geometry, setting up differential operators and others. In fact, the benefit here even outweighs the benefit that we can achieve in our AMG method by re-using the setup. This is a nice extra benefit of the new re-use strategy developed here.

Skipping the point cloud management phase in a time step means that in this time step no points are introduced or removed from the point cloud. Especially for problems with an inflow boundary, not introducing new points in some of the time steps leads to a distorted point cloud at the inflow. This certainly leads to some numeric noise in the result and it depends on the kind of question that a simulation is supposed to answer whether this noise is acceptable or not. For the bifurcated tube model, we chose to examine the velocity at the outflow boundary. Figure 4.11 shows that despite skipping

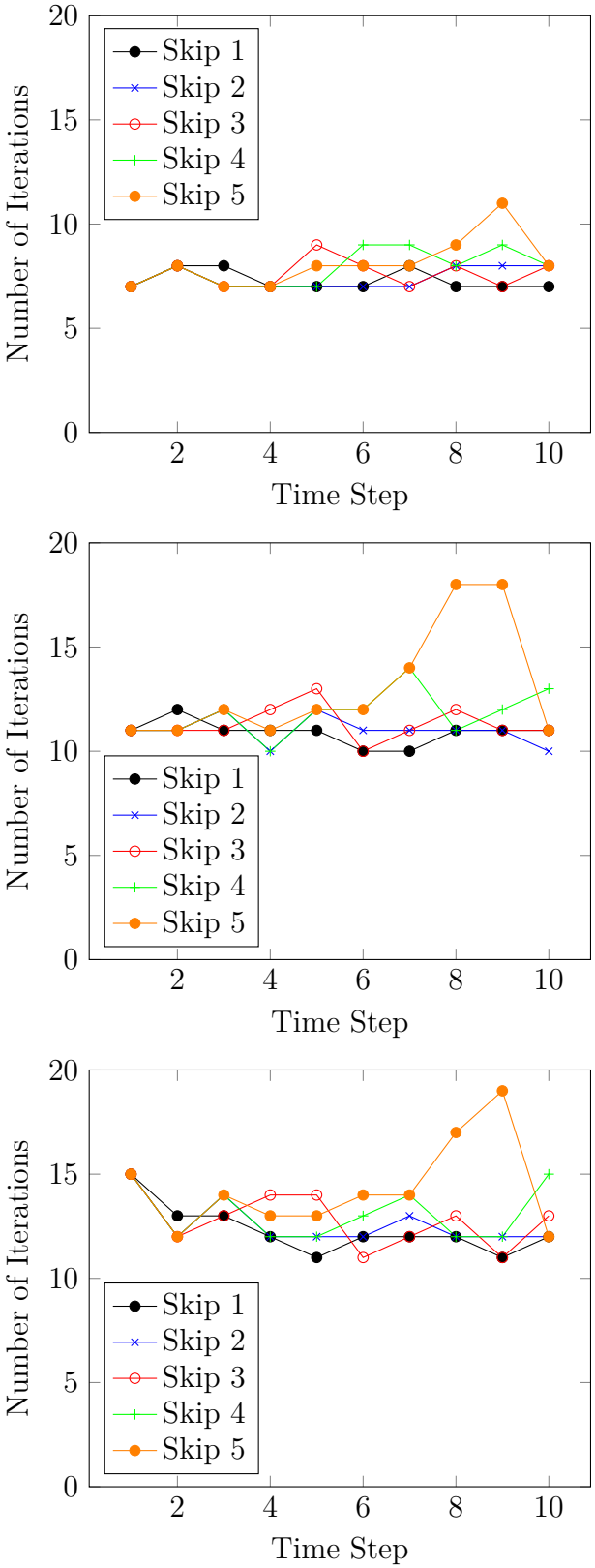


Figure 4.8: Iteration numbers for the three pressure systems (hydrostatic, correction, dynamic; from top to bottom) for 10 time steps with different frequencies of setup re-use.

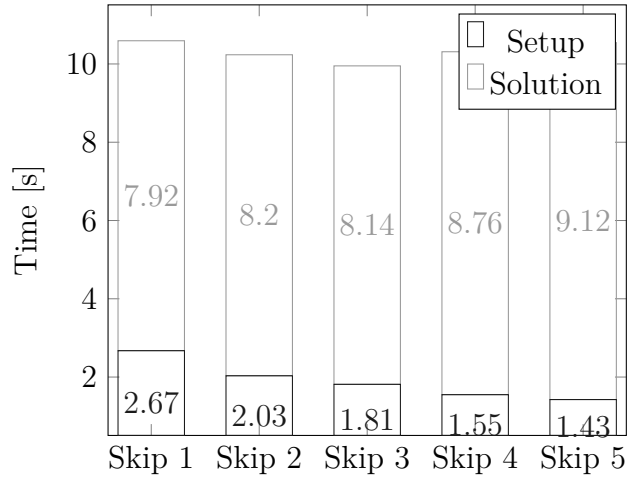


Figure 4.9: Solver run-time in the setup and the solution phase for 10 time steps (pressure systems) of the bifurcated tube model with different "Skip" settings. The setup is being re-used within every time step in all cases.

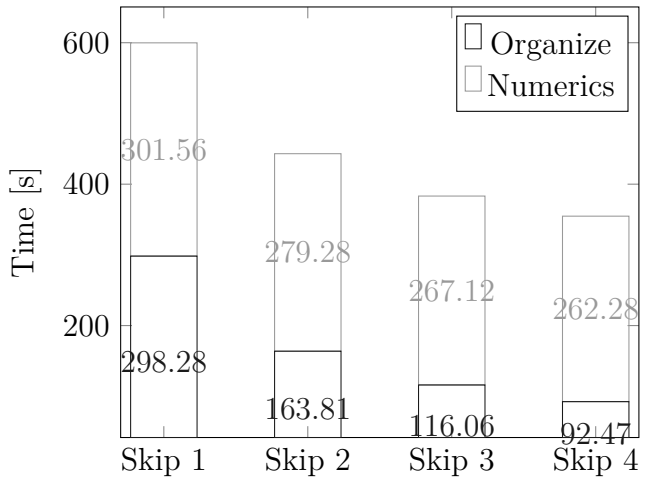


Figure 4.10: Ratio between the organization part in the FPM and the numerics part for 1000 time steps with different frequencies of point cloud management. For the "Skip 5" experiment, the physical results were not acceptable, therefore the timings are not shown here.

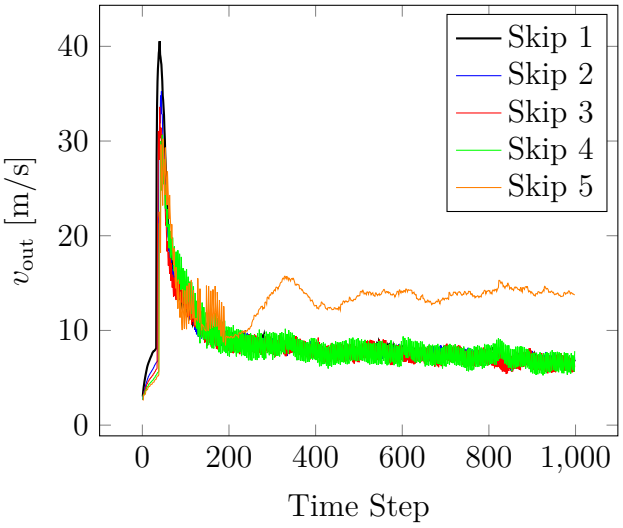


Figure 4.11: Velocities at the outflow for different setup re-use intervals.

the point cloud management phase the simulations "Skip 1" to "Skip 4" qualitatively show the same behavior[8]. The simulation which organizes the point cloud only in every 5th time step does not yield a result that would be acceptable in this case. Therefore, we do not consider the "Skip 5" simulation from here on. It can also be seen though that while the graph is fairly smooth for the simulation without skipping ("Skip 1"), the other simulations oscillate around this graph, cf. Figure 4.12. Now, if the goal of this simulation was to measure the overall outflow from the model over a certain period of time, those oscillations would probably be acceptable. If however the outflow velocity at every given time step is important, for example in order to couple this simulation with some other simulation at the outflow boundary, then these oscillations are not acceptable. Techniques like filtering the velocity signal would be an option in this case, but the point here is that depending on the specific use case of the simulation, skipping the point cloud management and re-using the setup may or may not be an option.

4.5.7 Coupled Systems: BiCGStab2 vs. ILU-Smoothing vs. Saddle Point Approach

In order to examine the coupled approach for the bifurcated tube model, we need to change the model a bit. So far, we chose the viscosity so that the Reynolds Number was 1000. Now, we examine a much more viscous fluid so that the Reynolds Number becomes $Re = 0.001$. Therefore, as we have seen in Section 2.3.4, the coupled approach in the FPM is mandatory.

[8] We attribute the spike in v_{out} at the beginning to the non-physical initial condition of $v_0 = 1.0$ everywhere in the domain.

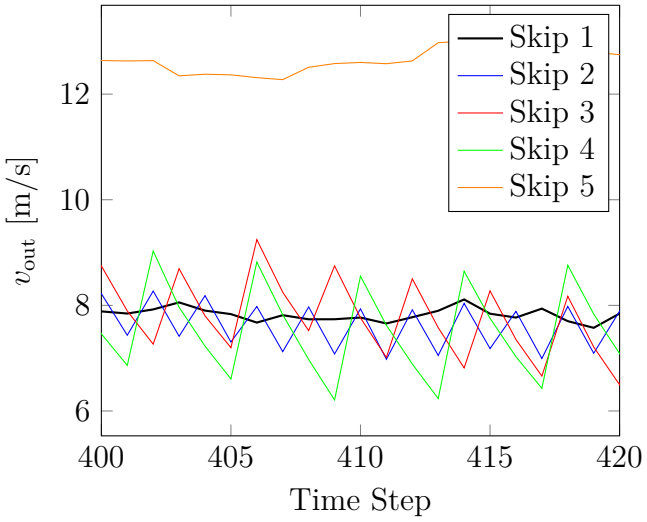


Figure 4.12: Velocities at the outflow for different setup re-use intervals (zoomed).

We want to study the influence of the virtual time step size on the result of the simulation. To this end, we carry out the same simulation with $\Delta t_{\text{virt}} = 0.1\Delta t$, $\Delta t_{\text{virt}} = 0.05\Delta t$, $\Delta t_{\text{virt}} = 0.01\Delta t$ and $\Delta t_{\text{virt}} = 0.005\Delta t$. The smoothing length is fixed at $h = 0.06$, leading to around 60k points. We also used the same inflow velocity of $v_0 = 1.0m/s$ as before and we are interested in the velocity at the outflow.

For the four different virtual time step sizes, we test the following linear solvers:

1. BiCGStab2,
2. A standard unknown-based AMG method [117],
3. Like (2), but with ILU-smoothing, cf. Section 4.3,
4. The Saddle Point AMG method described in Section 4.3 without the stabilized interpolation. In this section, we will base the coarsening of p in the pressure submatrix D only, rather than computing $C\hat{\mathbf{A}}^{-1}B + D$ (cf. Section 3.3.5).

In neither of the settings, BiCGStab2 was able to reduce the residual (cf. Remark 4.3) below 1.0. We therefore omit BiCGStab2 in the following.

Figure 4.13 shows that the velocity at the outflow boundary is quite different for $\Delta t_{\text{virt}} = 0.1\Delta t$ and $\Delta t_{\text{virt}} = 0.05\Delta t$ and only seems to settle in for $\Delta t_{\text{virt}} = 0.01\Delta t$ and $\Delta t_{\text{virt}} = 0.005\Delta t$. For this reason, the maximum virtual time step size that should be used in this model is $\Delta t_{\text{virt}} = 0.01\Delta t$.

Figures 4.14 and 4.15 however show that with $\Delta t_{\text{virt}} = 0.01\Delta t$ and $\Delta t_{\text{virt}} = 0.005\Delta t$ the number of iterations needed to converge for all three solvers increases dramatically, yielding longer run-times in turn. Because of the saddle point structure of those systems, the plain unknown-based AMG method does not converge. In order to improve

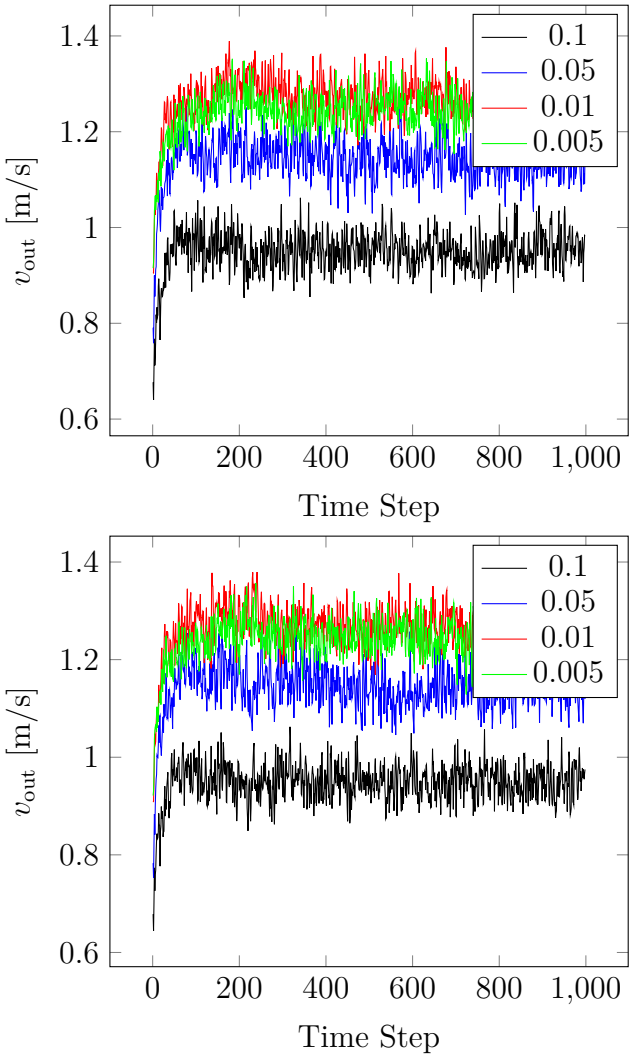


Figure 4.13: Velocities at the outflow for different virtual time step sizes with our saddle point solver (top) and the unknown-based AMG with ILU-smoothing (bottom). The two solutions are in good agreement, as should be expected with a reasonable small stopping criterion for the linear solver.

$\Delta t_{\text{virt}}/\Delta t =$	0.1	0.05	0.01	0.005
Saddle Point AMG	17.9	23.4	57.7	90.1
unknown-based AMG	3.3	7.5	-	-
unknown-based AMG + ILU-smoothing	5.3	6.1	26.0	79.0

Figure 4.14: Average number of iterations for the coupled velocity-pressure system. A dash indicates that the method did not converge.

$\Delta t_{\text{virt}}/\Delta t =$	0.1	0.05	0.01	0.005
Saddle Point AMG	1.29	1.42	2.32	3.12
unknown-based AMG	0.78	0.91	-	-
unknown-based AMG + ILU-smoothing	0.89	0.95	1.21	1.95

Figure 4.15: Average run-time for the coupled velocity-pressure system in seconds. A dash indicates that the method did not converge.

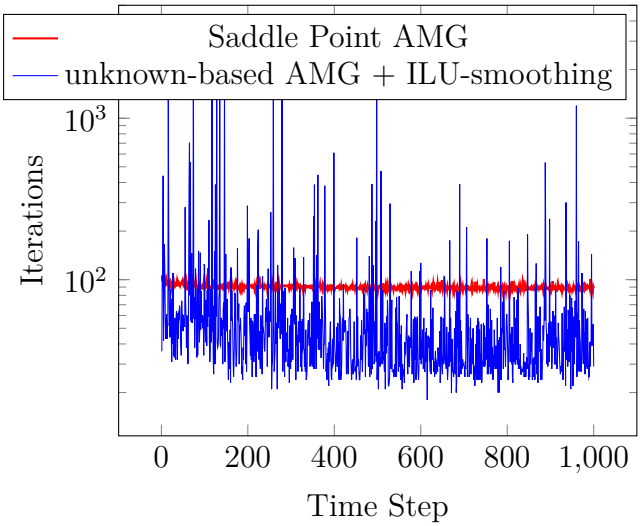


Figure 4.16: Number of iterations for the coupled velocity-pressure system per time step for the Saddle Point AMG and the unknown-based AMG with ILU-smoothing in the $\Delta t_{\text{virt}} = 0.005\Delta t$ case.

the convergence rate of the Saddle Point AMG method, we can use a coarsening for the pressure based on $C\hat{\mathbf{A}}^{-1}B + D$, but this would significantly increase the computational cost of the setup which does not pay off in this case. An example of when the more sophisticated Saddle Point AMG techniques are mandatory can be found in [99].

While the unknown-based AMG method with ILU-smoothing is faster compared to the Saddle Point AMG approach, it shows a lot more variance in the convergence rate, as depicted in Figure 4.16. In one of the 1,000 time steps, the method did not converge within 3,000 iterations. In this case, the FPM implementation we are using here falls back to using the old velocity solution from time step $t - 1$ to move the point cloud and then starts a new time step with a new linear system. Here, the unknown-based AMG method with ILU-smoothing was able to solve that new linear system again and the simulation went on. And as we have seen in Figure 4.13, this process did not have any severe impact on the simulation. If however the linear solver cannot solve the system for ten time steps in a row, the simulation is stopped.
Our conclusion from Figure 4.16 is that while the unknown-based AMG with ILU-smoothing is faster in the presented case, it is also less robust. Thus, the general

recommendation is to use the specialized Saddle Point AMG approach from Section 4.3 for the coupled velocity-pressure systems.

Re-using the Setup for the Coupled Velocity-Pressure System

Re-using the setup for the coupled velocity-pressure system is also possible, if the point cloud management step can be skipped, as in the high Reynolds number case. In the $\Delta t_{\text{virt}} = 0.01\Delta t$ case the average number of iterations needed increases to 59.6, 61.0 and 62.0 when re-using the setup for 2, 3 and 4 time steps respectively. But at the same time, the average run-time goes down to 2.01s, 1.93s and 1.89s. Together with the benefits of not having to perform the point cloud organization, the fastest method was 1.23x faster compared to the one without any re-use. Here, the impact of not organizing the point cloud in every time step is not as big as in the segregated case, because the linear solver takes up more of the overall run-time in comparison to the organization part. More specifically, the linear solver takes up about 75% of the simulation run-time, when re-using the organization and the setup. Most of this time is spent in the iteration phase, which is also different from the segregated case, where the ratio between the setup phase and the iteration phase is about 1:1.

The next chapter will introduce the techniques that are influenced by the parallelization of our method.

Chapter 5

Parallelization

Now that we have established an efficient AMG method for GFDMs, this chapter deals with the challenges arising when introducing parallelism to the linear solver. For the industrial size problems that we are interested in, parallelism is an indispensable part of the algorithm.

We therefore start this chapter by discussing the modifications to the serial AMG method that are necessary in order to parallelize the algorithm. Then, we turn to the issue of finding independent subsystems in the matrix, which can occur in the FPM and may be singular with a constant kernel. If those systems are singular, they can severely impact the convergence of the overall method, even leading to divergence. We overcome this issue by separating them from the linear system and solving them separately with a direct solver, fixing one variable to 1. Hence, we need to find those subsystems, but we need to do so in parallel because reducing the adjacency graph to just one process would be too expensive. With that we must be careful not to affect the linear scaling properties that AMG has. This chapter will show that the algorithm we develop has a linear complexity in the average case and the experiments that we conduct confirm that.

Another interesting aspect of the parallelization is the idea of renumbering the linear system. Since the points in the FPM point cloud are enumerated almost randomly, especially after having moved for a couple of time steps, the non-zero entries in the linear system matrix are distributed across the whole bandwidth, leading to a far from optimal computational performance. We examine this issue by renumbering the rows within and across processes. The results show that such renumbering strategies can lead to better run-times in some, but not all, cases.

We conclude the chapter with some experiments showing the parallel behavior of our newly developed AMG method. We also show how the aforementioned algorithm for finding the independent subsystems behaves in this context. Both algorithms show a good parallel speed-up.

After this chapter we have all the ingredients to run real world industrial size examples with the FPM using our linear solvers. The final chapter 6 will give a number of examples for that, comparing the computational cost of the simulation with and without using our new AMG method.

5.1 General Remarks on Parallel AMG

Throughout this chapter, we will use P to denote the number of processes involved in any algorithm. Note that we only discuss the distributed memory parallelization with MPI here. Shared memory models like OpenMP are not discussed. On the AMG side, we used the AMG library SAMG developed at Fraunhofer SCAI [139]. A quite deep analysis of the parallel features of AMG and the SAMG library in particular can be found in [112]. On the FPM side, we use an implementation of the FPM developed at the Fraunhofer ITWM.
Note that other parallel implementations of AMG are available. One that focuses specifically on parallel scalability is BoomerAMG [56]. Both [112] for the SAMG library and [170] for BoomerAMG go into detail about the difficulties of parallelizing AMG, so we will only go into some aspects here that affected our work.

In SAMG, every process p_i holds only its own share $\Omega_{p_i} = \{\min_i \leq k \leq \max_i\}$ of the matrix rows, i.e. SAMG works with a *row-wise* decomposition of the matrix. Moreover, each process p_i has the necessary overlap information for those coefficients in row k that couple to a row that belongs to process p_j.
The main algorithmic difficulties when parallelizing AMG occur in the smoother and the coarsening. Furthermore, the computation of the Galerkin operators and the (direct) solution of the coarsest system are challenging in terms of computational efficiency.

Lastly, note that many of the aspects covered in this section are directly related to the renumbering strategies discussed in Section 5.3 as the number of couplings between two processes can be reduced by those strategies, bringing the parallel algorithms closer to the serial ones numerically.

5.1.1 Smoothing Schemes

The first component of AMG that needs to be considered when parallelizing the method is the smoothing scheme. For the pressure systems, we use the classical Gauss-Seidel method. Gauss-Seidel is intrinsically serial as the update of a variable x_i depends on the updated values of x_j with $j < i$ in the current iteration. Running the Gauss-Seidel algorithm in serial in order to preserve its exact formulation is not an option, as the smoothing step is one of the main performance bottlenecks in any AMG method. We therefore need a parallelizable option.
A natural choice here is to use a classical Gauss-Seidel algorithm for those couplings that are local to one process and to use values from the previous Gauss-Seidel iteration for the couplings to rows on different processes. In this way, we only need to update the remote values once after every Gauss-Seidel iteration. The resulting method is a hybrid Jacobi/Gauss-Seidel method where remote couplings are treated like in a Jacobi method and the local couplings are updated in a Gauss-Seidel fashion.
This method does not yield the same convergence as a pure Gauss-Seidel method; the impact is generally limited, though. It is worth noting that this also means that the convergence of our smoothing scheme – and therefore of the whole AMG method

– depends on the number of processes used. Again, the influence of the number of processes is limited in most cases, but may cause the number of iterations necessary to reach the desired accuracy to change by 1 or 2 from time to time.

For the coupled velocity-pressure systems, we use either ILU- or Uzawa smoothing. ILU again is an inherently non-parallel method, as it is based on an LU-decomposition. We take a straight forward parallelization approach that again has influence on the numerics of the method, but provides the best possible scalability: Each process computes a local LU-decomposition for its local matrix, i.e. ignoring all remote couplings. Other options would include using the same strategy but with a certain overlap to other processes. This requires data exchanges at the process boundaries, which can be done in a Jacobi fashion for example, i.e. before every smoothing step. Another option is to parallelize ILU using wavefronts [92], which induces a significant amount of communication and ILU setup cost, though. In the models where ILU smoothing worked in serial, we did not experience any problems with our straight forward parallelization, so we did not look into these more sophisticated methods.

The Uzawa smoother that we use in the Saddle Point AMG approaches can be parallelized using similar techniques as we used for Gauss-Seidel and ILU, see [98] for details.

5.1.2 Coarsening

The Ruge-Stüben coarsening process is also serial in nature, as the choice of the next coarse level variable depends on the updated measure μ (cf. [138]) which is computed after choosing the previous coarse level variable. There are specialized parallel coarsening strategies available, like PMIS [170] and subdomain blocking [74]. While the former can cause a significant increase of the convergence rate, the latter can lead to high complexities in terms of memory, i.e. fairly dense coarse levels. Therefore, we use a different technique: On the first three levels, each process computes the C/F-splitting locally for its rows, treating couplings to other processes as weak couplings. On the fourth level, the matrix is agglomerated on a single process and the C/F-splitting is computed on a single process. The splitting is then redistributed to the other processes and the interpolation operator[1] as well as the Galerkin operator are computed in parallel. This is necessary in order to avoid large coarsest levels: Because each process only coarsens locally, the coarsening rate will decrease when only very few local couplings are left. Even if every process ends up with say 100 variables on the coarsest level, this would still imply a large coarsest level globally, when 256 processes are used, for example.
These situations do not occur with our method. The level on which the system is agglomerated and coarsened serially can be changed of course, but using the parallel coarsening on the first three levels has proven to be an efficient choice.

[1] Using standard interpolation and multi-pass interpolation [138] as discussed in Sections 3.3.3 and 4.2.2.

5.1.3 Galerkin Product

In SAMG, the Galerkin product $A_H = RA_hP$ is not influenced by the parallelization algorithmically, but note that computing a triple matrix product efficiently is not trivial in serial and especially in parallel, see for example [12]. The optimal algorithm to compute the Galerkin product highly depends on the matrices involved and the underlying hardware. In our method we compute $B = RA_h$ first and then $A_H = BP$. For comparatively dense interpolation operators and stencils as they occur in GFDMs, this is computationally more efficient than computing a triple matrix product directly. More recently, sparsification algorithms [37] have been added to the library, which aim to generate a sparser coarse level while preserving the spectral radius of the Galerkin operator. These approaches will not be covered in this thesis though.

5.1.4 Coarse Level Solvers

Regarding the coarse level solver, recall that with our parallel coarsening strategy we make sure that the coarsest level stays sufficiently small to be solved directly. As discussed in Chapter 4, this is done by either PARDISO or a classical LU-decomposition. For the PARDISO, the coarse level matrix needs to be agglomerated to one process in the setup phase in order to compute a PARDISO-setup. In each iteration, the coarse level right-hand side also needs to be agglomerated to that process and the coarse level system is solved there and then the solution is redistributed to the other processes. The LU-decomposition implementation that we use is fully parallel in both finding the decomposition during the setup phase and applying it later on in the solution phase. In Section 5.4.2 we conduct experiments showing which option is better in which situation.

5.2 Components

The moving point cloud implies a varying neighborhood structure in time. The local changes in the neighborhoods can lead to *groups* (or *components*, see Section 5.2.1) of points that are *independent* from the main point cloud. This means that there are no connections to or from those independent groups of points. There are various reasons for this phenomenon, see Section 5.2.3. In the end, in all cases we end up with a linear system that can be decomposed into multiple subsystems that are independent from each other. We can, or in some cases have to, take advantage of that property when solving the linear system. To this end, we use the algorithm described in Section 5.2.4 which finds such independent subsystems, even if they are distributed across multiple processes. The complexity of the component detection algorithm is analyzed in Section 5.2.5. After we have found the independent subsystems, we can treat those that have a zero row-sum and are hence are singular, separately. It is crucial to extract those subsystems before passing the system to the AMG algorithm because a special treatment is needed for those subsystems, see Section 5.2.6. Just passing the full system to AMG can lead to a severe increase in iterations required to converge or even divergence, as the experiments in Section 5.4.3 show. In order to describe the

algorithm used to detect the components, Section 5.2.1 introduces the necessary terms and propositions from graph theory.

5.2.1 Graph Theory Basics and Notation

In order to detect independent groups of points within the point cloud $\mathcal{P}$, we examine the graph $\bar{G}(A)$ associated to the matrix A that represents the linear system $Ap = g$ that is used to solve for one of the pressure fields needed in the FPM.

Definition 5.1. A *graph* is a pair $G = (V, E)$ of two sets V and E, where the elements of V are called *vertices* and the elements of E are called *edges* and

- $E \subset \{(v, w) : v, w \in V\}$ for *directed* graphs and
- $E \subset \{\{v, w\} : v, w \in V\}$ for *undirected* graphs

Another terminology from graph theory we need is the definition of *paths*:

Definition 5.2. A vertex v in a digraph $G = (V, E)$ is *connected* to vertex w via a *path* of length l if there exists a series of edges

$$e_1, e_2, \ldots, e_l, \quad e_k \in E \tag{5.1}$$

where $e_k = (v_{k-1}, v_k)$, $v_0 = v$, $v_l = w$. For undirected graphs, we use the same definition but with $e_k = \{v_{k-1}, v_k\}$.
If such a path exists for some $l > 0$, then i is *connected* to j.

Definition 5.3. In the context of this work, edges do not have a weight assigned to them, or equivalently, all edges have the weight 1. Hence, there exists a minimal distance between two connected vertices, which is the length of a shortest path between those two. The longest distance between any two vertices in G is called the *diameter* $\mathrm{diam}(G)$ of G.

Remark 5.1. In *directed* graphs, the relation of connectivity is not symmetric, i.e. vertex v can be connected to vertex w, but at the same time w might not be connected to v.

The notion of points that are connected to each other leads to a global property of the graph G:

Definition 5.4. Let $G = (V, E)$ be an undirected or directed graph. Then G is called a *strongly connected graph* if for any two vertices $v \in V$ and $w \in V$, v is connected to w and w is connected to v. In addition to that, a directed graph is called *weakly connected* if for any pair of vertices v and w either v is connected to w or w is connected to v.

If a graph G is not connected, it might have *subgraphs* that are connected:

Definition 5.5. For a graph $G = (V, E)$, a *subgraph* $G' = (V', E')$ consists of a subset of vertices V' together with all edges that are defined on this subset of vertices:

$$E' = \{e \in E : v, w \in V'\} \tag{5.2}$$

The combination of Definition 5.4 and Definition 5.5 gives the definition of a *component*:

Definition 5.6. If a subgraph G' of G is a connected graph, and G' is the largest subgraph with this property, then G' is called a *component* of G.

We can now define graphs $G(A)$ and $\bar{G}(A)$ that are associated with a matrix A in the following sense:

Definition 5.7. For $A \in \mathbb{R}^{n\times n}$ we can define the *associated graph* $G(A) = (V, E)$ with

$$V = \{i \in \mathbb{N} : 1 \leq i \leq n\} \text{ and } E = \{(i, j) : a_{ij} \neq 0\}. \tag{5.3}$$

In order to represent subsystems that are completely independent from the rest of the linear system, i.e. there are no couplings to or from this subsystem, we need to define the *undirected associated graph* $\bar{G}(A) = (V, \bar{E})$ by using

$$\bar{E} = \{\{i, j\} : a_{ij} \neq 0 \vee a_{ji} \neq 0\} \tag{5.4}$$

for the set of edges.

In order to write down the *Depth-First Search* algorithm, we need another definition:

Definition 5.8. The *adjacency list* $A(v)$ of vertex $v \in V$ in a graph $G(V, E)$ is a list of all vertices $w \in V$ that can be reached from v via a path of length 1.

Now that we have clarified some definitions, let us turn to a basic algorithm that is needed for the detection of components within a graph. Algorithm 1 introduces an algorithm called *Depth-First Search (DFS)*. Starting from some vertex s, the algorithm moves on to one of the neighbors w of v that it has not yet visited. It then continues to visit a neighbor of w that has not been visited and so on. Once it reaches a vertex z that has no neighbors that have not been visited yet, it continues the search at the node from which it has reached z. When the algorithm gets back to s and all neighbors of s have been visited, the algorithm terminates. Algorithm 1 is a recursive version of this method.

Algorithm 1 Depth-First Search (DFS) [148][149]

```
1: Procedure MARK = DFS(G,s)
2: MARK(s) := true;
3: for all w ∈ A(s) do
4:    if not MARK(w) then
5:       DFS(w);
6:    end if
7: end for
```

Remark 5.2. In contrast to other versions of the DFS, this version does not explicitly save the *parent* of each vertex v, which is the vertex $p(v)$ that v was first visited from.

Applying DFS to a graph G with starting point s will yield an array MARK where $\text{MARK}(v) = \textbf{true}$ if and only if v can be reached from s, i.e. if there exists a path from s to v.

Lemma 1. *As, if G is strongly connected, every edge of the graph is touched in the loop, DFS needs $\mathcal{O}(|E|)$ operations.*

In the following, we will consider *undirected* graphs, if not stated otherwise.

Lemma 2. *If G is an undirected graph, then the subgraph $G'(V', E')$ with*

$$V' = \{v \in V : MARK(v) = \textbf{\textit{true}}\} \tag{5.5}$$
$$E' = \{e = \{v, w\} \in E : v, w \in V'\} \tag{5.6}$$

is a component of G.

Proof.
1. There is a path from s to every other vertex v in G': By definition, there is a path from s to all $w \in A(s)$. For all $w \in A(s)$ there exist paths to all $w' \in A(w)$ for the same reason. Hence, for all w' we have a path $s \to w \to w'$. By applying this argument recursively, we can find paths from s to all v with $\text{MARK}(v) = \textbf{true}$.
2. Since G is undirected, there also exists a path from v to s.
3. Every two other vertices $v, w \in G'$ are connected via a path $v \to \cdots \to s \to \cdots \to w$.

□

In the case of undirected graphs G we can extend Algorithm 1 to Algorithm 2 in order to find all components of a graph G. To achieve that, the algorithm needs to re-start the DFS at every vertex that has not been reached by a previous DFS. When Algorithm 2 (DFS-C) terminates, every vertex has been visited, but the vertices have a label that indicates in which DFS run they have been visited. All vertices that have the same label belong to the same component of G.

If we integrate lines 6 – 10 from Algorithm 2 into the original DFS algorithm[2], the complexity of DFS-C is $\mathcal{O}(|V|+|E|)$. In arbitrary undirected graphs G, the worst-case scenario would be $|E| = |V|(|V|-1)/2$ which would be the case if every vertex in G was connected to every other vertex. Such graphs are called *dense* [90]. We are mainly interested in graphs associated with sparse matrices arising from discretizations using point clouds. In this case, $|E|$ depends on the size of the local neighborhoods in the point cloud, which is significantly smaller than $|V|$. In fact, the neighborhoods N_i in a point cloud have sizes $\ll 100$ in our applications, i.e. $|E| \approx c|V|$ with $c \ll 100$.

Definition 5.9. Undirected graphs with

$$|E| \ll \frac{|V|(|V|-1)}{2} \tag{5.7}$$

are called *sparse*.

[2]Which can be done by writing the component label directly into the MARK array instead of a binary true / false value.

Algorithm 2 Depth-First Search for Components (DFS-C) [148]

```
1: Procedure COMPONENTS = DFS-C(G)
2: MARK(:) := false
3: MARK_OLD(:) := MARK(:)
4: for all v ∈ V do
5:    MARK := DFS(v)
6:    for all w ∈ V do
7:       if MARK(w) ≠ MARK_OLD(w) then
8:          COMPONENTS(w) := v
9:       end if
10:   end for
11:   MARK_OLD(:) := MARK(:)
12: end for
```

Lemma 3. *Therefore, for the sparse graphs $\bar{G}(A)$ we are interested in, the complexity of DFS-C is $\mathcal{O}(|V|)$.*

Remark 5.3. Recall that for $\bar{G}(A)$ the number of vertices $|V|$ is the number of rows in the matrix A so that for these graphs DFS scales linearly in the number of points or matrix rows.

5.2.2 Related Work

DFS is a serial algorithm and using it in parallel is not straight forward, see for example [77] and more recently [36]. Also note that in the FPM every process only holds part of the matrix A and therefore of the graph $G(\mathcal{P})$, namely the part that is associated with the part of the point cloud that resides on that process.
For this reason, methods like the ones proposed in [59], [94] or [133] that assume access to a shared memory cannot be used. The method McColl et al. [94] present does have the benefit of being designed specifically for graphs that change over time. Having a distributed version of their algorithm would mean that we would not have to compute the component structure of the graph from scratch in every time step. On the other hand though, the cost for finding the components is relatively small compared to the overall linear solver, as our numerical experiments will show.
Hong et al. [61] consider directed graphs with the *small world* property, i.e. graphs with a small diameter compared to their size. We are mainly interested in undirected graphs that do not have the small world property. In our case the diameter is not small because every point is only connected to its spatial neighbors, meaning that a path from one end of the computational domain to the other is fairly long, especially if the smoothing length h is small. And because we are looking for fully independent subsystems, we are concerned about undirected connections between two points, as a connection in only one way still means that there is an exchange of information between the two parts of the graph the two points belong to.
The FW-BW method and its extension FW-BW-Trim introduced by Hendrickson et al. in [39] and [95] respectively, implement a divide and conquer strategy. Their

drawback is that after every divide step the remaining work needs to be redistributed across the participating processes, if a proper load balancing shall be achieved.

5.2.3 Origin of Components in the FPM

Now that we have clarified the notations and definitions used in the context of graph theory, let us focus on the question why the linear systems arising in the FPM may decompose into multiple smaller linear systems.

First of all, let us classify the types of components as those that can affect the existence or uniqueness of a solution for the linear system and those that cannot. Components in which both the pressure and the velocity are fixed through appropriate boundary conditions at at least one point each are well-posed components, as the corresponding linear systems have a unique solution. This is not the case, if either the velocity or the pressure is not fixed by applying the correct boundary conditions. For example, this would be the case for a component that is confined only by walls, which would mean that all of the boundary conditions for the pressure are of Neumann-type. Then, if p is a solution to the pressure in this particular component, so is $p + c$ for a constant c. Note however that this is not the case for the full linear system comprised of all components. From the linear solver perspective, it can be crucial to know about the components whose solution is only prescribed up to a constant, see Section 5.2.6.

The Geometric Case

Here we want to point out some situations that produce point clouds that geometrically induce graphs $\bar{G}(A)$ that decompose into components.

The simplest situation occurs when the simulation itself naturally introduces components because two separate flow domains are being simulated. As an example, consider simulating the flow through a valve that is opening over time. While there is only one flow domain when the valve is open, there are two flow domains when the valve is closed. As long as the simulation is set up in a physically correct way, this case leads to well-posed components.

A similar situation occurs when parts of the fluid are separated from the main part because of their velocity, i.e. when droplets of fluid are being formed. The boundary points of such a droplet are detected as free surface boundary points[3] which means we impose Dirichlet boundary conditions for the pressure and free surface boundary conditions for the velocities as described in [144] Section 5.3. Therefore, the linear system is well-posed unless both gravity and inertia tend towards 0, which would be the case for slowly moving droplets in zero gravity. This is a special case however, in which the underlying incompressible Navier–Stokes equations would admit multiple solutions, that we are not considering with our method.

[3]For the detection of free surfaces in the FPM, see [120].

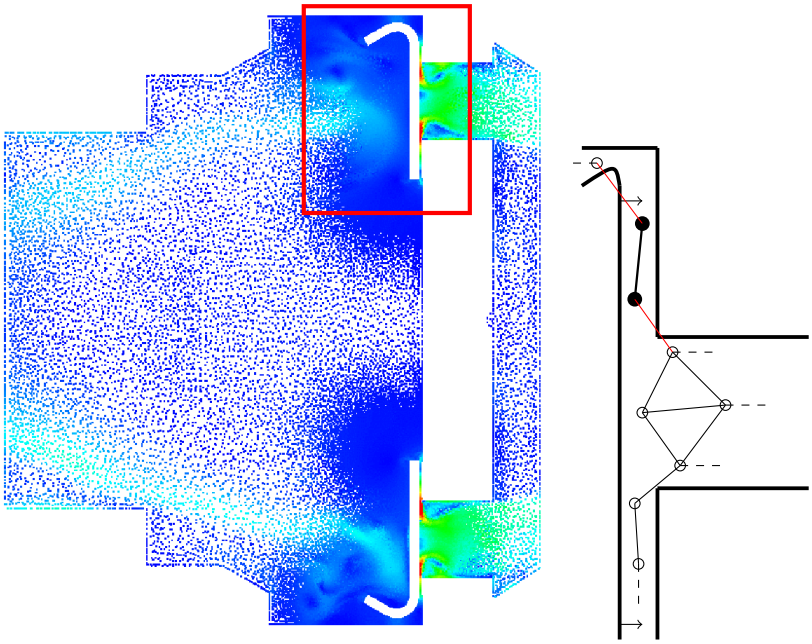


Figure 5.1: A valve that is almost closed. The left figure shows the whole valve while the right figure shows a schematic zoomed in on the highlighted area. The boundary points are not depicted in this figure as they would not change the notion of the (singular) component in this case. Note that this is only because the boundary conditions imposed at these points would be of Neumann type for the pressure system.

The most interesting case here is the case where components occur due to a fine detail of the geometry that cannot be resolved properly by the point cloud. This can happen if the geometry itself is much finer than the resolution of the point cloud from the beginning, or if the geometry is moving and during this movement fine channels in the geometry are created. An example for this would again be the opening valve. Right after the valve starts opening, there is a very small gap that can be much smaller in width than the average distance between two points of the point cloud.

Figure 5.1 shows how a component is formed while the valve is opening. The left part of the figure shows the full valve right after it has started opening. On the right-hand side we see a zoomed in version of the upper right-hand side part of the valve, shown as a schematic. The lines between two points are indicating that those points are neighbors in the point cloud. Red lines however indicate that the corresponding points are not connected numerically, e.g. there is no connection between the pressure variables in the linear system. In this case here, there is no numerical connection because of the walls intersecting the direct lines between two points. This means that the two points highlighted in bold form a component for themselves. The problem with that is that these two points are only neighbors to each other and to Neumann-type boundary points on the walls (which are not depicted here). Therefore, the linear subsystem for this component becomes singular because of the consistency conditions for the least squares problem used to construct the stencils at the two interior points.

More precisely, the linear (sub-)system arising from this component has a row-sum of 0 in every row owing to the first consistency condition for the Laplace-stencil, cf. Example 2.1. Therefore, the system's kernel includes constant functions. Analogously, there is no numerical coupling between the velocities of these two points and the velocities of the other points in the point cloud. In Section 5.4.3 we will see that this situation leads to severe problems in some linear solvers when not treated properly.

Situations like this are more of an issue of the discretization method rather than they should be an issue for the linear solver to deal with. Keep in mind though that detecting components only needs the knowledge of the connectivity graph with respect to the numerical stencils, finding a way to reconnect two components that geometrically should only be one component is a different task. One idea here would be to introduce another point to the point cloud that serves as a connection between the small and the large component. It is easy to see visually where a point like this should be located, but finding such a point numerically in a 3D problem is not an easy task. Therefore, we saw the need to deal with this situation within our AMG method by "filtering" out those components.

The Algebraic Case

Components in the linear system for solving for the velocity field can also occur even if the linear system for the pressure does not decompose into components. This can be the case when the velocities in the different directions are decoupled from each other because the viscosity η is constant in the entire domain and the boundary conditions also do not impose any couplings between the velocities. In this case though, the linear subsystems are not singular if the simulation is well-posed. Theoretically, components in the linear system may also occur across different physical unknowns. For example for two points x_1 and x_2 with physical unknowns (u_1, v_1, w_1, p_1) and (u_2, v_2, w_2, p_2) respectively, the unknowns $u_1, u_2, v_1, v_2, w_1, w_2$ and p_1, p_2 could form two components. It is possible to detect such components by using a suitable adjacency graph for the algorithm that we will introduce in the following. On the other hand, the point structure of the problem can be exploited to save some computational effort. Since these cases are somewhat rare though, we will not discuss the implementational details of this in the present thesis.

5.2.4 Detecting Components in Parallel

Let us concentrate on the geometric case of the previous section, i.e. we examine the connectedness of the undirected sparse graph $\bar{G}(\mathcal{A})$ associated to the linear system for one of the pressure fields in the FPM.
We use a variation of an algorithm described by Donev in [33]. It also fits into the framework used by Iverson et al. [66] whereas they use the term *label propagation* for what we will call *local diffusion*.

In order to detect all components in $\bar{G}(A)$, Algorithm 3 is run on all processes involved in parallel. Like the point cloud, the graph $\bar{G}(A)$ we are looking at is distributed across multiple processes. By V^{loc} we denote those vertices $v \in V$ of the graph that are

Algorithm 3 Parallel detection of components.

```
1: Procedure COMPONENTS = GET-CMP-Par(G)
2: // Find components locally on every process
3: COMPONENTS := DFS-C(G^loc = (V^loc, E^loc))
4: ROOTS:=COMPONENTS
5: // Condense remote edges to "root" of their component locally on every process
6: for all e = (v, w) ∈ E^remote do
7:    e_R := (COMPONENTS(v), COMPONENTS(w))
8: end for
9: // Define reduced graph locally on every process
10: E_R^loc := {e_R}
11: V_R^loc := {v ∈ V^loc : COMPONENTS(v) = v}
12: // Local diffusion
13: while not convergence do
14:    Synchronize COMPONENTS vector
15:    // Update components array locally on every process
16:    for all e ∈ E_R^loc do
17:       COMPONENTS(v) := min (COMPONENTS(v), COMPONENTS(w))
18:    end for
19: end while
20: // Update COMPONENTS label in full graph
21: for all v ∈ V^loc \ V_R^loc do
22:    COMPONENTS(v) := COMPONENTS(ROOTS(v))
23: end for
```

local to a process. Similarly, E^{loc} denotes all edges $e = \{v, w\}$ for which $v \in V^{\text{loc}}$ and $w \in V^{\text{loc}}$. In contrast to that, E^{remote} denotes edges $e = \{v, w\}$ that satisfy $v \notin V^{\text{loc}}$ or $w \notin V^{\text{loc}}$. Without loss of generality, in the following we will assume $v \in V^{\text{loc}}$ for $e \in E^{\text{remote}}$. Algorithm 3 yields a label for each vertex indicating to which component it belongs.

Remark 5.4. Here we examine the graph $\bar{G}(A)$ rather than the graph $G(A)$. We are interested in independent linear systems within a larger linear system and $\bar{G}(A)$ is a suitable representation for the connectivity of the larger system, see 5.2. Although $\bar{G}(A)$ is an undirected graph by the means of Definition 5.7, in this section we represent each undirected edge $e = \{v, w\}$ through two directed edges $e_1 = (v, w)$ and $e_2 = (w, v)$. This corresponds to the situation that we have when implementing these algorithms in software, where we save adjacency lists for all vertices, in which case an undirected edge is realized in the same fashion.

Remark 5.5. In this section and specifically in Algorithm 3 we use COMPONENTS as a global array, i.e. we implicitly assume that every process knows the value of COMPONENTS(i) for every i, even for those vertices that reside on other processes. In our specific implementation this is realized by using a local array for the values of COMPONENTS(i) that correspond to local vertices and another array for the values of non-local vertices that are actually needed. Thus, there is no need to have a global,

synchronized array COMPONENTS. The update of the latter array needs to be done before each iteration of the local diffusion part of our algorithm.

The first step in finding all connected components globally is to find all local components on every process first; that is ignoring all edges that are connections to vertices that reside on other processes. In the following, we will call the latter *remote edges*. Finding the local components (line 3 of Algorithm 3) is done by the DFS-C algorithm (see Algorithm 2) introduced in Section 5.2.1. Here, we could also use a different algorithm to detect the local components as long as it has the same asymptotic performance. Figure 5.2 (B) shows the original graph from Figure 5.2 (A) after finding the local components on all three processes, see Example 5.1. In line 4 of Algorithm 3 we store the information on the local components in the array ROOTS for later reference.

In the next step (lines 6–8; Figure 5.2 (C)) the algorithm examines all remote edges. Assume that $e = (v, w)$ is a remote edge where w is the remote vertex. We then want to introduce an edge $e_R = (v', w')$ that connects the two vertices that represent the local components which v and w belong to, i.e. $v' = \text{COMPONENTS}(v)$ and $w' = \text{COMPONENTS}(w)$.

Lines 10–11 (Figure 5.2 (D)) define a *reduced graph* with

$$E_R^{\text{loc}} := \{e_R\} \text{ and } V_R^{\text{loc}} := \left\{v \in V^{\text{loc}} : \text{COMPONENTS}(v) = v\right\}, \tag{5.8}$$

where the vertices are representatives of the local components and the edges indicate connections across processes between those components. Obviously, if two local components on different processes are connected in this reduced graph, they are really one component that is spread across those two processes. This means that these two local components should end up having the same global identifier, i.e. they should be detected as one large component.

On the reduced graph, we perform a *local diffusion* algorithm in lines 13–18: Every process checks for every remote edge $e_R = (v, w)$ if

$$\text{COMPONENTS}(w) < \text{COMPONENTS}(v). \tag{5.9}$$

If the inequality (5.9) holds, COMPONENTS(v) is updated as

$$\text{COMPONENTS}(v) := \text{COMPONENTS}(w). \tag{5.10}$$

Every process does this on its local copy of the COMPONENTS array (cf. Remark 5.5). Therefore, after every process has done the update for its local vertices, the COMPONENTS array needs to be synchronized. Then, the inequality (5.9) is checked again for every remote edge. These steps are repeated until the COMPONENTS array does not change any more. As a result, the minimal labels COMPONENTS(u) *diffuse* through the components of the reduced graph, see Figure 5.2 (E)–(G).

We now have the final labels for all vertices that are part of the reduced graph. The last step is to update the label for all other vertices (lines 20–22 of Algorithm 3). An easy way of doing this is to store from which root vertex a vertex has been visited during DFS-C (line 3 of Algorithm 3). We have already stored that information in the ROOTS array after running DFS-C. Therefore, this final update comes down to setting

$$\text{COMPONENTS}(v) := \text{COMPONENTS}(\text{ROOTS}(v)) \tag{5.11}$$

for all $v \in V^{\text{loc}} \setminus V_R^{\text{loc}}$. After this, the COMPONENTS array contains the correct label for every $v \in V$.

Remark 5.6. In the reduced graph, we could remove duplicate edges. Those might exist because two local components on two different processes can be connected via multiple edges from different vertices in both components. For example, in Figure 5.2 (A) both vertex 5 and vertex 6 that reside on process 2 are connected to vertex 2 on process 1. Because both vertex 6 and vertex 5 belong to the same local component, those two edges are reduced to edges e_R and e'_R connecting vertex 5 and vertex 1 in the reduced graph, see Figure 5.2 (D). One of those edges would be sufficient for the local diffusion part of the algorithm to work correctly, however the second edge does not cause a problem either. The loop in lines 14–16 performs one integer comparison for each remote edge. We would not have to do this comparison for the duplicate remote edges, if we removed them from the reduced graph. However, finding those remote edges would require at least

$$\mathcal{O}\left(|E_R^{\text{loc}}| \log |E_R^{\text{loc}}|\right) \overset{|E_R^{\text{loc}}| \leq c|V_R^{\text{loc}}|}{=} \mathcal{O}\left(|V_R^{\text{loc}}| \log |V_R^{\text{loc}}|\right) \tag{5.12}$$

operations for sorting all remote edges.
This would mean that the bound we will show in Section 5.2.5 would not hold any more. Our experiments in Section 5.4.3 also show that the potential gain here is relatively small as we do not perform many iterations of the local diffusion algorithm in practice.

Example 5.1. The graph depicted in Figure 5.2 (A) shall serve as an example for Algorithm 3. The vertex labels correspond to the value of the COMPONENTS label in the current step, whereas the color of each vertex indicates the process on which the vertex resides. Analogously, edges are colored according to the process on which they originate. In some sense, this is a worst-case example: The component that has label "1" at the very end (see Figure 5.2 (H)) stretches across all three processes and the diffusion of the minimum label "1" needs to pass all three processes before the algorithm stops. Hence in this case we reach the theoretical maximum of P local diffusion iterations that we will derive in Section 5.2.5.

5.2.5 Complexity of the Algorithm

In this section, we deal with the asymptotic complexity of our proposed algorithm in the case of sparse graphs $\bar{G}(A)$. Performance considerations regarding run-time will be the topic of Section 5.4.3.

Our argument is similar to the one in [66] but because of our more specific knowledge of the graph $\bar{G}(A)$ we can give more specific estimates.

We begin by formulating some estimates regarding the relationships between the number of vertices and the number of edges in various graphs involved here.

Lemma 4. *1.* $\bigcup V^{loc} = V$, $\sum_p |V^{loc}| = |V|$

2. $|V^{loc}| \leq |V|$, $|E^{loc}| \leq |E|$

3. $|E| \leq c|V|, |E^{loc}| \leq c|V^{loc}|$ *with* $c \ll 100$

4. $|V_R^{loc}| \leq |V^{loc}|$

Proof. 1. Every point in the point cloud and therefore every vertex in the graph we are considering here is associated to exactly one process.

2. V^{loc} and E^{loc} are subsets of V and E respectively

3. The number of edges in the graph $\bar{G}(A)$ is limited by the allowed neighborhood size in the point cloud. In the FPM, we usually allow up to 40 neighbors, see for example [141].

4. Vertices in the reduced graph represent local components and there cannot be more local components than local vertices in the original graph.

□

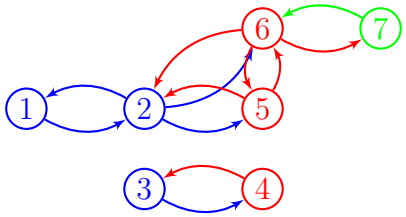
Lemma 5. *For sparse graphs and a set* $\mathbf{P}$ *of* P *processes, Algorithm 3 has an asymptotic complexity of*

$$\mathcal{O}\left(|V| \cdot P\right) \tag{5.13}$$

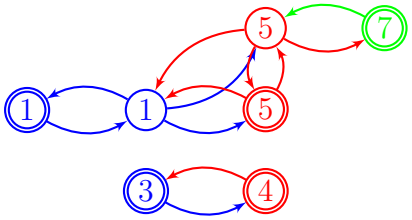
in the worst case and

$$\mathcal{O}\left(|V|\right) \tag{5.14}$$

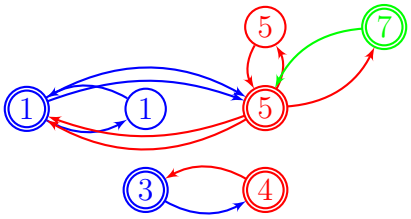
in the average case.



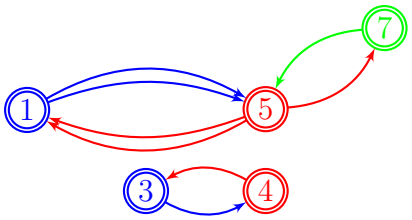
(a) The original graph.



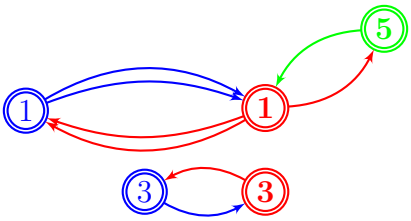
(b) After detecting local components. Component roots with two circles.



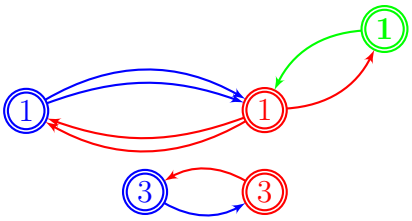
(c) With remote edges changed to start and end at root vertices.



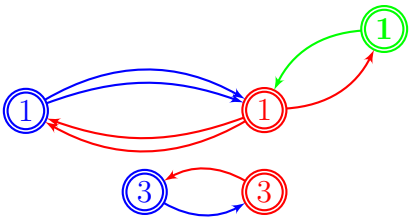
(d) The reduced graph.



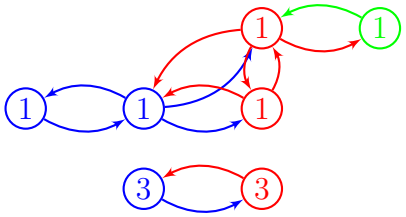
(e) The reduced graph after one iteration of local diffusion. Updated values in **bold**.



(f) The reduced graph after two iterations of local diffusion. Updated values in **bold**.



(g) The reduced graph after three iterations of local diffusion. No values have been updated, so the algorithm stops.



(h) After updating the local edges with the new COMPONENTS label.

Figure 5.2: Example for Algorithm 3.

Proof. As we have already seen in Lemma 3, DFS-C has an asymptotic complexity of $\mathcal{O}(|V^{\text{loc}}|)$ on every process. The assignment in line 4 of Algorithm 3 is also of complexity $\mathcal{O}(|V^{\text{loc}}|)$.

Finding E^{remote} which is needed in line 6 can be done in $\mathcal{O}(|E^{\text{loc}})$[4]. Because of Lemma 4, this is also $\mathcal{O}(|V^{\text{loc}}|)$. The loop in lines 6–8 needs E^{remote} iterations which, by the same argument, is also bound by $\mathcal{O}(|V^{\text{loc}}|)$.

Lines 10 and 11 are only notations that are not carried out in software, so we omit those two in our considerations.

For the local diffusion part in lines 13–18 first consider the inner loop in lines 14–16. The number of iterations in this loop is $|E_R^{\text{loc}}|$ and by Lemma 4 we have

$$|E_R^{\text{loc}}| \leq c|V_R| \leq c|V| \quad \text{where } V_R = \bigcup V_R^{\text{loc}}. \tag{5.15}$$

It remains to examine how many local diffusion iterations are needed before convergence is reached.

To this end, consider the reduced graph

$$G_R = \left(V_R, \bigcup E_R^{\text{loc}}\right), \tag{5.16}$$

in which each vertex represents a local component. The components of G_R then yield the global components. In the worst case the minimum label in a component in G_R needs to propagate along the longest shortest path (*diameter*; see Definition 5.3) in G_R. Afterwards, another iteration is needed to notice that the local diffusion has converged. This makes for a worst-case iteration count of

$$I \leq \operatorname{diam}(G_R) + 1. \tag{5.17}$$

Since the vertices in G_R represent local components on each process, there are no edges between two vertices in G_R on the same process. Therefore

$$\operatorname{diam}(G_R) \leq P - 1, \tag{5.18}$$

which yields

$$I \leq P. \tag{5.19}$$

Another assumption for the worst-case scenario would be that the vertices are spread unevenly across the processes, i.e.

$$\max_{p \in \mathbf{P}} |V^{\text{loc}}| \approx |V|. \tag{5.20}$$

In this case, all the local complexities in the previous steps of the algorithm become $\mathcal{O}(|V|)$ and local diffusion needs P iterations of an $\mathcal{O}(|V|)$ loop, meaning the local

[4] With an appropriate numbering of the local vertices versus the remote vertices or by labeling those edges beforehand.

diffusion part has the highest complexity in this algorithm: $\mathcal{O}(|V| \cdot P)$.

For the average case we assume that the vertices are spread evenly across the processes, i.e.

$$\max_{p \in \mathbf{P}} |V^{\text{loc}}| \approx |V|/P. \tag{5.21}$$

Then, the complexities in the local part of the algorithm become $\mathcal{O}(|V|/P)$ and the local diffusion part is

$$\mathcal{O}(|V|/P \cdot P) = \mathcal{O}(|V|). \tag{5.22}$$

□

Remark 5.7. Section 5.4.3 will show that this theoretical complexity is a pessimistic estimate for many cases. The main bottleneck for the complexity is the number of iterations of the local diffusion part. In the proof above, we have estimated

$$I \leq \operatorname{diam}(G_R) + 1 \leq P. \tag{5.23}$$

Note that the diameter of the reduced graph G_R mainly depends on the partition of the point cloud onto the processes which in turn depends on the shape of the computational domain. For example, consider a long and thin domain like a tube. In this case, the partition would also follow this shape leading to a graph G_R with a large diameter like $P-1$ in the worst case. If however the domain is a cube and every process has an equal cubical part of the point cloud to compute, then

$$\operatorname{diam}(G_R) \approx \sqrt{3}\sqrt[3]{P}, \tag{5.24}$$

so in this case the complexity of Algorithm 3 would be

$$\mathcal{O}\left(|V| \cdot P^{-\frac{2}{3}}\right). \tag{5.25}$$

5.2.6 Dealing with Components in the Linear Solver

There are two main types of components that our AMG method distinguishes: *Parallel components* which are solved on multiple processes. In order to determine which components will be solved in parallel, we first find the P largest components. These components are assigned to sets of processes of different sizes according to their size. Every process solves at most one parallel component. Therefore, a redistribution of the linear system across the processes may be required. Specifically, we employ Algorithm 4 for this task.

Algorithm 4 Algorithm to decide how many processes are assigned to a component.

Procedure ASSIGN
// N is the global number of matrix rows.
$N_{\text{left}} := N$
$P_{\text{assigned}} := 0$
// largest_cmp contains a list of the P largest components ordered by size
for cmp $\in$ largest_cmp **do**
  target_procs(cmp) $:= \min\left(\lfloor N_{\text{cmp}}/N_{\text{left}} \cdot (P - P_{\text{assigned}}) + 0.5 \rfloor, (P - P_{\text{assigned}})\right)$
  $N_{\text{left}} := N_{\text{left}} - N_{\text{cmp}}$
  $P_{\text{assigned}} := P_{\text{assigned}} -$ target_procs(cmp)
end for

Serial components which are solved on a single process. Every component that does not belong to the P largest component and every component that has been assigned only one process in Algorithm 4 becomes a serial component. Those serial components that are already located on a single process will stay on this process and will be solved by the same process after it has solved the parallel component it was assigned to. Serial components that reside on more than one process get redistributed to a single process in a round-robin fashion. They are solved by their assigned process after it has solved its parallel component.

For serial components whose size is below a certain threshold, the default being 100 variables, we do not employ an AMG method but use a direct solver, MKL's PARDISO, right away.

Before passing any component to a linear solver, we check if the row sum for that component is zero. If so, the component's kernel includes constant functions. Both the coarse level solver in our AMG method as well as the stand-alone direct solver that is used for small components then need to compute a solution to the linear system up to a constant. To this end, we fix the first variable of any such component to 1, defining a unique solution. Since in the FPM we only need the pressure gradient, rather than the absolute value for the pressure, the constant function does not play a role in the method anyway. Hence, the solution we find is perfectly acceptable in the context of the FPM and other pressure correction methods.

5.3 Renumbering Strategies

Now that we have separated the possible subsystems from the rest of the matrix, let us assume again that the matrix we are dealing with does not decompose into subsystems.
In Section 2.4.1 we noted that the numeration of the variables in the FPM leads to matrices with a high bandwidth, compared to classical Finite Difference Methods for Poisson-like equations. We also discussed the possible implications on the algorithmic and computational performance as well as on the MPI communication. Here, we briefly introduce two algorithms that can improve the bandwidth and the amount of MPI communication respectively. Section 5.4.4 will test the effect of those algorithms on the overall performance of our AMG method.

The first algorithm is the Reverse Cuthill-McKee algorithm (RCM) [87] that aims to minimize the bandwidth of the matrix. It is similar to a Breadth-First-Search (BFS) in a graph, but the order in which the nodes are *visited* depends on the order[5] of each node. Since the RCM algorithm requires a structurally symmetric matrix, we use the algorithm on the symmetrized adjacency matrix corresponding to $\bar{G}(A)$, similar to Section 5.2. The output of the RCM algorithm is a permutation of the matrix rows that yields a smaller bandwidth than the original matrix.

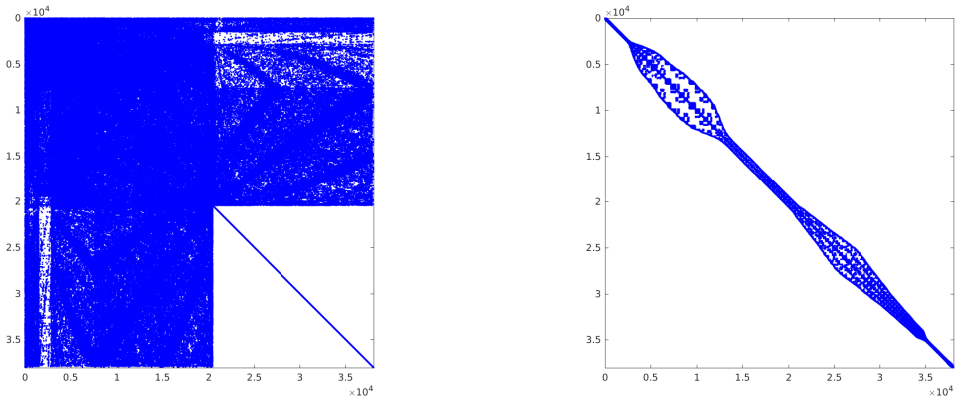


Figure 5.3: Matrix structure for the hydrostatic pressure system in the initial time step of the bifurcated tube model. The original structure is shown on the left-hand side. The right-hand side shows the matrix structure after performing an RCM renumbering.

For example, recall from Section 2.4.1 that the original matrix for the hydrostatic pressure system in the bifurcated tube model has the structure shown on the left-hand side of Figure 5.3. After renumbering the matrix according to the RCM algorithm,

[5]The order of a node in a graph is the number of edges that go in and out of that node. Note however, that the order of most nodes in the FPM case will be 40, because we restricted the number of neighbors for every point to 40 in the point cloud management. In this case, RCM is very similar to BFS, except for at the boundary.

we get the structure shown on the right-hand side. Obviously, this version has a much smaller bandwidth and should therefore lead to a computationally more efficient method.

Parallelizing the RCM algorithm is not trivial (albeit possible, see for example [70]) and thus for our purposes we use two variants that retain the serial character of RCM: The first option, that we will call *serial RCM* algorithm, is to agglomerate the symmetrized adjacency matrix onto one process and perform the serial RCM algorithm on that process only. This induces computational cost for gathering the adjacency matrix on one process and also for redistributing the system afterwards. The second option, the *localized RCM* algorithm, is to run the RCM algorithm on the local part of the matrix on each process separately, omitting the couplings to remote processes. This means that the result will not be the same as in the serial RCM algorithm, but no communication is needed at all.

The second algorithm is the PT-SCOTCH algorithm [25]. While RCM deals with the bandwidth of the matrix, PT-SCOTCH is used to reduce the amount of communication between MPI processes needed in the linear solver. This means that the matrix is re-distributed across the MPI processes in such a manner that the amount of couplings from one process to the other is reduced. In Section 5.4.4 we will discuss to what extend both methods can improve the performance of our AMG method.

5.4 Numerical Experiments

In this section, we come back to the bifurcated tube model introduced in Chapter 4. This time, we focus on the aspects of the parallelization of our AMG method.

5.4.1 Coarse Level Operators for FPM Matrices with Standard and Aggressive Coarsening

To create the second level of our multigrid hierarchy, we use a more aggressive coarsening, compared to the standard coarsening, cf. Section 4.2.2. In this section we want to show why this is a good choice in the case of GFDMs.
For comparison, we run the same simulation with full Ruge-Stüben coarsening and with aggressive coarsening on all levels. The experiment shows that using at least one level of aggressive coarsening is beneficial for the overall performance, while using aggressive coarsening on all levels does not yield much benefit or even a small increase in overall run-time like in the parallel case shown in Figure 5.5.

Again, we ran the bifurcated tube model for 1,000 time steps, using the FPM's segregated approach. Figures 5.4 and 5.5 show statistics for the serial and the parallel run with 16 processes respectively. We refer to Section 5.1 regarding the implications of the multi-processes case on AMG. Note that for the industrial sized applications the FPM focuses on, good parallel performance is a necessary condition. Therefore, although the results for the serial case are easier to interpret in many cases, the focus

	Standard	1 Level Aggressive	2 Level Aggressive
Setup time [s]	0.224	0.077	0.078
Time per cycle [s]	0.032	0.022	0.022
Overall time [s]	0.449	0.290	0.278
Cycles	7.1	9.7	8.9
Levels	4	3	2
Coarse level size	196	238	1072
g_cmplx	1.288	1.024	1.020
a_cmplx	1.600	1.011	1.009
a_avrge	49.68	39.48	39.58
w_avrge	4.25	1.63	1.59
Peak memory usage [MB]	67.184	40.314	40.314

Figure 5.4: AMG statistics (averages) for 1,000 time steps of the bifurcated tube model. Serial run. Note that the setup times are averaged across all systems, including those without a full setup.

	Standard	1 Level Aggressive	2 Level Aggressive
Setup time [s]	0.038	0.015	0.017
Time per cycle [s]	0.004	0.002	0.003
Overall time [s]	0.064	0.038	0.041
Cycles	6.7	9.1	8.6
Levels	4	3	2
Coarse level size	610	461	1368
g_cmplx	1.903	1.334	1.304
a_cmplx	1.754	1.011	1.011
a_avrge	36.84	30.44	31.01
w_avrge	3.19	1.35	1.34
Peak memory usage [MB]	103.200	78.976	77.264

Figure 5.5: AMG statistics (averages) for 1,000 time steps of the bifurcated tube model. 16-processes case. Note that the setup times are averaged across all systems, including those without a full setup.

should be on the parallel performance.

Remark 5.8. Let us briefly introduce some specific terminology for this benchmark:

- *g_cmplx* (grid-complexity) is the sum of the number of rows across all levels, divided by the number of rows on the finest level.
- *a_cmplx* (coefficient-complexity) is the sum of the number of non-zero entries across all levels, divided by the number of non-zero entries on the finest level. This is an important measure when it comes to the performance of the method, especially the solution phase, as the number of non-zero entries is the main factor in the performance of both the smoother and the accelerator.
- *a_avrge* is the average number of non-zero entries per row across all levels. This is a good indicator for how dense the coarse levels become.
- *w_avrge* is the average number of interpolation weights per row across all levels. It is a good indicator for the computational effort needed to compute the Galerkin product $A_H = RA_hP$.
- *Peak memory usage* is the maximum amount of memory allocated at any given time during AMG, including both the setup phase and the solution phase. In the parallel case, this is to be understood as the sum of memory needed across all processes.

Comparing Serial and Parallel Runs

The most important number in terms of performance is the overall run-time. Note that the time per cycle in the 16-processes case is between 7 and 11 times faster, which is a good result given that the model only has around 63k points and therefore the number of rows in the pressure system per process is only around 4k.

Remark 5.9. As a rule of thumb, a recommended minimum number of rows per process in standard AMG method is 100k. With our method for GFDM matrices we have been able to obtain decent speed-ups with at little as 25k rows per process [99].

On the other hand, the speed-up in the setup is only between 4x and 6x, which is a common observation in standard AMG methods. Since the convergence rate is almost constant between the serial and the parallel case[6] the overall speed-up of the 16-processes case is between 6.8x and 7.6x. This speed-up comes at the cost of an increased memory consumption of around 2x, which is not an issue in this case, but one needs to keep that in mind when thinking about larger models that consume more memory themselves.

The sum of rows across all levels, the grid-complexity, increases in the parallel case, because our AMG method treats couplings across process boundaries as weak couplings, despite their actual size. Therefore, in the parallel case the overall number

[6] See Section 5.1 on why this does not have to be the case.

of strong couplings is smaller than in the serial case, which means that for every C-point there are fewer F-points, increasing the number of C-points needed. For the same reason, the average number of interpolation weights, w_average, decreases. This in turn means that the average number of non-zero entries per row across all levels, a_avrge, also decreases. On the other hand, the sum of non-zero entries across all levels, the coefficient-complexity, increases, because of the increase in the grid-complexity.

Because the coarsening only takes place locally on every process on the first three levels, the size of the coarsest level in the parallel case is generally higher compared to the serial case. This can become very critical in applications with a high number of processes of 1000 and more, see [112]. We see this effect starting to happen even in our 16-processes experiment here: For all three coarsening strategies, the sizes of the coarsest level are much higher than in the serial case, although they are still manageable and we could have decided to create more levels in the parallel case here[7]. However, despite being manageable and not taking too much time to solve directly, they do contribute to the increased memory consumption of the parallel run.

Why Does the Aggressive Approach Pay Off?

Since we have now discussed the differences between the serial and the parallel case, let us turn to the question which of the three approaches should be used in production, i.e. which approach yields the shortest overall run-time and why that is the case.

In the parallel case, which is the more interesting one, the aggressive strategy on the first level is the one with the shortest overall run-time, beating the standard coarsening by a factor of about 1.7x. This number varies from model to model and also depends on the number of processes, but this strategy has shown to be the most successful one in a number of cases.

The most benefit of the 1-level aggressive coarsening strategy is in the setup, where it is 2.5x faster than the standard setup. The main reason for this is that because of the more aggressive coarsening strategy, the coarse levels become much smaller (grid-complexity) and also the interpolation and restriction operators are sparser (w_avrge), leading to less computational effort in the Galerkin product. Additionally, less levels are needed until a matrix size is reached that is suitable to be solved directly.

On the other hand, the selection of fewer C-points negatively impacts the convergence rate. Therefore, the number of cycles needed to converge increases compared to the standard coarsening. This increase is about 1.4x in both the serial and the parallel case. Owing to the lower complexities of the aggressively built hierarchy though, the time per cycle decreases by 1.5x in the serial case and even 2x in the parallel case, which more than makes up for the additional iterations needed.

[7] This is not always feasible with very high core counts as it can induce a lot of communication. A good strategy in this case is to agglomerate the setup phase down to a lower core count.

Aggressive coarsening strategies in general are a trade-off between a faster, but less accurate setup phase with a lower time per cycle, and a more computation intensive setup that leads to more expensive cycles, that converge faster in turn. For problems that only need a few iterations with the standard coarsening techniques, faster setup and time per cycle typically outweigh the penalty in the convergence rate. Another factor that needs to be accounted for in these considerations is the number of possible setup re-uses: When a setup can be re-used for a large number of time steps, the number of iterations per setup is higher, therefore investing more time in a standard setup can pay off more easily as compared to a situation where a new setup is done for every linear system. To this end, the FPM is more on the side of having a fairly limited setup re-use capability. Other, mesh-based and quasi-stationary applications in the field of CFD can sometimes re-use the same setup for thousands of time steps, making a standard coarsening strategy much more valuable. In some other domains, like elasticity, the convergence rates are not as good as we observe here even when using specialized coarsening and interpolation strategies, some of which are described in [46]. In these cases, aggressive coarsening strategies cannot be used, as the negative impact on the convergence rate would be much more severe and may even lead to divergence of the method.

The idea of applying an aggressive coarsening strategy on the first level can easily be extended to applying this strategy on more or even all levels. The effect of applying aggressive coarsening strategies to more than just one level is comparable to applying it on the first level, however the effect is amplified. In the bifurcated tube model shown here, most numbers are very close to the ones with just one level of aggressive coarsening. This is mostly because with the aggressive coarsening on level 1, the second level is already comparatively small (by a factor of ≈100x), which makes the compute time needed on level 2 and below almost vanish. Still, there are some differences with the 2-level aggressive coarsening which we benchmarked here:
First of all, the number of cycles needed to converge slightly decreases again, which is not a general observation but is related to the fact that in the case shown here, there are only two levels in the AMG hierarchy. This often gives a slightly better convergence compared to a deeper hierarchy.
Secondly, the overall run-time is about the same. The reason for that is that with the 2-level aggressive coarsening, the coarsening stops on level 2, because the number of rows on level 3 would be very low and the number of rows on level 2 is already within the range of a direct solver. However, level 2 is three times as large as level 3 in the 1-level aggressive coarsening case. Thus, both the setup time, where the decomposition for the direct coarse level solver takes place, and the cycling time, where the coarse level solver is applied, increase slightly, as opposed to decreasing when comparing the 1-level aggressive coarsening to the standard coarsening. Therefore, the 2-level aggressive coarsening yields slightly worse results compared to the 1-level aggressive coarsening in the parallel case and only slightly better results in the serial case. Both results are not general observations, but problem dependent. In any case though, the effect is not as big as when changing from standard coarsening to 1-level aggressive coarsening. Since the 2-level aggressive coarsening has a higher chance of severely impacting the convergence rate on the one hand, and does not give a substantial ben-

	Standard	1 Level Aggressive	2 Level Aggressive
Setup time [s]	0.164	0.098	0.051
Time per cycle [s]	0.004	0.003	0.001
Overall time [s]	0.184	0.121	0.062
Cycles	5	9	10
Levels	4	4	3
Coarse level size	1365	953	503
g_cmplx	2.971	2.142	1.999
a_cmplx	3.096	1.608	1.196
a_avrge	7.29	5.26	4.187
w_avrge	1.64	0.95	0.93
Peak memory usage [MB]	20.817	13.707	3.515

Figure 5.6: AMG statistics for a 3D Poisson equation on a $40 \times 40 \times 40$-cube using a regular 9-point FD stencil. 16-processes case with 4k rows each.

efit on the other, our choice in the context of the FPM is to use 1-level aggressive coarsening only.

Differences to the FD Case

How do these observations differ from a standard 9-point Finite Difference Method stencil for a 3D Poisson problem discretized with FD?
In order to answer this question, we look at the same statistics as above, but this time comparing a standard Poisson equation on a 3D cube of size 1 with Dirichlet boundary conditions, discretized by a 9-point Finite Difference Method stencil, with the same cube modeled by the FPM. As a right-hand side, we use the vector $A \cdot \phi$, where A is the system matrix and ϕ is a random vector with $\|\phi\|_2 = 1$. In both cases, we reduced the residual by 8 orders of magnitude. For the Finite Difference discretization, we chose a $40 \times 40 \times 40$-grid, leading to exactly 64k matrix rows. For the GFDM discretization in the FPM, we cannot prescribe an exact number of points / matrix rows. With the smoothing length of $h = 0.07$ we chose for this benchmark, we ended up with ≈67k points.

Figures 5.6 and 5.7 show that the effect of the aggressive coarsening strategy is very similar in both discretizations. We also observe that, despite the FPM matrix lacking some of the *nice* properties the FD matrix has, the numbers of cycles are very comparable. The key difference however is in the complexities: The coarsening is a lot faster in the GFDM case than in the FD case. For the 9-point stencil, the standard coarsening yields a coarsening rate of exactly 0.5 on the first level, whereas for the GFDM discretization it yields 0.06, which means in the latter case the second level already has one order of magnitude rows less than in the FD case. Note however, that the cost per cycle is the same in both cases, because the FD matrices are a lot sparser, which also means that the solver needs far less memory.

	Standard	1 Level Aggressive	2 Level Aggressive
Setup time [s]	0.094	0.039	-
Time per cycle [s]	0.004	0.003	-
Overall time [s]	0.109	0.062	-
Cycles	4	8	-
Levels	3	2	-
Coarse level size	784	736	-
g_cmplx	1.532	1.382	-
a_cmplx	1.097	1.005	-
a_avrge	28.653	29.088	-
w_avrge	1.81	0.81	-
Peak memory usage [MB]	279.408	172.00	-

Figure 5.7: AMG statistics for the hydrostatic pressure system in a 3D cube using meshfree GFDM operators from the FPM. 16-processes case with ≈4.3k rows each. Since the 1-level aggressive coarsening already yielded only two levels, there are no numbers for the 2-level aggressive coarsening.

Brief Experiments with Aggregation-Based AMG

Apart from the standard and the aggressive coarsening techniques that were discussed earlier in this section, another class of coarsening strategies are the aggregation-based techniques [163] [164] [162]. Instead of choosing a subset of the fine level variables to become coarse level variables, these methods form *aggregates* that are composed of multiple variables. In many applications, these techniques lead to faster setups at the cost of a higher convergence rate.

Here, we compare three different aggregation-based techniques: First, a greedy algorithm that starts at a random point and adds the point with the most strong couplings to the current aggregate into the current aggregate. When the maximum number of points in an aggregate is reached (by default this number is four, see below), it starts a new aggregate at one of the neighbors of the current one. Secondly, we use the aggregation technique based on local quality measures introduced in [102]. Lastly, we use a diameter based technique as introduced in [47], which tries to minimize the diameter of each aggregate, whereas the distance of two variables of the aggregate is defined as the absolute inverse of their coupling coefficient in the matrix. For all three techniques, we conducted experiments using them only to create the second level and using them to create all levels, as one would do in purely aggregation-based AMG methods. In all cases, we used the *-cycle which scales the coarse level Galerkin operator in order to improve the coarse level correction [20].

Figure 5.8 shows the setup time, the iteration count and the overall solution time averaged across the three pressure systems (with setup re-use) in the bifurcated tube model using one process. Like in the previous section, the model has around 63k points. For this experiment, we use a maximum aggregate size of four, which is com-

Aggregation Technique	Levels	Setup Time [s]	Overall Time [s]	Iterations
Greedy	Second Level	0.203	1.617	56.0
	All Levels	0.195	3.747	132.3
Quality Measure	Second Level	1.000	7.477	5.0
	All Levels	1.021	6.565	5.0
Diamater	Second Level	0.182	0.539	17.0
	All Levels	0.169	1.818	70.0

Figure 5.8: Setup time, iteration count and overall solution time averaged across the three pressure systems (with setup re-use) in the bifurcated tube model using one process.

Aggregation Technique	Levels	Setup Time [s]	Overall Time [s]	Iterations
Greedy	Second Level	0.164	2.078	114.3
	All Levels	0.156	1.875	94.7
Quality Measure	Second Level	3.229	8.206	5.0
	All Levels	3.128	8.417	5.0
Diamater	Second Level	0.591	1.099	28.7
	All Levels	0.588	2.453	101.0

Figure 5.9: Setup time, iteration count and overall solution time averaged across the three pressure systems (with setup re-use) in the bifurcated tube model using one process. This time using a maximum aggregate size of 70.

mon in order to prevent the convergence rate from degrading too much. With this setting though, the setup phase takes longer than in the aggressive setting from the previous section[8]. That is simply because a maximum aggregate size of four does not allow the coarsening to be as fast as in the aggressive case. Therefore, the complexities of the hierarchies built using aggregation-based techniques are higher, which, together with the degraded convergence rate, leads to longer run-times. Note that the quality-measure-based technique is not able to coarsen efficiently and therefore ends up with a hierarchy with only two levels, leading to a very high setup time.

In order to allow for a faster coarsening, we can set the maximum aggregate size to 70 which would allow for a coarsening rate that is comparable to the coarsening rate of the aggressive coarsening. Note however that the three aggregation techniques we show here do not guarantee that any or all of the aggregates being formed have the maximum size. Therefore, real coarsening rate is still slower than in the aggressive case, but it is comparable. Figure 5.9 shows the results for these tests. Constructing aggregates of that size unfortunately makes the setup even more expensive in the lat-

[8]When comparing the numbers from this section to the previous one, keep in mind that in the previous section we gave average numbers across multiple time steps. The variance across the time steps is fairly limited though, and since the numbers in this section are significantly worse than in the section before, this variance can be omitted.

h	0.03			0.02			0.015		
Processes	16	64	256	16	64	256	16	64	256
CL Size	479	1025	2751	965	1155	2020	1719	4591	3191
Iterations	31	28	27	30	30	27	31	29	28
LU-decomp.	0.061	0.077	0.342	0.092	0.080	0.264	0.112	0.357	0.378
PARDISO	0.031	0.088	0.188	0.079	0.051	0.141	0.080	0.293	0.220

Figure 5.10: Coarse level size, iterations (for all three pressure systems) and overall time spent in the two different coarse level solvers for different smoothing lengths h and different numbers of processes. Bifurcated tube model with a Reynolds number of 1000 using the segregated approach.

h	0.03			0.02			0.015		
Processes	16	64	256	16	64	256	16	64	256
CL Size	1151	1767	2433	409	2563	3750	661	881	6399
Iterations	32	31	29	44	38	34	61	61	36
LU-decomp.	0.096	0.124	0.290	0.044	0.190	0.442	0.061	0.122	0.720
PARDISO	0.111	0.156	0.319	0.031	0.317	0.563	0.062	0.062	1.080

Figure 5.11: Coarse level size, iterations (for the coupled system) and overall time spent in the two different coarse level solvers for different smoothing lengths h and different numbers of processes. Bifurcated tube model with a Reynolds number of 0.001 using the coupled approach.

ter two techniques. In the quality-measure-based technique, the measure needs to be computed much more often and in the diameter based technique, the diameter of much larger aggregates needs to be computed. For the first technique we observe a slightly faster setup due to the faster coarsening rate, but the convergence is still much worse than with the aggressive coarsening technique from the previous section. Therefore, the overall run time is also not very competitive.

The situation is very similar in the parallel case: With 16 processes, the first technique on the second level with a maximum aggregate size of 70 is the best out of the techniques tested here. However with an average setup time of 0.034s and an average overall time of 0.101s it is still not faster than the aggressive techniques. The other aggregation-based techniques perform even worse than that. Because of these results, we will not consider these techniques for the rest of this thesis.

5.4.2 Coarse Level Solver: LU-Decomposition vs. PARDISO

The choice of the coarse level solver being used mainly depends on the size of the coarsest level and the number of iterations that we expect. Recall that for the PARDISO direct solver, we need to agglomerate the coarsest level to one process, whereas the LU-decomposition is fully parallel. However, the initial *setup* costs for

the LU-decomposition are higher compared to PARDISO. This can pay off though if sufficiently many iterations are performed. Also, because of its parallelism, the LU-decomposition is better suited for larger coarse levels than the PARDISO solver. Because of the parallel coarsening techniques in SAMG, larger coarse levels are more likely to occur with higher numbers of processes.

Figure 5.10 shows benchmarks with the bifurcated tube model in the segregated approach. Here, in almost every case PARDISO is the better choice, which is not unexpected because of the low number of iterations. Note that the iterations numbers given here are across all three pressure systems. In the coupled system, the benchmarks shown in Figure 5.11 indicate that the LU-decomposition is the better choice. The coarse levels are generally larger and more iterations are performed. Note that the two instances where the PARDISO solver is the better option in this figure, the coarse levels happen to be comparatively small for their respective process counts.

In most cases that we examine here, the choice of the coarse level solver is not crucial for the overall effectiveness of the method. Some improvement can be obtained by choosing the right solver though. For our new AMG method, we use the PARDISO solver for the pressure systems and the LU-decomposition for the coupled velocity-pressure systems. Keep in mind though that this decision might need to be revisited when moving to much higher process counts beyond $1,000$ processes.

5.4.3 Components

In this section, we want to examine the practical performance of the algorithm to detect the independent components of our linear systems in parallel. Towards this end, let us turn to the case of an opening valve again, see Section 5.2.3. In this real world example, we find that at some point during the simulation, right after the valve starts opening, a small component with 5 points is formed. Note that this is very small compared to the overall size of the point cloud, which is 380641. Because of the consistency conditions for the stencil [35] [68], the row sums of the rows in the pressure system for those 5 points are all zero. In this particular model, gravity is disabled which means that the external body forces acting on these points is 0. At the same time, the initial guess is zero, which means that the subsystem has the form

$$\begin{pmatrix} 1 & -\frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} \\ -\frac{1}{4} & 1 & -\frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} \\ -\frac{1}{4} & -\frac{1}{4} & 1 & -\frac{1}{4} & -\frac{1}{4} \\ -\frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & 1 & -\frac{1}{4} \\ -\frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \tag{5.26}$$

Hence, the linear system is singular, but the initial guess is already a solution. Consequently, both BiCGStab2 and Gauss-Seidel solve the full linear system without being affected by the singular subsystem, as both methods do not change the values of the initial guess at the corresponding rows. In the one process case, their convergence rates are 0.947 and 0.999 respectively, so they need a lot of iterations to solve the system up to the desired accuracy. But this observation is the same without a singular

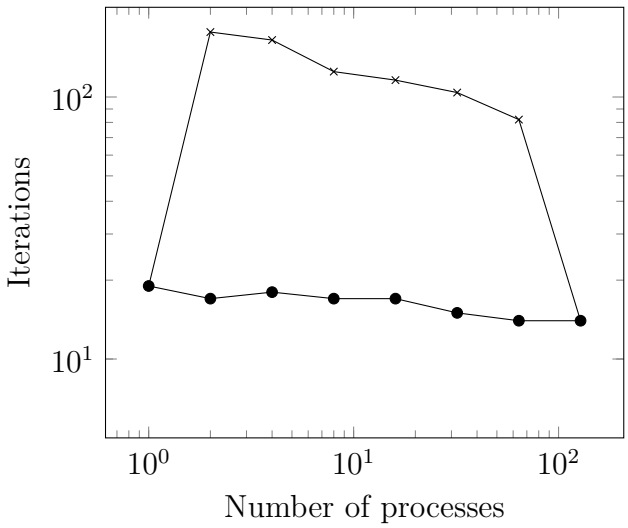


Figure 5.12: Number of AMG iterations when working on the full matrix (crosses) and with the solver being aware of components (dots).

subsystem, see [103].

On the other hand, our AMG method is affected severely by the singular subsystem: Figure 5.12 shows that except the for 1 and 128 processes case, the number of AMG iterations is much higher when solving the full linear system, compared to solving the system without the singular subsystem. In order to understand the exceptions of the 1 and 128 processes case, we first need to look at why the other cases need that many iterations.

We find that the coarse level solver is not finding an appropriate solution to the coarse level problem in these cases, because the coarse level problem is singular. This is simply because the singularity on the finest level was transferred down all the way to the coarse level. In the serial case, this does not happen because when constructing the second level, one of the five rows corresponding to the singular subsystem is picked to be on the second level, while the others automatically become fine level rows, as all the couplings are equal $(-1/4)$ in the stencil at the singular component. Therefore, all the couplings in every row are considered to be strong couplings, see [138], so as soon as one of the five rows is chosen to be on the coarse level, all others have to stay on the fine level. Then on the second level, the former singular component is represented by only one single, independent variable[9]. In this case, our AMG algorithm flags this variable to stay on the second level – as the smoother will solve for this variable directly – before starting to construct the third level. Consequently, the singular component is not represented any more from the third level downwards.

The reason why this is not happening in most of the parallel cases is that, as explained in Section 5.1, in our AMG method we consider couplings that couple rows which reside on different processes to be weak couplings, no matter how large they are

[9]Note that if following the algorithm described in [138] closely, the diagonal entry in the corresponding row would be 0, leading to a coarse level equation $0 = 0$. Our AMG method implements a check to avoid non-positive diagonals on coarse levels, which causes this entry to be 1 instead.

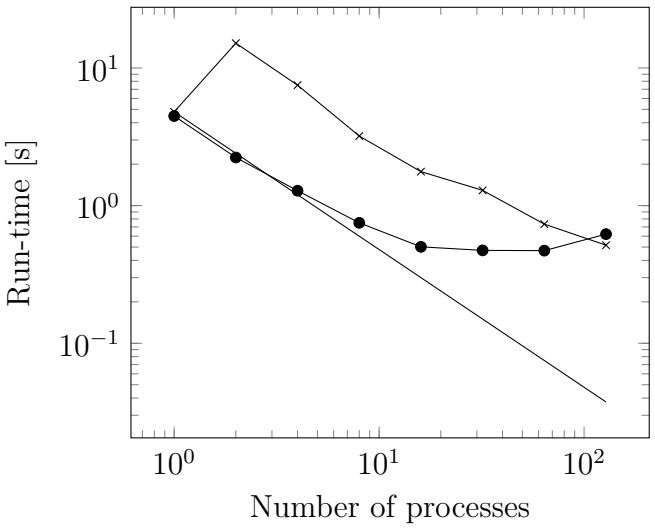


Figure 5.13: Run-time when working on the full matrix (crosses) and with the solver being aware of components (dots). Optimal scaling indicated by the black line.

by absolute value. This means that the singular subsystem is not reduced to a single point and then not taken to the next coarse level as in the serial case. Instead, if the singular subsystem resides on more than one process, it is reduced to a single point on every process, but it remains on all coarser levels because of its couplings to the other processes. Note that, to this end, the 128 processes case is a special case where the five points comprising the singular component happen to be all on one process again.

With the component detection turned on, we only solve the main, regular linear system with our AMG method and the small, singular linear system is passed to MKL's direct solver PARDISO, where we fix one variable to 1, thereby defining a unique solution. Note that since in the FPM we are only interested in the pressure gradient ∇p anyway, so knowing the solution to the small subsystem up to a constant is sufficient for our purposes.

Comparing the run-times of both the AMG method with and without the detection of independent components, we see that the benefit from finding the singular component in terms of run-time is smaller than the difference in iteration numbers indicates, see Figure 5.13. That is because the setup cost when solving the reduced system without the singular component is between 31% and 64% of the overall solver run-time and this portion of the run-time is the same whether the full system is solved or just the non-singular part. Note that the parallel scalability of our AMG method flattens out around 16 processes. This is because with more than 16 processes, the number of matrix rows per process drops below 10,000. For these relatively small numbers of rows per process we cannot expect parallel scalability any more, see Section 5.1. The discussion here is geared towards the comparison of the method with and without finding the independent subsystems. Regarding the parallel scalability of the method, we refer to Section 5.4.5.

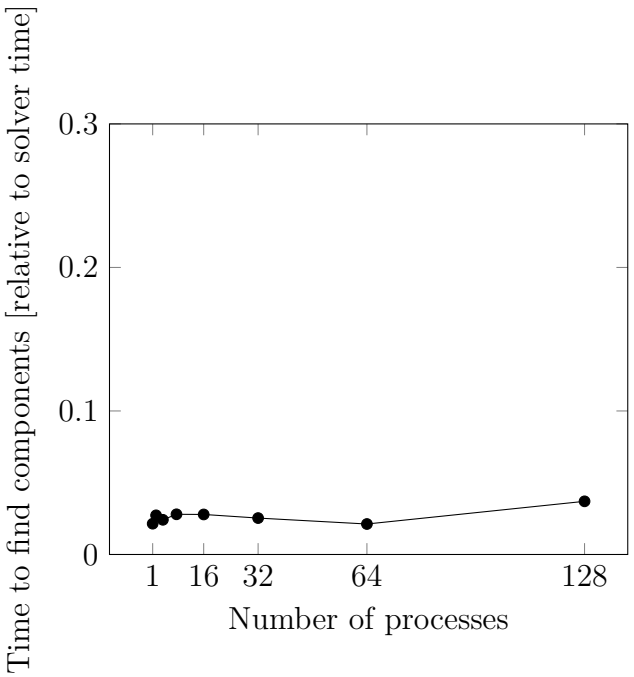


Figure 5.14: Time needed to determine the component structure of the graph relative to the overall time required for solving the system.

Removing the singular component from the system only speeds up the iteration part of the solving process by reducing the number of iterations needed.

On the other hand though, finding the components and redistributing the system according to Section 5.2.6 requires additional time that is not needed when solving the full system. Figure 5.14 shows that the former task is achieved in less than 4% of the overall solver run-time. This is better than the theoretical estimate in Lemma 5 suggests. The reason for that is that in Lemma 5 we assumed that the number of iterations of the local diffusion algorithm can be up to $P + 1$, where P is the number of processes involved.

However, Figure 5.15 shows that even though the number of iterations is increasing with the number of processes used, for 1,024 processes still only 9 iterations are needed. This observation means that the local diffusion part of Algorithm 3 is actually a lot less expensive than predicted and that Remark 5.7 is important here. It is a *best case* addition to Lemma 5 and although the actual number of iterations depends on the geometry of the given problem, many results are closer to the best case than to the average case.

After the components have been found, we need to redistribute the system according to Section 5.2.6. This task is much more expensive in terms of run-time, as Figure 5.16 shows. In other words, while finding the components only takes less than 4% of the overall run-time, redistributing the matrix takes between 20% and 30%.

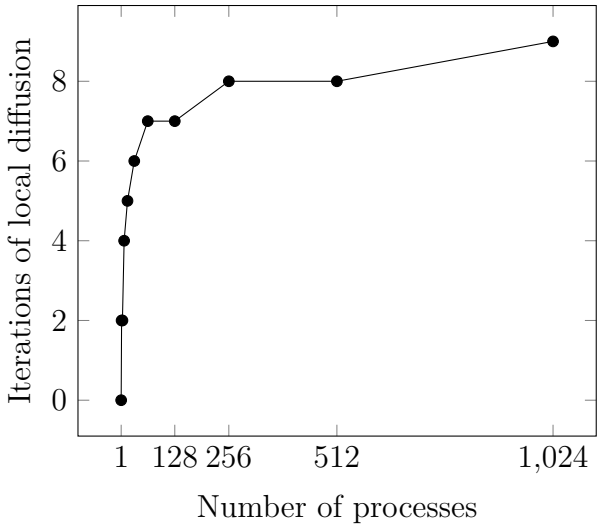


Figure 5.15: Number of iterations of local diffusion for different numbers of processes.

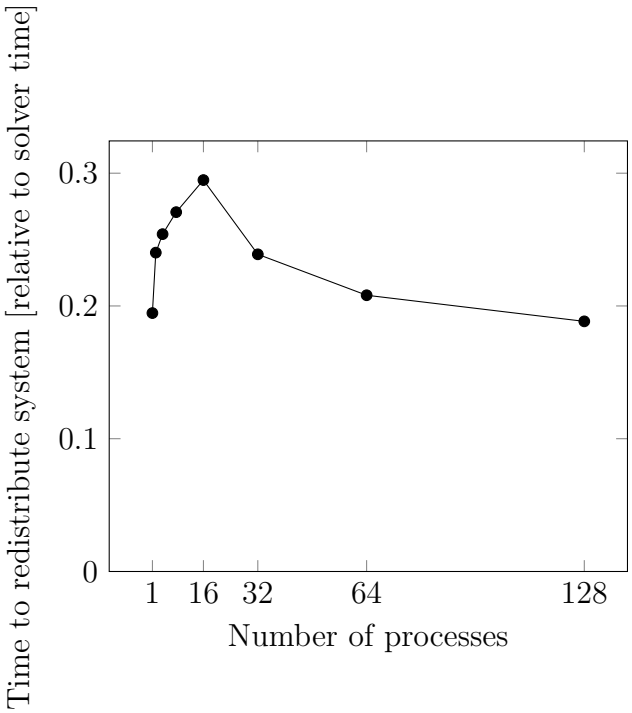


Figure 5.16: Time needed to redistribute the matrix across the processes relative to the overall time needed to solve the system.

Avoiding Redistribution

These results indicate that *finding* the components comes at a negligible cost compared to *redistributing* the linear system. Thus, future work should be dedicated to finding means to avoid or at least reduce the amount of redistribution that needs to be done. For the case of one large component and one or several very small ones, like in the example of the opening valve, it would be beneficial to only redistribute those small components and solve them on one dedicated process using a direct solver. In the valve case shown here, this method would have worked and the redistribution could have been avoided all together as only one component would have been left, which our AMG method then would have solved with the given partition (rather than a new, redistributed one) across the processes. Another option would be to add some small value to the diagonal of the singular components. This introduces a small error, but that error would only affect very few points. A third option is to treat the small components as a Schwarz block without any couplings to the remainder block with the large component. Again, this would remove the small components from the AMG method and solve them separately without a need to redistribute the full linear system.

5.4.4 Renumbering of Matrix Rows

This section examines the impact of both the RCM algorithm and the *PT-SCOTCH* method on the performance of our AMG method.
The 100th time step of the bifurcated tube model again serves as an example here. We begin by looking at the pressure systems and will give all numbers across all three of these. By doing so we make sure that the influence of the setup phase, which is only present in the first linear system, is included correctly.
Figure 5.17 shows that the best method in terms of run-time is the localized RCM algorithm. Note however that it is only better than the version without any renumbering in the 2 processes case for the two largest models. In the 16 processes case, the two are en par. The reason for that is that in the 2 processes case, the improvement in the run-time per cycle is substantial, whereas in the 16 processes case it is not, see Figure 5.18. This is because the smaller the number of rows per process becomes, the smaller the potential performance improvement through a banded matrix turns out to be. On the other hand, for a small number of rows per process, the amount of communication between processes becomes bigger and outweighs the local computation. In the end, the slight improvement in the 16 processes case cannot make up for the cost of applying the localized RCM in the first place. Figure 5.19 shows that cost for the different methods. It also makes clear that the bad performance of *PT-SCOTCH* is mainly caused by the high cost of applying the algorithm and redistributing the system, which in turn is at least partially caused by the fairly high number of non-zero coefficients per matrix row we see in GFDMs. At the same time, it shows why the localized RCM variant is better than the serial one.

Remark 5.10. Applying both *PT-SCOTCH* and localized RCM together would yield an even better performance per cycle, but we did not conduct that experiment due to the high cost of the *PT-SCOTCH* setup and redistribution.

h	.02	.03	.04	.05	.06	.07	.08	.09	.1
Matrix Rows	1.1m	366k	166k	93k	63k	41k	27k	22k	17k
No renumber	16.996	4.698	1.693	.859	.553	.373	.258	.232	.182
Serial RCM	12.244	3.845	1.765	.924	.648	.410	.303	.267	.211
Localized RCM	10.885	4.226	1.630	.860	.606	.393	.286	.256	.199
PT-SCOTCH	21.605	5.700	2.356	1.274	.799	.551	.411	.383	.319
No renumber	1.938	.683	.370	.246	.191	.153	.124	.113	.105
Serial RCM	2.753	.984	.520	.308	.247	.190	.154	.133	.128
Localized RCM	1.964	.717	.405	.253	.198	.169	.139	.118	.107
PT-SCOTCH	4.421	1.998	1.348	1.107	1.040	.820	.671	.605	.523

Figure 5.17: Overall AMG run-time in seconds (across all three pressure systems) for one time step in the bifurcated tube model on 2 and 16 processes.

h	.02	.03	.04	.05	.06	.07	.08	.09	.1
Matrix Rows	1.1m	366k	166k	93k	63k	41k	27k	22k	17k
No renumber	.441	.111	.040	.018	.011	.007	.005	.004	.004
Serial RCM	.214	.07	.031	.016	.011	.007	.005	.004	.004
Localized RCM	.215	.072	.031	.017	.011	.007	.005	.004	.004
PT-SCOTCH	.427	.106	.041	.018	.011	.007	.005	.004	.003
No renumber	.04	.013	.007	.004	.003	.003	.002	.002	.001
Serial RCM	.037	.014	.007	.004	.003	.002	.002	.001	.001
Localized RCM	.036	.013	.007	.004	.003	.002	.002	.002	.001
PT-SCOTCH	.037	.013	.006	.004	.003	.002	.002	.002	.001

Figure 5.18: Time per AMG cycle in seconds (across all three pressure systems) for one time step in the bifurcated tube model on 2 and 16 processes.

Lastly, Figure 5.20 shows that the number of cycles needed in this time step slightly decreases in most cases when using the two RCM variants. This is also a reason why especially the localized RCM variant performs fairly well overall. Similar observations with other linear solvers have been made for example by ur Rehman et al. in [159]. The figure also shows that the number of cycles needed by our AMG method is almost constant across both the number of matrix rows, which depends on the smoothing length, and the number of processes used.

Figures 5.17 and 5.18 suggest that the effectiveness of the localized RCM algorithm for our AMG method mainly depends on the number of rows per process. In order to analyze this further, Figure 5.21 plots the overall run-time across the three pressure systems over the number of matrix rows per process for 2, 8 and 16 processes. The run-times when using the original matrix are shown using dashed lines, whereas continuous lines show the run-times when using the localized RCM algorithm. It can be seen that in all cases the localized RCM algorithm pays off from around $50,000$ rows

h	.02	.03	.04	.05	.06	.07	.08	.09	.1
Matrix Rows	1.1m	366k	166k	93k	63k	41k	27k	22k	17k
Serial RCM	2.405	.738	.331	.154	.105	.071	.047	.046	.032
Localized RCM	1.553	.633	.216	.121	.085	.051	.036	.032	.025
PT-SCOTCH	2.882	1.041	.502	.316	.217	.179	.143	.153	.129
Serial RCM	.876	.286	.134	.072	.052	.037	.028	.025	.023
Localized RCM	.233	.079	.042	.022	.019	.014	.013	.008	.007
PT-SCOTCH	2.504	1.297	.987	.863	.858	.675	.555	.489	.423

Figure 5.19: Renumbering time (computing the renumbering and redistribute and/or renumber the system) in seconds (across all three pressure systems) for one time step in the bifurcated tube model on 2 and 16 processes.

h	.02	.03	.04	.05	.06	.07	.08	.09	.1
Matrix Rows	1.1m	366k	166k	93k	63k	41k	27k	22k	17k
No renumber	30	32	29	31	31	33	28	31	28
Serial RCM	30	28	29	28	29	26	27	28	27
Localized RCM	27	27	28	25	28	28	27	30	26
PT-SCOTCH	34	32	31	34	33	31	30	31	31
No renumber	31	30	29	30	30	27	28	29	30
Serial RCM	28	27	28	25	27	25	27	27	27
Localized RCM	27	25	26	25	26	26	28	25	28
PT-SCOTCH	30	28	30	31	27	28	27	29	29

Figure 5.20: Number of AMG cycles (across all three pressure systems) for one time step in the bifurcated tube model on 2 and 16 processes.

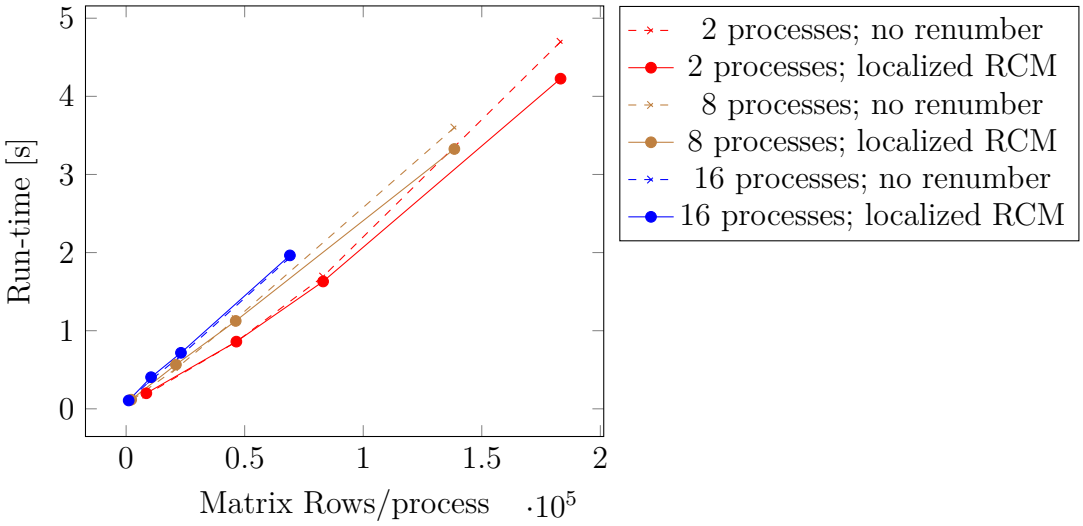


Figure 5.21: Run-time of our AMG method across all three pressure system with different numbers of matrix rows per process, different process counts and with or without localized RCM renumbering.

per process upwards. Considering that in order to achieve good scaling properties in terms of parallelization our AMG method requires $> 25,000$ rows per process (cf. the parallelization experiment in [99] and the following section), 50,000 rows would be desirable anyway. However we need to take into account here that other parts of the overall simulation, for example the point cloud management in the case of the FPM, might have different requirements in terms of a desirable number of points per process. In fact, many of the industrial size applications using the FPM have less than those $50,000$ points per process because otherwise other parts like the point cloud management become too expensive. This means that using the localized RCM algorithm does not pay off in the pressure systems in those cases.

The situation can change in the coupled velocity-pressure system, though. First, there are four rows per point in this linear system and secondly our AMG method usually needs more iterations to solve it. This means that the benefit of having a faster iteration becomes more significant compared to the cost for the initial redistribution. Also note that we use the specialized approaches described in Section 4.3 to solve the coupled velocity-pressure systems. For the Saddle Point AMG method, we use the Alternate-Block-Factorization, which highly profits from the renumbering. In Figure 5.22 we show the same data as in Figure 5.21, but this time for the coupled velocity-pressure system in the bifurcated tube model, using the Saddle Point AMG method with Uzawa-smoothing but without stabilization. What we see is that in the case of the coupled systems, the localized RCM method has a much greater impact in almost all of the test cases. In fact, it still achieved a small benefit when using 256

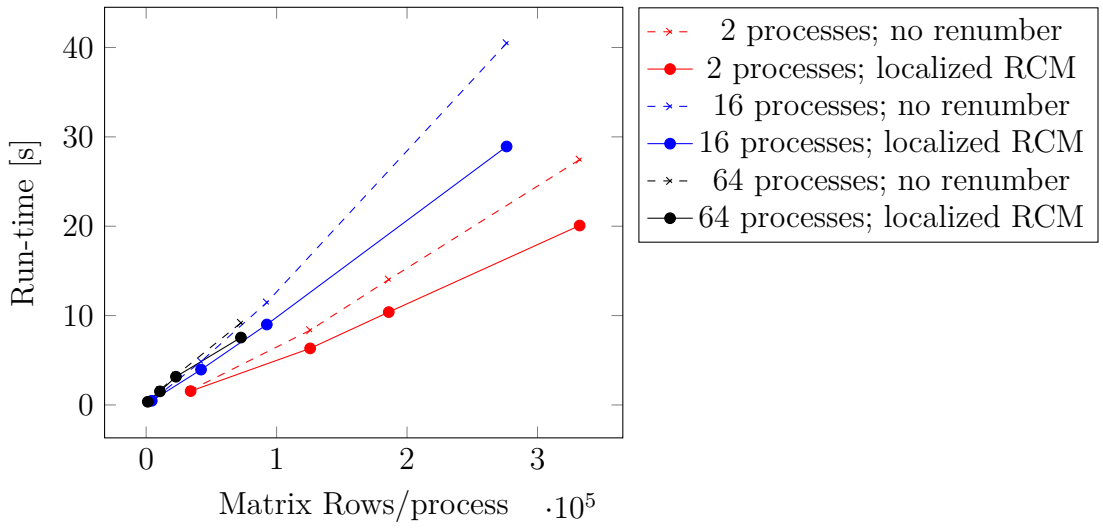


Figure 5.22: Run-time of our AMG method for the velocity-pressure system with different numbers of matrix rows per process, different process counts and with and without localized RCM renumbering.

processes and only 436 rows per process[10]. Therefore, we use localized RCM in our Saddle Point AMG method for the coupled velocity-pressure systems.

Remark 5.11. The *PT-SCOTCH* method has shown to be not as efficient mainly because of its *setup phase*, meaning the computation of a renumbering and also the redistribution of data across the MPI network. This cost could be reduced severely by incorporating this strategy into the discretization method (the FPM in our case) itself. In other words, if the point cloud was distributed across the processes with a minimized communication halo in the first place, it would not be necessary to redistribute the linear system in the linear solver. Because of the many aspects in the point cloud management, especially the changing neighborhoods, the migration of points from one process to the other and also the inserting and merging of points, this is a highly non-trivial task and beyond the scope of this work. A similar argument holds for the bandwidth-reducing RCM methods: If the points were already numbered in a *bandwidth-reducing* fashion (i.e. by applying RCM to the point cloud itself, using the neighborhood relationships as an adjacency matrix), RCM would not have to be applied in the linear solver. Again, it would be difficult to maintain such a numeration of points across time steps though, due to the point cloud management phase. On the other hand, this would also give benefits in the rest of the FPM.

5.4.5 Parallel Scalability of the New Method

An important property of a parallel linear solver is its scalability with the number of processes being used. In this experiment we solve for the hydrostatic pressure in the bifurcated tube model with $\approx$ 14m points. Each time, the system is solved with a

[10]Not shown in the plot for better readability.

Processes	t_{set} [s]	t_{cyc} [s]	t_{tot} [s]	ρ
32	2.844	0.185	5.805	0.241 (16)
64	1.617	0.099	3.102	0.218 (15)
128	0.961	0.055	1.727	0.189 (14)
256	0.695	0.032	1.180	0.204 (15)
512	1.664	0.043	2.266	0.192 (14)

Figure 5.23: AMG timings for: setup (t_{set}), one cycle (t_{cyc}) and for the overall method (t_{tot}); convergence rates (ρ) and iteration counts (in brackets) are fairly constant.

relative residual reduction of $\epsilon_{\text{AMG}} = 10^{-8}$ and the time measured includes the AMG setup. As we can see in Figure 5.23, both the setup time and the overall run-time scale decently up to 256 processes in this case. Because the number of iterations is almost constant, this translates to a decent speed-up in the total solver run-time as well. Note that the 256 processes case amounts to $\approx$ 55,000 matrix rows per process, whereas in the setting in [99] we were able to achieve good speed-ups down to $\approx$ 25,000 rows per process, which would be equivalent to the 512 processes case here. The main difference between the results in [99] and the results here is the increase in the setup time going from 256 processes to 512 processes in Figure 5.23. This was caused by the fact that in the 512 processes case here, 6 AMG levels are build rather than 5 levels in the 256 processes case, leading to larger costs in the setup. A recommended row count per process in the SAMG software library is 100,000 and both results are well below that mark. It is worth noting though that scaling AMG methods to thousands and tens of thousands of processes is ongoing research and both the numerical as well as the computational implications of such core counts must not be underestimated. The author of this thesis is not aware of any simulations using meshfree GFDMs running on such process counts at the moment though. Therefore, within the scope of this thesis, the scaling properties of our methods are decent.

Chapter 6

Final Experiments Showing the Benefits in Real World Cases

The previous chapters have established a new AMG method for the solution of all kinds of linear systems arising in the FPM. The method is efficient and robust in terms of the linear subsystems that can arise, including potentially singular subsystems. It is also parallelized and we have examined the scaling properties in both the number of matrix rows and the process count.

In this last chapter of the thesis we use the FPM together with our newly developed AMG method to solve some models of industrial size. We show that with the work in this thesis, it is now possible to carry out simulations that were not possible before because they took prohibitively long to run.

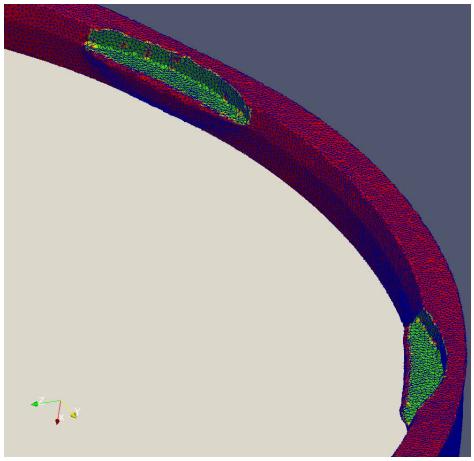
6.1 Crimping

Crimping is a process to join two (mostly metal) pieces together. This is achieved by deforming one or both of the pieces using a tool. Figure 6.1 shows the crimping of a chip (white disc) and a metal ring (red/green). The figure does not show the tool, which is applying force to the green areas, which are deforming in return. The questions this simulation aims to answer include

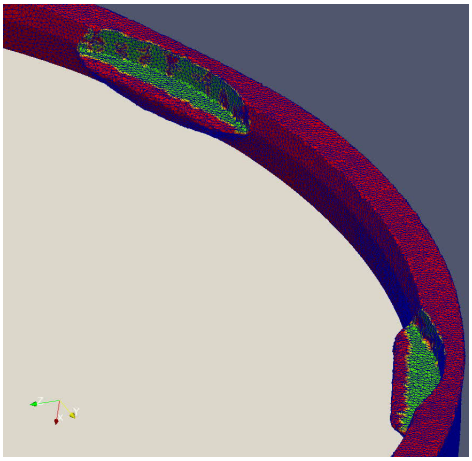
- the shape of the crimps and
- the force needed to push the chip out of the ring.

The process and the material model are described in [157] and [158]. The main idea here is to use the Johnson and Cook model [69] for the viscosity μ in equation (2.35).

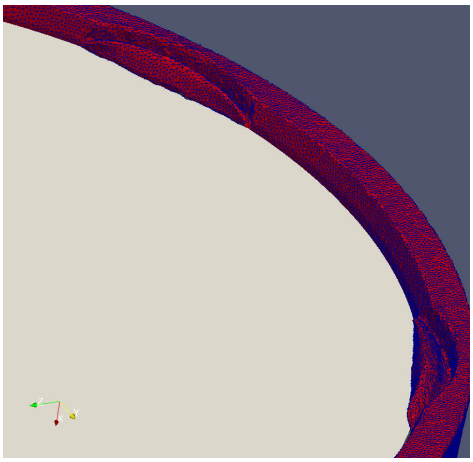
Using the AMG techniques described in this thesis for both the coupled velocity-pressure systems and the pressure systems, we were able to reduce the run-time for this model by a factor of 4.9x compared to the one-level BiCGStab2 solver. For this experiment, we simulated the process using 267k points on 64 processes, which makes around 4k points per process. Although this is not a lot of points per process, using 64 processes is a good idea because in the coupled velocity-pressure system can be quite expensive to solve, especially when using the BiCGStab2 solver. We use the coupled approach with a virtual time step size of $0.01\Delta t$, which means that we use the Saddle



(a) The tool crimps the material (red/green) ...



(b) ... until the chip (white) is fully secured.



(c) Then, the chip is pushed back against the crimps.

Figure 6.1: The crimping process.

Point AMG method introduced in Section 4.3.

Then looking at the linear solver alone, the improvement is 7.4x. The solver run-time went down from 93% of the overall run-time to 65%. Having said this, there is still room there for further improvement in those 65%, if we were to fine-tune the AMG method to this specific problem. However, our goal here was to develop a method that is both efficient and robust, which sometimes means sacrificing some efficiency for robustness. While BiCGStab2 needs around 500 iterations in this model to solve the coupled velocity-pressure system, our AMG method solves those systems in less than 10 iterations. Hence, most of the AMG time (around 75%) is spent in the setup phase. We can improve the speed-up even more when re-using the setup for the coupled velocity-pressure systems across time steps. This comes at the cost of a reduced accuracy of the simulation as we have seen in Chapter 4, though.

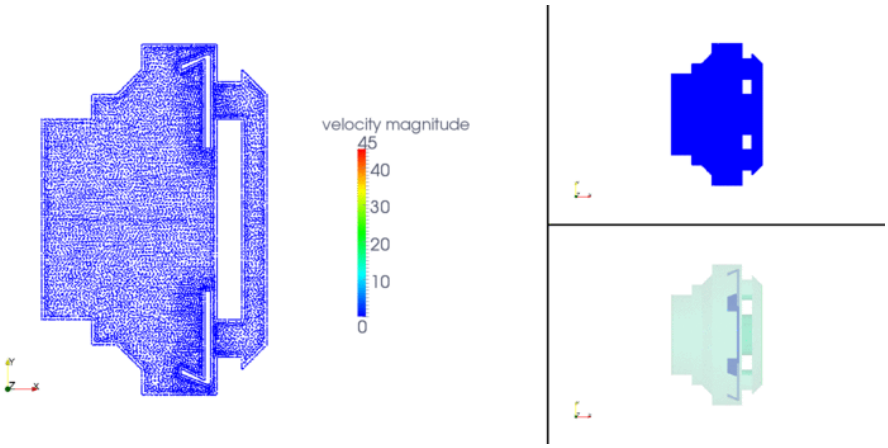
6.2 Valve

A valve is completely filled with oil at a temperature of 90°C. There is a pressure difference between inflow and outflow of 150 kPa. Due to this pressure difference the plate which is inside the valve starts to move. The movement of the plate is simulated with the help of rigid body movement, so we do not model fluid structure interaction here. The objective of this simulation is to determine how long it takes until a static flow is established.

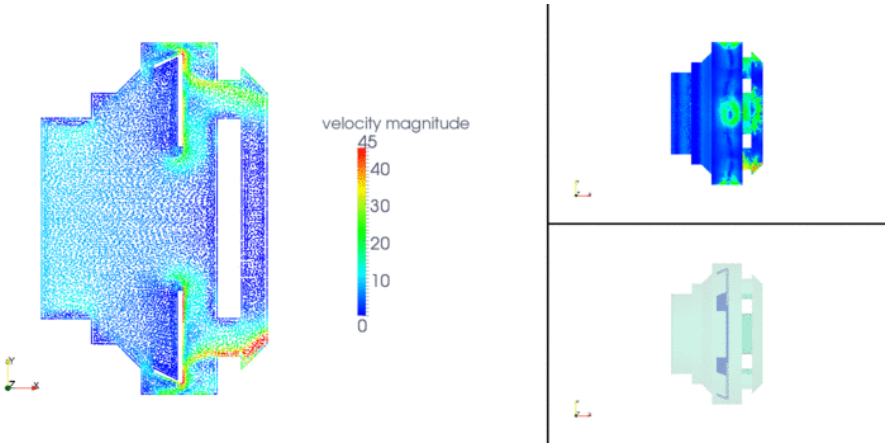
Because of the viscosity of the oil, the coupled FPM approach has to be used here. The virtual time step size is $0.33\Delta t$ and the smoothing length is $h = 0.0004$ near the plate and $h = 0.0005$ away from the plate, leading to about 659k points. Note that this means we have about 2.6m rows in the coupled velocity-pressure system. We use 8 processes for this problem.

If we use a standard AMG method without any treatment of independent linear subsystems here, the linear solver fails to converge when the first points enter the small gap between the plate and the outer hull of the valve. This is because those points quickly form a singular linear subsystem, as described in Section 5.2.

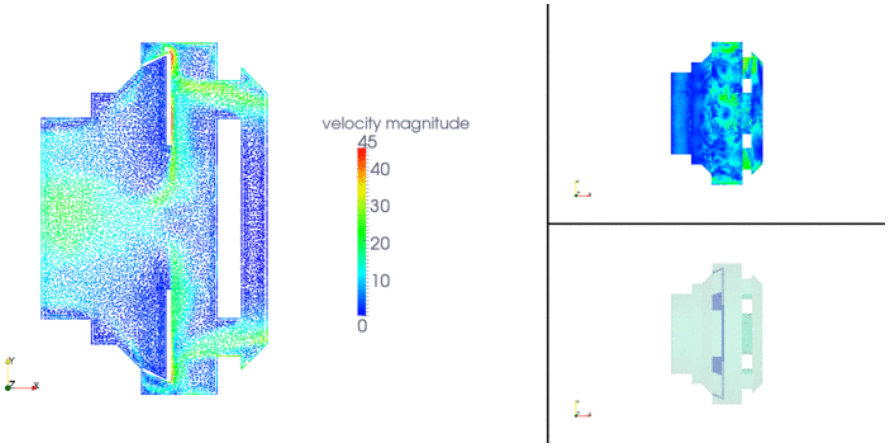
With our new method, this subsystem is detected and removed from the large linear system. One of the pressure variables in this system is fixed to 1 and a direct solver is used to solve the regularized system. With the AMG techniques presented in this thesis, the speed-up over a one-level BiCGStab2 solver is 2.9x. This reduces the percentage spent in the linear solver from 17% to only 6%. This is one of the simulations where the linear solver is not the main bottleneck. Some reasons for this are explained in the outlook in the next chapter.



(a) Initial state.



(b) Half way open.



(c) Open.

Figure 6.2: Valve opening. Each figure shows the velocity flow field on the left-hand side, the pressure field at the top right and the geometry without any point cloud at the bottom right. Image by Fraunhofer ITWM.

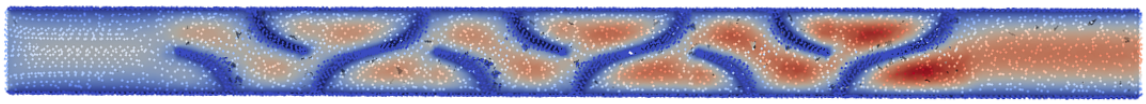


Figure 6.3: Velocities in a static Kenics mixing device with the fluid flow calculated by the FPM.

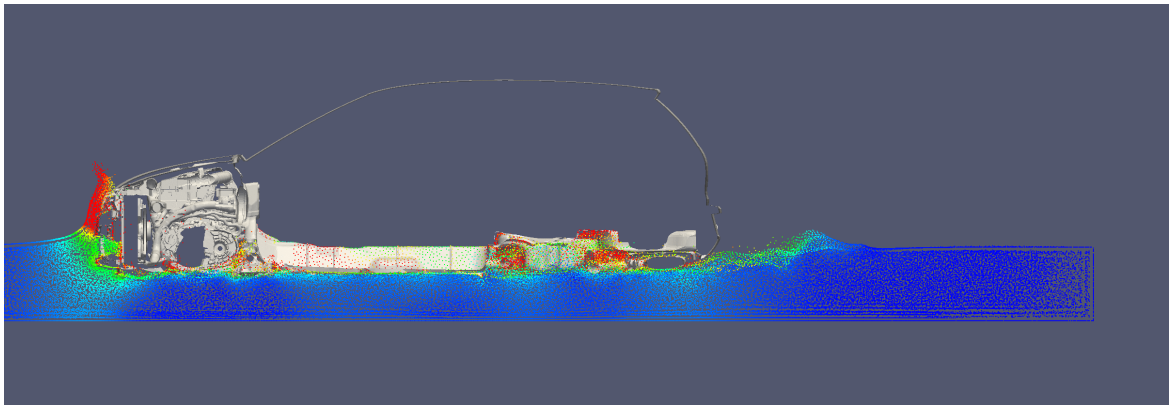


Figure 6.4: Watercrossing with the full engine compartment being modeled. Image by Fraunhofer ITWM.

6.3 Static Mixing

In this simulation, the fluid flow through a static mixing device is simulated. The so-called Kenics mixer [60] is used to mix two fluids, but here we are only interested in the velocity profile of a single fluid. The fluid has a density of 10 kg/m^3 and a viscosity of 0.08 Pa $\cdot$ s. The diameter of the mixer is 5.08cm and the inflow velocity is 1.00 m/s. This leads to a Reynolds number of 6 which means that in this simulation, the segregated approach will deliver accurate results. Figure 6.3 shows the velocity profile computed with a simulation using 1.05m points on 32 processes.

With the new AMG method, we see an improvement of the overall run-time of 6.1x and an improvement of 33.7x in the linear solver. This brings down the linear solver portion of the overall run-time from 71% to 13%, removing the linear solver as a bottleneck.

6.4 Watercrossing

There are different applications in the automotive industry that involve water and sometimes air. The first application here is a car driving through a pool of water of a certain depth. In these simulations one of the questions that can be answered is whether some of the water is sucked into the air intake for the engine or for the air conditioning system. Other questions include the soiling of the vehicle body or the exterior mirrors with dirt. When looking into autonomous driving, the contamination

of different sensors also becomes an issue.

An example from such a simulation can be seen in Figure 6.4. In this simulation, the car's geometry includes the engine compartment in order to examine the effects of water entering from below the engine as well. Keep in mind that with the FPM it is unnecessary to mesh the inside of the engine compartment, simply because the point cloud will adopt to the geometry once the car enters the water. Depending on the question that has to be answered, such simulations can be carried out with or without the air surrounding the car being simulated as well. Simulations that do not model the air as well are often faster but less accurate. For the experiment shown here, the air was not modeled.

Remark 6.1. Although this model includes surface tension, we will not go into the modeling of surface tension using the FPM here. Details on that can be found in [153]. When comparing models with and without surface tension, one does not find a significant difference in terms of the convergence of our AMG method. In [99] one of the experiments was run without surface tension and delivered good results as well.

The smoothing length in this simulation varies between $h = 0.04$ near the car itself and $h = 0.3$ away from the car, with a linear increase between those two values. This leads to an overall number of points of around 2.8m, which leads to around 43k points per process on 64 processes. Because of the fairly high Reynolds number in this model, the segregated approach is appropriate and the coupled approach is not needed. Therefore, each time step involves the solution of three pressure systems and one velocity system. As we have seen in the thesis, the velocity system can be solved easily using BiCGStab2. For the pressure systems however, we get an improvement of 2.6x when using our new AMG method. However, this improvement only translates to an improvement of 1.2x in the overall simulation run-time, because even the BiCGStab2 solver in the original version only takes up 40% of the run-time. Some reasons for that are discussed in Chapter 7. Although 1.2x does not sound a lot, it brings the run-time for the simulation of 10s of real world time from 14.17 days down to 12.5 days.

6.5 Rainwater Management

Rainwater management is another application from the automotive industry involving water and air. Here, the great benefit of the FPM is in the easy representation of rain drops using the point cloud. With mesh-based methods, these simulations would only be possible using Volume of Fluid methods. Adapting the mesh to represent the rain drops exactly would be nearly impossible. With the FPM, we can simply *inject* rain drops – i.e. small clusters of points – at the inflow that then deform, merge and move naturally as the points move. This simulation also involves the air surrounding the car, i.e. this is a multiphase flow simulation [153]. Figure 6.5 shows this scenario with an openly available geometry of a McLaren F1 LM at 60 km/h. The rain is injected at the bottom end of the windshield. The screenshot shown here is taken at 0.8503 s. With a smoothing length of $h = 0.15$ near the car and $h = 0.75$ away from the car, we end up with around 68m points. The number of points changes over the

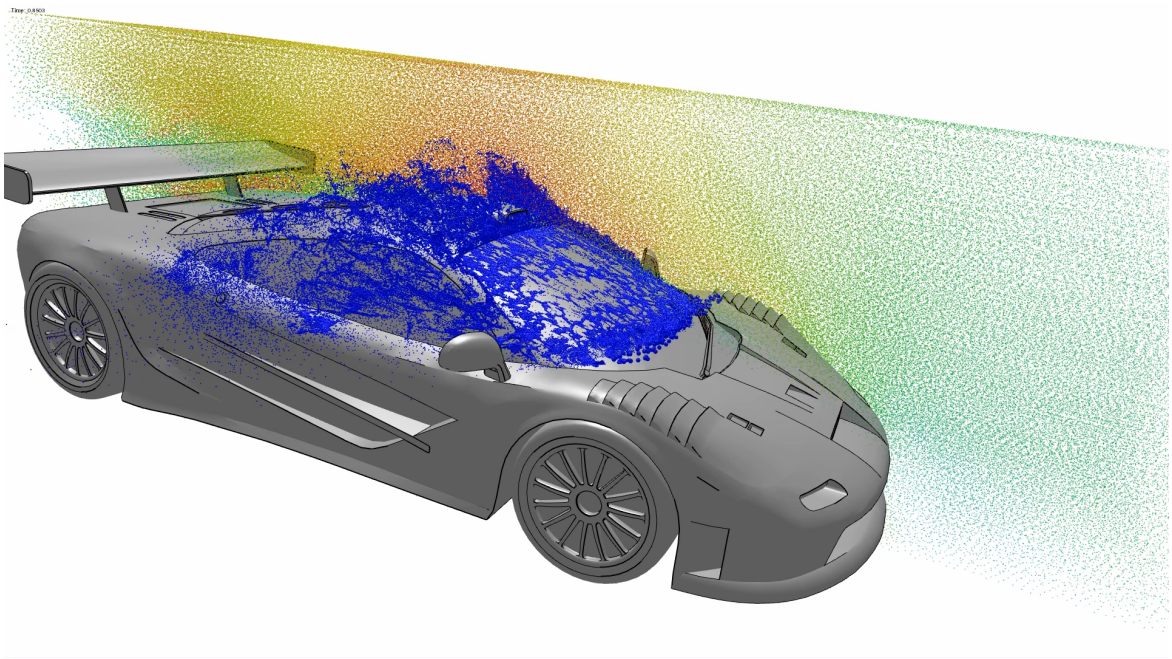


Figure 6.5: McLaren F1 in heavy rain at 60 km/h. Image by Fraunhofer ITWM.

course of the simulation, because of the water being injected. But the main source of complexity here is the air, which contributes the majority of the points. Therefore, we use 256 processes in this experiment, which makes 265k points per process. With such a detailed simulation, the benefits of our new AMG method are much more significant compared to the watercrossing simulation in the last section. Although we only need to solve two pressure systems per time step because we do not take gravity into account here[1], the speed-up in the linear solver compared to BiCGStab2 is still 8.2x. Because the BiCGStab2 solver takes up 85% of the run-time in this case, the overall speed-up is 4.6x. In this case, this means that we can bring down the simulation run-time for one second of physical time down from 16.1 days to only 3.5 days. With the AMG method presented here, the linear solver now takes up only 26% of the overall run-time.

[1] With the water being injected downwards and the car driving forward at 60 km/h, gravity is negligible when we are just interested at the general direction of the water after hitting the wind shield.

Chapter 7

Summary and Outlook

The goal of this thesis was to develop a robust and efficient Algebraic Multigrid method for the solution of linear systems arising in Generalized Finite Difference Methods. With the FPM, one particular method using a GFDM served as a demonstrator showing the benefits of our new AMG method. Our AMG method makes simulations with the FPM much faster, as we have shown in Chapter 6.

We have seen that the application of AMG is not straight forward. The special properties of the linear systems arising from GFDMs need to be taken into account. To this end, this thesis gave a comprehensive analysis of both the pressure systems and the coupled velocity-pressure systems. It also showed that classical one-level solvers and multigrid methods are not efficient enough in many situations. Therefore, the development of a new AMG method was motivated by developing a Geometric Multigrid method for point clouds first. Here we saw that some fundamental ideas of multigrid methods carry over to the more general situation of GFDMs.

Since AMG methods are in many ways more robust than geometric methods, the rest of the thesis dealt with the development of a new AMG method tailored to the linear systems arising in the FPM. This method needed to take into account all the properties that were studied before. More specifically, the matrix had to be scaled appropriately before being passed to the linear solver. Even more importantly, singular subsystems needed to be eliminated from the full matrix. To this end, we developed a parallel graph-component detection algorithm with linear complexity in the average case, which we verified both theoretically and experimentally. We also discussed how the results of this algorithm are used in order to re-distribute the subsystems to the different linear solvers. For the AMG method itself, we found the most efficient and robust techniques for both the pressure and the coupled velocity-pressure systems. We found that for the pressure systems, aggressive coarsening strategies need to be employed in order to avoid a slow coarsening and high complexities in the AMG hierarchy. For the coupled velocity-pressure systems in flows with low Reynolds numbers, we needed to employ different variants of specialized Saddle Point AMG strategies, combined with an Alternate-Block-Factorization. We also evaluated the use of renumbering strategies, both affecting the enumeration of rows within a process as well as the re-distribution of rows across processes in order to minimize communication. A localized Reverse Cuthill-McKee method turned out to be most useful at least for the coupled velocity-pressure systems. There is potential in this strategy if it could be

realized within the FPM itself. Lastly, the re-use of the AMG setup phase was a big topic in this thesis. While the setup phase can be re-used quite easily for the different pressure systems within each time step, the re-use across multiple time steps is much more involved, because of the moving character of the FPM point cloud with points being merged or added in between time steps. We introduced multiple strategies to deal with this issue and ended up modifying the FPM point cloud management itself, allowing for the AMG setup to be re-used, with some impact on the accuracy of the simulation though. On the other hand, this strategy also yielded some performance benefits in the FPM itself.

This thesis has therefore contributed to both the extension of the applicability of AMG methods to new problems, as well as to the improvement of the performance of the FPM. Combined, these contributions extend the relevance of the FPM in industrial applications.

The results shown in this thesis are only a selection of cases in which the application of AMG yields benefits for the overall simulation run-time. Many problems fall into the categories of problems where AMG can substantially improve the performance. On the other hand though, in some examples, and also in the velocity systems for the most part, applying AMG does not yield any or only a very limited benefit over the classical one-lever iterative solvers. There can be multiple reasons for this:

1. Classical one-level solvers only need a very low number of iterations.

2. The fraction of the simulation that is spent in the linear solver is small compared to other components like the point cloud management.

Often times, (1) implies (2). In order to increase the performance in both cases, it is necessary to reconsider and improve other parts of the FPM algorithm. This is especially true for the parallelization aspects of the method, parts of which are significantly more involved than in mesh-based methods. There are a number of tasks in the FPM algorithm that are prone to introducing load balance issues, for example the computation of distances between points and the boundaries of the domain: Every point needs to compute its distance to every triangle of the (triangulated) geometry within a certain radius. In cases with highly irregular geometries, the number of triangles within this radius can vary highly between different points. If a few processes happen to hold a lot of points with many triangles around them, these processes will need longer to compute the distances of all their points to the boundary compared to those processes that only hold points that are close to a very simple boundary or even no boundary at all. This issue is greatly simplified in the case of a static mesh, because distances are not changing and the distribution of nodes across processors stays constant over time, whereas in Lagrangian methods like the FPM it is subject to change. Therefore, the issue of load balancing for Lagrangian GFDMs should be a topic of further research.

But there is also room for improvement in our new AMG method: The benefits of re-using the setup could be enhanced by a *locally* new setup, i.e. by keeping most parts

of the setup, but re-constructing a new setup where points have been merged or inserted. This would be a completely new development in the context of AMG methods. It would allow the partial re-use of the setup phase despite the point cloud management being applied between time steps. Hence, performance improvements could be made without sacrificing the accuracy of the simulation. The setup phase could also profit from using strategies like PMIS and sparsification, that we touched on briefly in Section 5.1. Lastly, the redistribution of independent components across processes should be limited to a necessary minimum in order to reduce communication overhead.

It has become clear in the present work that the development of the method (FPM) and the development of the underlying linear solver (AMG) should go hand in hand in order to achieve optimal performance.

Appendices

Appendix A

Fornberg's Algorithm in Matlab

```
function d = fornberg(M, x0, a)

N=numel(a)-1;
d(1:N+1,1:N+1,1:M+1) = 0;
d(1,1,1) = 1;
c1 = 1;

for n = 2:N+1
 c2 = 1;
 for nu = 1:n-1
  c3 = a(n) - a(nu);
  c2 = c2*c3;
  if ( n <= M+1 )
   d(nu,n-1,n) = 0;
  end
  for m = 1:min(n,M)+1
   if ( m > 1 )
    d(nu,n,m) = ((a(n) - x0) * d(nu,n-1,m) - (m-1) *
                * d(nu,n-1,m-1))/c3;
   else
    d(nu,n,m) = ((a(n) - x0) * d(nu,n-1,m) - 0)/c3;
   end
  end
 end
 for m = 1:min(n,M)+1
  if ( m > 1 )
   d(n,n,m) = c1/c2 * ( (m-1)*d(n-1,n-1,m-1) - (a(n-1) - x0) *
              * d(n-1,n-1,m));
  else
   d(n,n,m) = c1/c2 * ( 0 - (a(n-1) - x0) * d(n-1,n-1,m));
  end
 end
 c1=c2;
end
```

Appendix B

Hardware and Software Used for Benchmarks

The experiments in this thesis using the FPM were carried out with the MESHFREE software developed by Fraunhofer ITWM and Fraunhofer SCAI. The software version used was beta19.02.0.

In tests where matrix-based AMG benchmarks were conducted, the SAMG library (release version 2018) developed by Fraunhofer SCAI was used.

Both software packages were compiled using the Intel Fortran compiler version 18.0.3 20180410 and the Intel(R) MPI Library for Linux* OS, Version 2018 Update 3 Build 20180411.

The Multicloud approach in Section 3.3.2 was developed using MATLAB. The source code for that approach can be found on Github at `https://github.com/fpnick/ofpm`, although it should be mentioned that this code is neither maintained not meant for usage by anyone else other than the author of this thesis.

Hardware wise, the non-MATLAB experiments were carried out on the LOEWENBURG cluster at Fraunhofer SCAI consisting of 128 compute nodes with 1xIntel Xeon Gold 6130F@2.10GHz + 1xIntel Xeon Gold 6130@2.10GHz, 192 GB RAM on each node. The cluster is equipped with an Omni-Path interconnect with 100Gb/s ports. At the time of writing this, the cluster was running CentOS Linux release 7.4.1708.

Bibliography

[1] M. Adams, M. Brezina, J. Hu, and R. Tuminaro, *Parallel multigrid smoothing: polynomial versus Gauss–Seidel*, Journal of Computational Physics, 188 (2003), pp. 593–610.

[2] M. F. Adams, *Algebraic multigrid methods for constrained linear systems with applications to contact problems in solid mechanics*, Numerical linear algebra with applications, 11 (2004), pp. 141–153.

[3] A. Andoni and P. Indyk, *Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions*, Communications of the ACM, 51 (2008), p. 117.

[4] I. Babuška, U. Banerjee, and J. E. Osborn, *Survey of meshless and generalized finite element methods: A unified approach*, Acta Numerica, 12 (2003), pp. 1–125.

[5] ——, *Generalized finite element method - main ideas, results, and perspective*, International Journal of Computational Methods, 1 (2004), pp. 67–103.

[6] I. Babuška and J. M. Melenk, *The partition of unity method*, International journal for numerical methods in engineering, 40 (1997), pp. 727–758.

[7] A. H. Baker, T. V. Kolev, and U. M. Yang, *Improving algebraic multigrid interpolation operators for linear elasticity problems*, Numerical Linear Algebra with Applications, 17 (2010), pp. 495–517.

[8] T. J. Baker, *Automatic mesh generation for complex three-dimensional regions using a constrained delaunay triangulation*, Engineering with Computers, 5 (1989), pp. 161–175.

[9] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, and H. Zhang, *PETSc Web page*. `http://www.mcs.anl.gov/petsc`, 2018.

[10] ——, *PETSc users manual*, Tech. Rep. ANL-95/11 - Revision 3.10, Argonne National Laboratory, 2018.

[11] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, *Efficient management of parallelism in object oriented numerical software libraries*, in Modern Software Tools in Scientific Computing, E. Arge, A. M. Bruaset, and H. P. Langtangen, eds., Birkhäuser Press, 1997, pp. 163–202.

[12] G. Ballard, C. Siefert, and J. Hu, *Reducing communication costs for sparse matrix multiplication within algebraic multigrid*, SIAM Journal on Scientific Computing, 38 (2016), pp. C203–C231.

[13] T. Banachiewicz, *Méthode de résolution numérique des équations linéaires, du calcul des déterminants et des inverses, et de réduction des formes quadratiques*, Bull. Intern. Acad. Polon. Sci. A, (1938), pp. 393–401.

[14] R. E. Bank, T. F. Chan, W. Coughran, and R. K. Smith, *The alternate-block-factorization procedure for systems of partial differential equations*, BIT Numerical Mathematics, 29 (1989), pp. 938–954.

[15] R. E. Bank, B. D. Welfert, and H. Yserentant, *A class of iterative methods for solving saddle point problems*, Numerische Mathematik, 56 (1989), pp. 645–666.

[16] M. Benzi and G. H. Golub, *A preconditioner for generalized saddle point problems*, SIAM Journal on Matrix Analysis and Applications, 26 (2004), pp. 20–41.

[17] M. Benzi, G. H. Golub, and J. Liesen, *Numerical solution of saddle point problems*, Acta numerica, 14 (2005), pp. 1–137.

[18] L. S. Blackford, A. Petitet, R. Pozo, K. Remington, R. C. Whaley, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, et al., *An updated set of basic linear algebra subprograms (BLAS)*, ACM Transactions on Mathematical Software, 28 (2002), pp. 135–151.

[19] S. L. Borne and L. Grasedyck, *H-matrix preconditioners in convection-dominated problems*, SIAM Journal on Matrix Analysis and Applications, 27 (2005), pp. 1172–1183.

[20] D. Braess, *Towards algebraic multigrid for elliptic problems of second order*, Computing, 55 (1995), pp. 379–393.

[21] A. Brandt, *Multigrid techniques*, Arbeitspapiere der GMD, 85 (1984).

[22] ——, *Algebraic multigrid theory: The symmetric case*, Applied Mathematics and Computation, 19 (1986), pp. 23–56.

[23] A. Brandt, S. McCormick, and J. Ruge, *Algebraic multigrid (AMG) for sparse matrix equations*, Evans, D.: Sparsity and its Applications, (1985).

[24] J. S. Chen, M. Hillman, and S. W. Chi, *Meshfree methods: Progress made after 20 years*, Journal of Engineering Mechanics - ASCE, 143 (2017).

[25] C. Chevalier and F. Pellegrini, *PT-Scotch: A tool for efficient parallel graph ordering*, Parallel computing, 34 (2008), pp. 318–331.

[26] E. K.-y. Chiu, Q. Wang, R. Hu, and A. Jameson, *A conservative mesh-free scheme and generalized framework for conservation laws*, SIAM Journal on Scientific Computing, 34 (2012), pp. A2896–A2916.

[27] E. K.-y. Chiu, Q. Wang, and A. Jameson, *A conservative meshless scheme: general order formulation and application to Euler equations*, in 49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition, 2011, p. 651.

[28] A. J. Chorin, *A numerical method for solving incompressible viscous flow problems*, Journal of Computational Physics, 2 (1967), pp. 12–26.

[29] T. Clees, *AMG Strategies for PDE Systems with Applications in Industrial Semiconductor Simulation*, PhD thesis, Universität Köln, 2005.

[30] P. Culière, A. Trameçon, G. Pierrot, and J. Kuhnert, *Finite point set method: a mesh free approach to model airbag inflation*, tech. rep., SAE Technical Paper, 2003.

[31] S. J. Cummins and M. Rudman, *An SPH projection method*, Journal of computational physics, 152 (1999), pp. 584–607.

[32] E. F. D'Azevedo, M. R. Fahey, and R. T. Mills, *Vectorized sparse matrix multiply for compressed row storage format*, in International Conference on Computational Science, Springer, 2005, pp. 99–106.

[33] A. Donev, *Connected components of a graph*. http://computation.pa.msu.edu/NO/ConnCompPresentation.html (Accessed 02/16/18).

[34] J. J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson, *Algorithm 656: An extended set of basic linear algebra subprograms: Model implementation and test programs*, ACM Transactions on Mathematical Software (TOMS), 14 (1988), pp. 18–32.

[35] C. Drumm, S. Tiwari, J. Kuhnert, and H.-J. Bart, *Finite pointset method for simulation of the liquid–liquid flow field in an extractor*, Computers & Chemical Engineering, 32 (2008), pp. 2946–2957.

[36] J. A. Edwards and U. Vishkin, *Better speedups using simpler parallel programming for graph connectivity and biconnectivity*, in Proceedings of the 2012 International Workshop on Programming Models and Applications for Multicores and Manycores, ACM, 2012, pp. 103–114.

[37] R. D. Falgout and J. B. Schroder, *Non-Galerkin coarse grids for algebraic multigrid*, SIAM Journal on Scientific Computing, 36 (2014), pp. C309–C334.

[38] M. Faustmann, J. M. Melenk, and D. Praetorius, *$\mathcal{H}$-matrix approximability of the inverses of FEM matrices*, Numerische Mathematik, 131 (2015), pp. 615–642.

[39] L. K. Fleischer, B. Hendrickson, and A. Pinar, *On identifying strongly connected components in parallel*, in International Parallel and Distributed Processing Symposium, Springer, 2000, pp. 505–511.

[40] B. Fornberg, *Generation of finite difference formulas on arbitrarily spaced grids*, Mathematics of computation, 51 (1988), pp. 699–706.

[41] T.-P. Fries and T. Belytschko, *The extended/generalized finite element method: An overview of the method and its applications*, International journal for numerical methods in engineering, 84 (2010), pp. 253–304.

[42] C. Gauss, *Geodäsie. Fortsetzung von Band 4*, 1903.

[43] S. A. Gershgorin, *Über die Abgrenzung der Eigenwerte einer Matrix*, (1931), pp. 749–754.

[44] R. A. Gingold and J. J. Monaghan, *Smoothed particle hydrodynamics: theory and application to non-spherical stars*, Monthly notices of the royal astronomical society, 181 (1977), pp. 375–389.

[45] G. H. Golub and C. F. Van loan, *Matrix computations*, The Johns Hopkins, (1996).

[46] M. Griebel, D. Oeltz, and M. A. Schweitzer, *An algebraic multigrid method for linear elasticity*, SIAM J. Sci. Comp, 25 (2003), p. 407.

[47] S. Gries, *Algebraische mehrgitteransätze für industrierelevante cfd-probleme - druckkorrektur im kontext openfoam*, diplomarbeit, Universität Köln, 2011.

[48] S. Gries, *System-AMG Approaches for Industrial Fully and Adaptive Implicit Oil Reservoir Simulations*, PhD thesis, Universität zu Köln, 2015.

[49] M. H. Gutknecht, *Variants of bicgstab for matrices with complex spectrum*, SIAM journal on scientific computing, 14 (1993), pp. 1020–1033.

[50] W. Hackbusch, *On the multi-grid method applied to difference equations*, Computing, 20 (1978), pp. 291–306.

[51] ——, *Multi-Grid Methods and Applications*, Springer, 1985.

[52] W. Hackbusch, *Iterative Solution of Large Sparse Systems of Equations*, Springer, 1994.

[53] W. Hackbusch, *A sparse matrix arithmetic based on H-matrices. Part I: Introduction to H-matrices*, Computing, 62 (1999), pp. 89–108.

[54] W. Hackbusch and T. Probst, *Downwind Gauss–Seidel smoothing for convection dominated problems*, Numerical linear algebra with applications, 4 (1997), pp. 85–102.

[55] S. Hagmann, *Adaptive Verfeinerungsverfahren in der gitterfreien Finite-Pointset-Methode (FPM)*, PhD thesis, Institut für Höchstleistungsrechnen (IHR) der Universität Stuttgart, 2019.

[56] V. E. Henson and U. M. Yang, *BoomerAMG: A parallel algebraic multigrid solver and preconditioner*, Applied Numerical Mathematics, 41 (2002), pp. 155 – 177. Developments and Trends in Iterative Methods for Large Systems of Equations - in memorium Rudiger Weiss.

[57] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, et al., *An overview of the Trilinos project*, ACM Transactions on Mathematical Software (TOMS), 31 (2005), pp. 397–423.

[58] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*, vol. 49, NBS Washington, DC, 1952.

[59] D. S. Hirschberg, A. K. Chandra, and D. V. Sarwate, *Computing connected components on parallel computers*, Communications of The ACM, 22 (1979), pp. 461–464.

[60] D. Hobbs and F. Muzzio, *The kenics static mixer: a three-dimensional chaotic flow*, Chemical Engineering Journal, 67 (1997), pp. 153–166.

[61] S. Hong, N. C. Rodia, and K. Olukotun, *On fast parallel detection of strongly connected components (SCC) in small-world graphs*, in Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis on, 2013, p. 92.

[62] A. Huerta, T. Belytschko, S. Fernández-Méndez, T. Rabczuk, X. Zhuang, and M. Arroyo, *Meshfree methods*, Encyclopedia of Computational Mechanics Second Edition, (2018), pp. 1–38.

[63] D. Huynh and T. Belytschko, *The extended finite element method for fracture in composite materials*, International Journal for Numerical Methods in Engineering, 77 (2009), pp. 214–239.

[64] M. Ihmsen, J. Cornelis, B. Solenthaler, C. Horvath, and M. Teschner, *Implicit incompressible SPH*, Visualization and Computer Graphics, IEEE Transactions on, 20 (2014), pp. 426–435.

[65] A. Iske, *On the construction of mass conservative and meshless adaptive particle advection methods*, Advances in Meshfree Techniques, (2007), pp. 169–186.

[66] J. Iverson, C. Kamath, and G. Karypis, *Evaluation of connected-component labeling algorithms for distributed-memory systems*, parallel computing, 44 (2015), pp. 53–68.

[67] G. Jaumann, *Geschlossenes System physikalischer und chemischer Differentialgesetze (I. Mitteilung)*, Sitzungsber. der kaiserliche Akad. der Wissenschaften in Wien, CXVII (Mathematisch IIa), 385 (1911).

[68] A. Jefferies, J. Kuhnert, L. Aschenbrenner, and U. Giffhorn, *Finite pointset method for the simulation of a vehicle travelling through a body of water*, in Meshfree Methods for Partial Differential Equations VII, Springer, 2015, pp. 205–221.

[69] G. Johnson and W. Cook, *A constitutive model and data for metal subjected to large strains, high stain rates and high temperatures*, in Symposium on Ballestics (The Hague, the Netherlands), vol. 541, 1983.

[70] K. I. Karantasis, A. Lenharth, D. Nguyen, M. J. Garzarán, and K. Pingali, *Parallelization of reordering algorithms for bandwidth and wavefront reduction*, in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE Press, 2014, pp. 921–932.

[71] A. Katz and A. Jameson, *Multicloud: Multigrid convergence with a meshless operator*, Journal of Computational Physics, 228 (2009), pp. 5237–5250.

[72] R. Kendall, G. Morrell, D. Peaceman, W. Silliman, J. Watts, et al., *Development of a multiple application reservoir simulator for use on a vector computer*, in Middle East Oil Technical Conference and Exhibition, Society of Petroleum Engineers, 1983.

[73] R. Kettler, *Analysis and comparison of relaxation schemes in robust multigrid and preconditioned conjugate gradient methods*, in Multigrid methods, Springer, 1982, pp. 502–534.

[74] A. Krechel and K. Stüben, *Parallel algebraic multigrid based on subdomain blocking*, Parallel Computing, 27 (2001), pp. 1009–1031.

[75] J. Kuhnert, *General smoothed particle hydrodynamics*, Shaker, 1999.

[76] J. Kuhnert, *Meshfree Numerical Scheme for Time Dependent Problems in Fluid and Continuum Mechanics*, Anne Books, New Delhi, 2014, pp. 119–136.

[77] V. Kumar and V. N. Rao, *Parallel depth first search. Part II. Analysis*, International Journal of Parallel Programming, 16 (1987), pp. 501–519.

[78] P. Lancaster and K. Salkauskas, *Surfaces generated by moving least squares methods*, Mathematics of computation, 37 (1981), pp. 141–158.

[79] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, *Basic linear algebra subprograms for Fortran usage*, ACM Transactions on Mathematical Software (TOMS), 5 (1979), pp. 308–323.

[80] R. J. LeVeque, *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-state and Time-Dependent Problems*, SIAM, 2007.

[81] S. Li and W. K. Liu, *Meshfree and particle methods and their applications*, Applied Mechanics Reviews, 55 (2002), pp. 1–34.

[82] ——, *Meshfree particle methods*, Springer Science & Business Media, 2007.

[83] T. Liszka, C. Duarte, and W. Tworzydlo, *hp-meshless cloud method*, Computer Methods in Applied Mechanics and Engineering, 139 (1996), pp. 263–288.

[84] T. Liszka and J. Orkisz, *The finite difference method at arbitrary irregular grids and its application in applied mechanics*, Computers & Structures, 11 (1980), pp. 83–95.

[85] G.-R. Liu, *Meshfree Methods: Moving Beyond the Finite Element Method*, CRC press, 2009.

[86] M. Liu, W. Xie, and G. Liu, *Modeling incompressible flows using a finite particle method*, Applied mathematical modelling, 29 (2005), pp. 1252–1270.

[87] W.-H. Liu and A. H. Sherman, *Comparative analysis of the Cuthill–McKee and the reverse Cuthill–McKee ordering algorithms for sparse matrices*, SIAM Journal on Numerical Analysis, 13 (1976), pp. 198–213.

[88] O. E. Livne and A. Brandt, *Lean algebraic multigrid (LAMG): Fast graph Laplacian linear solver*, SIAM Journal on Scientific Computing, 34 (2012), pp. B499–B522.

[89] J. Lottes, *Towards Robust Algebraic Multigrid Methods for Nonsymmetric Problems*, Springer, 2017.

[90] L. Lovasz and B. Szegedy, *Limits of dense graph sequences*, Journal of Combinatorial Theory, Series B, 96 (2006), pp. 933–957.

[91] L. B. Lucy, *A numerical approach to the testing of the fission hypothesis*, The astronomical journal, 82 (1977), pp. 1013–1024.

[92] S. Ma and Y. Saad, *Distributed ILU (0) and SOR preconditioners for unstructured sparse linear systems*, Citeseer, 1994.

[93] J.-C. Marongiu, F. Leboeuf, J. Caro, and E. Parkinson, *Free surface flows simulations in Pelton turbines using an hybrid SPH-ALE method*, Journal of Hydraulic Research, 48 (2010), pp. 40–49.

[94] R. McColl, O. Green, and D. A. Bader, *A new parallel algorithm for connected components in dynamic graphs*, in 20th Annual International Conference on High Performance Computing, 2013, pp. 246–255.

[95] W. Mclendon III, B. Hendrickson, S. J. Plimpton, and L. Rauchwerger, *Finding strongly connected components in distributed graphs*, Journal of Parallel and Distributed Computing, 65 (2005), pp. 901–910.

[96] J. A. Meijerink and H. A. van der Vorst, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Mathematics of computation, 31 (1977), pp. 148–162.

[97] J. M. Melenk and I. Babuška, *The partition of unity finite element method: Basic theory and applications*, Computer Methods in Applied Mechanics and Engineering, 139 (1996), pp. 289–314.

[98] B. Metsch, *Algebraic Multigrid (AMG) for Saddle Point Systems*, PhD thesis, Rheinische Friedrich-Wilhelms-Universität Bonn, 2013.

[99] B. Metsch, F. Nick, and J. Kuhnert, *Algebraic multigrid for the finite pointset method*, Computing and Visualization in Science, (submitted).

[100] M. F. Murphy, G. H. Golub, and A. J. Wathen, *A note on preconditioning for indefinite linear systems*, SIAM Journal on Scientific Computing, 21 (2000), pp. 1969–1972.

[101] A. Napov and Y. Notay, *An algebraic multigrid method with guaranteed convergence rate*, SIAM Journal on Scientific Computing, 34 (2012), pp. A1079–A1109.

[102] F. Nick, *Analyse von verschiedenen vergröoberungsstrategien in algebraischen mehrgitterverfahren, insbesondere klassische und aggregative vergröberung und hybride methoden*, diploma thesis, Universität Köln, 2013.

[103] F. Nick, B. Metsch, and H.-J. Plum, *Linear solvers for the finite pointset method*, Recent Advances in Computational Engineering, (2018), p. 89.

[104] Y. Notay, *An aggregation-based algebraic multigrid method*, Electronic Transactions on Numerical Analysis, 37 (2010), pp. 123–146.

[105] ——, *Aggregation-based algebraic multigrid for convection-diffusion equations*, SIAM Journal on Scientific Computing, 34 (2012), pp. 2288–A2316.

[106] Y. Notay and P. S. Vassilevski, *Recrusive Krylov-based multigrid cycles*, Numerical linear algebra with applications, 15 (2008), pp. 473–487.

[107] E. Oñate, S. Idelsohn, O. Zienkiewicz, and R. Taylor, *A finite point method in computational mechanics. Applications to convective transport and fluid flow*, International journal for numerical methods in engineering, 39 (1996), pp. 3839–3866.

[108] H. Oertel and M. Böhle, *Prandtl-Führer durch die Strömungslehre: Grundlagen und Phänomene*, vol. 11, Springer, 2002.

[109] E. Ogbuobiri, W. F. Tinney, and J. W. Walker, *Sparsity-directed decomposition for Gaussian elimination on matrices*, IEEE Transactions on Power Apparatus and Systems, (1970), pp. 141–150.

[110] N. Perrone and R. Kao, *A general finite difference method for arbitrary meshes*, Computers & Structures, 5 (1975), pp. 45–57.

[111] S. Pirzadeh, *An adaptive unstructured grid method by grid subdivision, local remeshing, and grid movement*, in 14th Computational Fluid Dynamics Conference, 1999, p. 3255.

[112] H.-J. Plum, A. Krechel, S. Gries, B. Metsch, F. Nick, M. A. Schweitzer, and K. Stüben, *Parallel algebraic multigrid*, in Scientific Computing and Algorithms in Industrial Simulations, Springer, 2017, pp. 121–134.

[113] T. Rabczuk, S. Bordas, and G. Zi, *On three-dimensional modelling of crack growth using partition of unity methods*, Computers & structures, 88 (2010), pp. 1391–1411.

[114] E. O. Reséndiz-Flores, J. Kuhnert, and F. R. Saucedo-Zendejo, *Application of a generalized finite difference method to mould filling process*, European Journal of Applied Mathematics, (2017), pp. 1–20.

[115] A. Robinson-Mosher, C. Schroeder, and R. Fedkiw, *A symmetric positive definite formulation for monolithic fluid structure interaction*, Journal of Computational Physics, 230 (2011), pp. 1547–1566.

[116] J. Ruge, *Report on ITWM problems.* 11 2015.

[117] J. Ruge and K. Stüben, *Algebraic multigrid (AMG)*, Frontiers in Applied Mathematics, 5 (1986).

[118] Y. Saad, *ILUT: A dual threshold incomplete LU factorization*, Numerical linear algebra with applications, 1 (1994), pp. 387–402.

[119] Y. Saad, *Iterative methods for sparse linear systems*, 2000.

[120] F. R. Saucedo-Zendejo, E. O. Reséndiz-Flores, and J. Kuhnert, *Three-dimensional flow prediction in mould filling processes using a GFDM*, Computational Particle Mechanics, (2019), pp. 1–15.

[121] M. Schäfer and J. Kuhnert, *Finite pointset method for chip formation.* Preprint Fraunhofer ITWM, 2010.

[122] J. Schöberl and W. Zulehner, *On Schwarz-type Smoothers for Saddle Point Problems*, Numerische Mathematik, 95 (2003), pp. 377–399.

[123] M. A. Schweitzer, *A Parallel Multilevel Partition of Unity Method for Elliptic Partial Differential Equations*, PhD thesis, Universität Bonn, 2002.

[124] M. A. Schweitzer, *Meshfree and Generalized Finite Element Methods*, Habilitation, Institute for Numerical Simulation, University of Bonn, 2008.

[125] ——, *Generalizations of the finite element method*, Central European Journal of Mathematics, 10 (2012), pp. 3–24.

[126] B. Seibold, *M-Matrices in Meshless Finite Difference Methods*, PhD thesis, University of Kaiserslautern, 2006.

[127] B. Seibold, *Minimal positive stencils in meshfree finite difference methods for the Poisson equation*, Computer Methods in Applied Mechanics and Engineering, 198 (2008), pp. 592–601.

[128] T. Seifarth, *Numerische Algorithmen für gitterfreie Methoden zur Lösung von Transportproblemen.*, BoD–Books on Demand, 2018.

[129] A. Shakibaeinia and Y.-C. Jin, *MPS mesh-free particle method for multiphase flows*, Computer Methods in Applied Mechanics and Engineering, 229 (2012), pp. 13–26.

[130] J. Shao, H. Li, G. Liu, and M. Liu, *An improved SPH method for modeling liquid sloshing dynamics*, Computers & Structures, 100 (2012), pp. 18–26.

[131] M. S. Shephard and M. K. Georges, *Automatic three-dimensional mesh generation by the finite octree technique*, International Journal for Numerical methods in engineering, 32 (1991), pp. 709–749.

[132] G. L. G. Sleijpen and D. R. Fokkema, *BiCGstab(L) for linear equations involving unsymmetric matrices with complex spectrum*, Electronic Transactions on Numerical Analysis, 1 (1993), pp. 11–32.

[133] G. M. Slota, S. Rajamanickam, and K. Madduri, *BFS and coloring-based parallel algorithms for strongly connected components and related problems*, in IPDPS '14 Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium, 2014, pp. 550–559.

[134] B. Southworth, T. Manteuffel, S. McCormick, S. Munzenmaier, and J. Ruge, *Reduction-based algebraic multigrid for upwind discretizations*, arXiv preprint arXiv:1704.05001, (2017).

[135] B. Southworth, T. Manteuffel, and Ruge, *Nonsymmetric algebraic multigrid based on local approximate ideal restriction (LAIR)*, (2017).

[136] B. S. Southworth and T. A. Manteuffel, *Convergence in norm of nonsymmetric algebraic multigrid*, arXiv preprint arXiv:1806.04274, (2018).

[137] V. Strassen, *Gaussian elimination is not optimal*, Numerische mathematik, 13 (1969), pp. 354–356.

[138] K. Stüben, *An introduction to Algebraic Multigrid*, Academic Press, 2000.

[139] K. Stüben, J. W. Ruge, T. Clees, and S. Gries, *Algebraic multigrid: From academia to industry*, in Scientific Computing and Algorithms in Industrial Simulations, Springer, 2017, pp. 83–119.

[140] K. Stüben and U. Trottenberg, *Multigrid methods: Fundamental algorithms, model problem analysis and applications*, in Multigrid methods, Springer, 1982, pp. 1–176.

[141] P. Suchde, *Conservation and Accuracy in Meshfree Generalized Finite Difference Methods*, PhD thesis, University of Kaiserslautern, 2018.

[142] P. Suchde and J. Kuhnert, *Point cloud movement for fully lagrangian meshfree methods*, Journal of Computational and Applied Mathematics, 340 (2018), pp. 89–100.

[143] P. Suchde and J. Kuhnert, *A fully lagrangian meshfree framework for PDEs on evolving surfaces*, arXiv preprint arXiv:1902.08107, (2019).

[144] P. Suchde, J. Kuhnert, and S. Tiwari, *On meshfree GFDM solvers for the incompressible Navier–Stokes equations*, Computers & Fluids, 165 (2018), pp. 1 – 12.

[145] D. Sulsky, Z. Chen, and H. L. Schreyer, *A particle method for history-dependent materials*, Computer methods in applied mechanics and engineering, 118 (1994), pp. 179–196.

[146] T. Takahashi, Y. Dobashi, I. Fujishiro, T. Nishita, and M. C. Lin, *Implicit formulation for SPH-based viscous fluids*, in Computer Graphics Forum, vol. 34, Wiley Online Library, 2015, pp. 493–502.

[147] T. Takahashi and M. C. Lin, *A multilevel SPH solver with unified solid boundary handling*, in Computer Graphics Forum, vol. 35, Wiley Online Library, 2016, pp. 517–526.

[148] R. Tarjan, *Depth-first search and linear graph algorithms*, SIAM journal on computing, 1 (1972), pp. 146–160.

[149] ——, *Finding dominators in directed graphs*, SIAM Journal on Computing, 3 (1974), pp. 62–89.

[150] W. F. Tinney and J. W. Walker, *Direct solutions of sparse network equations by optimally ordered triangular factorization*, Proceedings of the IEEE, 55 (1967), pp. 1801–1809.

[151] S. Tiwari and J. Kuhnert, *Finite pointset method based on the projection method for simulations of the incompressible Navier-Stokes equations*, Springer, 2003.

[152] S. Tiwari and J. Kuhnert, *Grid free method for solving Poisson equation*, Wavelet Analysis and Applications, (2004), pp. 151–166.

[153] S. Tiwari and J. Kuhnert, *Modeling of two-phase flows with surface tension by finite pointset method (FPM)*, Journal of Computational and Applied Mathematics, 203 (2007), pp. 376–386.

[154] S. Toledo, *Improving the memory-system performance of sparse-matrix vector multiplication*, IBM Journal of research and development, 41 (1997), pp. 711–725.

[155] F. X. Trias, O. Lehmkuhl, A. Oliva, C. D. Pérez-Segarra, and R. Verstappen, *Symmetry-preserving discretization of Navier–Stokes equations on collocated unstructured grids*, Journal of Computational Physics, 258 (2014), pp. 246–267.

[156] U. Trottenberg, C. W. Oosterlee, and A. Schüller, *Multigrid*, Academic Press, 2000.

[157] E. Uhlmann, R. Gerstenberger, M. Graf von der Schulenburg, J. Kuhnert, and A. Mattes, *The finite-pointset-method for the meshfree numerical simulation of chip formation*, in Proceedings of the 12th CIRP Conference on Modeling of Machining Operations, vol. 1, 2009, p. 145.

[158] E. Uhlmann, R. Gerstenberger, and J. Kuhnert, *Cutting simulation with the meshfree finite pointset method*, Procedia CIRP, 8 (2013), pp. 391–396.

[159] M. ur Rehman, C. Vuik, and G. Segal, *A comparison of preconditioners for incompressible Navier–Stokes solvers*, International Journal for Numerical methods in fluids, 57 (2008), pp. 1731–1751.

[160] H. Uzawa, *Iterative methods for concave programming*, Studies in linear and nonlinear programming, 6 (1958), pp. 154–165.

[161] H. A. Van der Vorst, *Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems*, SIAM Journal on scientific and Statistical Computing, 13 (1992), pp. 631–644.

[162] P. Vanek, M. Brezina, and J. Mandel, *Convergence of algebraic multigrid based on smoothed aggregation*, UCD/CCM Report, 126 (1998).

[163] P. Vanek, J. Mandel, and M. Brezina, *Algebraic multigrid on unstructured meshes*, UCD/CCM Report, 34 (1994), pp. 123–146.

[164] ——, *Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems*, Computing, 56 (1995), pp. 179–196.

[165] R. S. Varga, *Factorization and normalized iterative methods*, tech. rep., Westinghouse Electric Corp. Bettis Plant, Pittsburgh, 1959.

[166] A. E. Veldman and K.-W. Lam, *Symmetry-preserving upwind discretization of convection on non-uniform grids*, Applied Numerical Mathematics, 58 (2008), pp. 1881–1891.

[167] E. Wang, Q. Zhang, B. Shen, G. Zhang, X. Lu, Q. Wu, and Y. Wang, *Intel math kernel library*, in High-Performance Computing on the Intel® Xeon Phi, Springer, 2014, pp. 167–188.

[168] R. Webster, *Stabilisation of AMG solvers for saddle-point stokes problems*, International Journal for Numerical Methods in Fluids, 81 (2016), pp. 640–653.

[169] G. Wittum, *On the robustness of ILU smoothing*, SIAM journal on scientific and statistical computing, 10 (1989), pp. 699–717.

[170] U. M. Yang, *Parallel algebraic multigrid methods – high performance preconditioners*, in Numerical solution of partial differential equations on parallel computers, Springer, 2006, pp. 209–236.

[171] M. Yerry and M. Shepard, *Automatic three-dimensional mesh generation by the modified-octree technique*, International journal for numerical methods in engineering, 20 (1984), pp. 1965–1990.

[172] F. Zhang, *The Schur complement and its applications*, vol. 4, Springer Science & Business Media, 2006.