DELFT UNIVERSITY OF TECHNOLOGY

REPORT 05-02

Parallel Deflated CG Methods applied to Linear Systems from Moving Boundary Problems

J.M. Tang

ISSN 1389-6520

Reports of the Department of Applied Mathematical Analysis

Delft 2005

Abstract

In this report we give a short overview of aspects on parallel deflated conjugate gradient method which is applied on large, sparse, symmetric and semi-positive definite linear systems obtained from moving boundary problems. Moreover, we present some results of small numerical experiments.

After introducing the Navier-Stokes equations for multiphase flows, the pressure correction method is described shortly. In this method the Poisson equation dominates and leads to an ill-conditioned and huge linear sytem. This system is solved with the conjugate gradient method preconditioned with incomplete Cholesky or block-Jacobi. To accelerate the convergence of the iterative method a deflation technique is incorporated. We end with some aspects on implementing the whole technique in a parallel environment.

Keywords: Deflation, preconditioning, conjugate gradient method, Navier-Stokes equations, parallelism, domain decomposition methods.

Contents

List of Figures

List of Tables

Chapter

1 Introduction

Physical principles governing the flow of fluids and gases, such as water and air, have mathematically been understood since the times of Newton. This classical discipline is also known as fluid dynamics. Fluid flows encounter in everyday life include meteorological phenomena (rain, hurricanes, floods), environmental hazards (air pollution), processes in human body (blood flow, breathing) and so on.

Computational Fluid Dynamics (CFD) is a rather new discipline starting from the 1960's and which is in a state of rapid development. CFD gives an insight into flow patterns that are difficult, expensive or impossible to study using traditional (experimental) techniques. It provides a qualitative and sometimes even quantitative prediction of fluid flows or other phenomena in the fluid dynamics by means of mathematical modelling (partial differential equations), numerical methods (discretization and solution techniques) and software tools (solvers, pre- and postprocessing utilities). Numerical methods are required, since the most partial differential equations can not be solved analytically.

The Navier-Stokes equations describe many physical processes and, therefore, they are the main equations in the CFD. In Chapter 2 we will treat the Navier-Stokes equations in more detail. As a special application, multi-phase flows are considered where the phases are non-stationary resulting in moving boundary problems. We deal shortly with the varying interfaces which give the most difficulties in this kind of problems.

The so-called pressure-correction method is employed to find solutions of the Navier-Stokes equations. Most time-consuming part to solve inside this method is the Poisson equation. We apply the iterative conjugate gradient method (CG method) to solve the huge and ill-conditioned linear system which follows after discretization of the Poisson equation. However, the CG method performs very slowly, even after preconditioning the linear system with for instance the incomplete Cholesky decomposition (resulting in the ICCG method). Deflation techniques can be incorporated in the ICCG method to accelerate the convergence. This results in the DICCG method, which is the subject of Chapter 3.

Figure 1.1: An example of a two-phase flow: a droplet splash.

In this chapter different variants and aspects about deflation methods are considered in more detail.

In 3-dimensional multi-phase flows the linear system can be very large. This may result in a problem which can not be solved on a sequential computer, even when the DICCG method is applied. In this case, we have to use domain decomposition methods (DDM) and switch to a parallel environment. Parallelization of DICCG is not straightforward since the IC preconditioner has not been designed for this purpose. We apply other preconditioners like the so-called block-Jacobi preconditioner. In Chapter 4, this subject is treated in more detail.

Chapter

2 Moving Boundary Problems

2.1 Introduction

A moving boundary problem (MBP) is a non-linear initial or boundary value problem with a moving boundary whose position has to be determined as part of the solution. An MBP arises in several branches of applied mathematics. The difficulties inherent in these problems represent an analytical as well as a numerical challenge, because MBP's are always non-linear. Perhaps the oldest problem of this type was treated by Isaac Newton, in Book II of his great 'Principia Mathematica' of 1687, by considering the optimal nose-cone shape for the motion of a projectile subject to air resistance. Many other applications of MBP's are known, such as the ice-water front during the melting of ice, the filling front during the filling of a mold and the shoreline of a sedimentary basin.

Recently, moving boundary problems have received much attention due to their applicative relevance in diffusion and heat flow processes. Most of the previous studies are concerned with the linear heat equation, particularly in connection with the classical Stefan problem and its generalizations. In for instance Segal *et al.* [33] and Vermolen & Vuik [37], the dissolution of metallurgical particles occurring during the heat treatment of alloys are considered. The mathematical model of this dissolution process contains a description of the particle interface, of which the position varies in time. Such a model is called a Stefan problem. For a general two- or three-dimensional Stefan problem, it is impossible to obtain an analytical solution, so numerical schemes are required to solve this.

Other current research items deal with MBP's in fluid flows. Applications are bubbly or multi-phase flows, for instance the appearance of oil droplets or air bubbles in water, see for instance Van der Pijl *et al.* [30]. High density-ratio flows with complex interface topologies can occur in these flows. Between the phases a sharp front can exist, where density and viscosity change abruptly. This leads to MBP's which are numerically hard to solve.

In this chapter, we will focus on the mathematical model in modelling bubbly

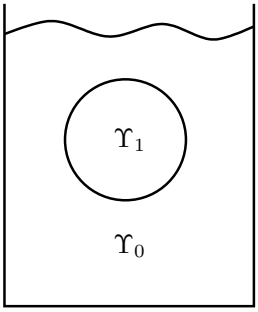


Figure 2.1: Geometry of the two-phase model with the fluids Υ_0 and Υ_1.

flows and it is mainly based on [30].

2.2 Mathematical model

For simplicity, we consider a two-phase flow with fluids Υ_0 and Υ_1 in domain $\Omega \in \mathbb{R}^2$, where Υ_1 corresponds with a bubble and Υ_0 with the environment of that bubble, see Figure 2.2. Generalizing to $\Omega \in \mathbb{R}^3$ and with more fluids/bubbles is straightforward.

Both fluids are assumed to be incompressible, so that the flow in each fluid is divergence-free. From the continuity equation we can derive

$$\nabla \cdot u = 0, \tag{2.1}$$

which holds inside Υ_0 or Υ_1. Moreover, $u = (u_1, u_2)^T$ is the velocity vector in (2.1).

The flow is governed by the well-known Navier-Stokes equations for incompressible flow:

$$\frac{\partial u}{\partial t} + u \cdot \nabla u + \frac{1}{\rho}\nabla p = \frac{1}{\rho}\nabla \cdot \mu \left(\nabla u + \nabla u^T\right) + g - f_{\sigma\kappa}, \tag{2.2}$$

with $x \in \Omega$ and $t > 0$. In (2.2), $g = (g_1, g_2)^T$ and $f_{\sigma\kappa} = (f_{\sigma\kappa 1}, f_{\sigma\kappa 2})^T$ are vectors of the gravity and surface tension force (which will also be defined later on) and ρ, p, μ are the density, pressure and viscosity, respectively. Furthermore, we assume that $\rho = \rho_0$ and $\mu = \mu_0$ hold in Υ_0 and $\rho = \rho_1$ and $\mu = \mu_1$ in Υ_1. This means that density ρ and viscosity μ are non-continuous but piecewise-continuous on Ω. [1]

When the horizontal and vertical direction are denoted by x_1 and x_2, respectively, we can rewrite (2.2) in components:

$$\frac{\partial u_i}{\partial t} + u_1\frac{\partial u_i}{\partial x_1} + u_2\frac{\partial u_i}{\partial x_2} + \frac{1}{\rho}\frac{\partial p}{\partial x_i} = \frac{1}{\rho}\vartheta_i + g_i - f_{\sigma\kappa i}, \tag{2.3}$$

with stress tensor ϑ defined by

$$\vartheta_i = (= \tau \cdot n) \equiv \frac{\partial}{\partial x_1}\mu\left[\frac{\partial u_i}{\partial x_1} + \frac{\partial u_1}{\partial x_i}\right] + \frac{\partial}{\partial x_2}\mu\left[\frac{\partial u_i}{\partial x_2} + \frac{\partial u_2}{\partial x_i}\right], \tag{2.4}$$

[1] Note that from Eqs. (2.1) and (2.2), we obtain $\nabla \cdot \mu\left(\nabla u + \nabla u^T\right) \neq \mu\,\Delta u$, since μ is non-constant (only piecewise constant) in Ω.

which holds for $i = 1, 2$.

2.3 Assumptions for Some Parameters

To implement the above mathematical model in a numerical model, some assumptions are made about the parameters acting in the Navier-Stokes equations (2.2).

Viscosity μ

Without loss of generality, the viscosity μ can be made continuously by smoothing (see Exp. (10) and (11) of [30]). Due to the smoothed viscosity μ, the velocity u and its derivatives are continuous, so that the gradients of u can be approximated by central differences in the numerical implementation. In addition, the interface conditions at the interface of Υ_0 and Υ_1, say S, are simplified. Considering μ and the continuity of mass and momentum, we can express the following interface conditions (cf. Section 2.1 of [30]):

$$\begin{cases} [u] &= 0, \\ [p] &= \sigma\kappa, \end{cases} \tag{2.5}$$

where the brackets denote jumps across the interface, σ is the surface tension coefficient and κ is the curvature of the interface.

Density ρ

Note that, in contrast to the viscosity, smoothing the density ρ does not preserve the conservation of mass anymore [30]. Therefore, ρ has to remain discontinuous in the model. This discontinuous density field can be dealt with similarly to the ghost fluid method, see Section 3.1.2 of [30]. We need the jump condition

$$\left[\frac{1}{\rho}\nabla p\right] = \underline{0} \tag{2.6}$$

to discretize the pressure derivative $\frac{1}{\rho}\nabla p$ at the left-hand-side of the Navier-Stokes equations (2.2) in the neigbourhood of the interface. For the first component we obtain:

$$\frac{1}{\rho}\frac{\partial p}{\partial x}\mid_{i+1} = \frac{1}{\overline{\rho}_{i+1}}\frac{p_{i+\frac{3}{2}} - p_{i+\frac{1}{2}}}{\Delta x}. \tag{2.7}$$

Parameter $\overline{\rho}$ is the weighted average defined by:

$$\overline{\rho} = (1-\theta)\rho_0 + \theta\rho_1, \tag{2.8}$$

where the relative fraction $\theta \in (0, 1)$ is defined according to Figure 2.3. [2]

[2] If $\theta = \frac{1}{2}$, then we obtain the original average:

$$\overline{\rho} = \frac{\rho_0 + \rho_1}{2}. \tag{2.9}$$

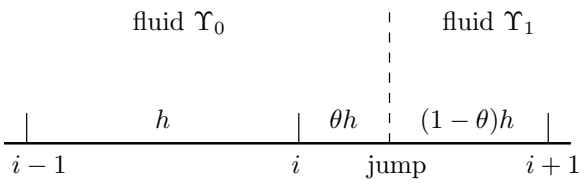


Figure 2.2: Geometry of the two-phase model with parameter θ.

Surface Tension Force $f_{\sigma\kappa}$

The parameter $f_{\sigma\kappa}$ has been incorporated in the Navier-Stokes equations (2.2), since the surface tension force is acting at the interface S. Due to the smoothed μ, the velocity components are also continuous which results in the interface conditions (2.5). To make the pressure also continuously, we smooth $f_{\sigma\kappa}$ by (cf. (20) of [30]):

$$f_{\sigma\kappa} \equiv \frac{2}{\rho_0 + \rho_1} \sigma\kappa\delta_\alpha(\Phi)\nabla\Phi, \tag{2.10}$$

where

$$\kappa = \nabla \cdot \frac{\nabla\Phi}{|\nabla\Phi|}, \tag{2.11}$$

which can be approximated with central differences. In (2.10) and (2.11), δ_α is the smoothed delta function and function Φ is defined as in the level-set method (see later on).

2.4 Pressure Correction Method

The Navier-Stokes equations (2.2) are discretized using an FD scheme with a uniform Cartesian grid. The unknown velocity components u_1 and u_2 are solved sequentially. Superscript n denotes time-level n. First a tentative velocity u_i^* is computed by the predictor step:

$$\frac{u_i^* - u_i^n}{\Delta t} = -u_1^n \left(\frac{\partial u_i}{\partial x_1}\right)^* - u_2^n \left(\frac{\partial u_i}{\partial x_2}\right)^* + \frac{1}{\rho}\left(\nu_i^* + \varrho_i^n\right), \tag{2.12}$$

with

$$\nu_i^* \equiv \left(\frac{\partial}{\partial x_1}\mu\frac{\partial u_i}{\partial x_1} + \frac{\partial}{\partial x_2}\mu\frac{\partial u_i}{\partial x_2}\right)^*, \quad \varrho_i^n \equiv \left(\frac{\partial}{\partial x_1}\mu\frac{\partial u_1}{\partial x_i} + \frac{\partial}{\partial x_2}\mu\frac{\partial u_2}{\partial x_i}\right)^n, \tag{2.13}$$

which has been splitted into a part on an implicit time level * and a part on an explicit time level n, due to the fact that u_1 and u_2 are solved sequentially where also Eq. (2.1) has to be incorporated. To relax the time step Δt in the numerical model, the whole stress tension ϑ_i can be made implicitly by replacing n by $*$ which results in a harder solvable scheme.

The resulting matrix-vector system of equations can be solved by a direct or iterative method. We can note that in general we obtain a 'favorable' matrix in the sense of consisting eigenvalues relatively close to one. This is the consequence of the appearance of Δt, which is very small in practice, on the left-hand-side of (2.12).

After the prediction step (2.12), the velocities at the new time instant $n+1$ can be computed by the correction step:

$$\frac{u_i^{n+1} - u_i^*}{\Delta t} = -\frac{1}{\rho}\nabla p + g_i - f_{\sigma\kappa i}, \quad i = 1, 2, \tag{2.14}$$

under the constraint of the continuity equation (2.1). This yields

$$\begin{cases} u_i^{n+1} &= u_i^* + \Delta t \left(-\frac{1}{\rho}\nabla p + g_i - f_{\sigma\kappa i} \right), \quad i = 1, 2, \\ \nabla \cdot u^{n+1} &= 0, \end{cases} \tag{2.15}$$

where the divergence and the gradient operators still have to be discretized. Combining both expressions of (2.15) leads to the pressure correction step, also known as the Poisson equation,

$$\nabla \cdot \left(\frac{1}{\rho}\nabla p \right) = f, \tag{2.16}$$

with $f \equiv \nabla \cdot \left(\frac{1}{\Delta t} u^* + g - f_{\sigma\kappa} \right)$. In contrast to the velocity u, the pressure is discontinuous due to the discontinuous density ρ. Therefore, discretizing and solving of (2.15) is not straightforward.

In summary, solving the Navier-Stokes equation (2.2) at each time step consists of sequentially carrying out the velocity prediction (2.12), pressure correction (2.16) and velocity correction (2.15) steps. As earlier mentioned, the pressure correction step (2.16) is by far the most difficult one since the pressure operator is the leading contributor to stiffness. In other words, it is the pressure correction step which is the most computationally challenging, despite its elliptic origins.

Further details on the pressure-correction method and its accuracy can be found in [30] and Van Kan [17].

2.5 Interface Advection

Recently, many methods are proposed to treat bubbly flows and to advect the moving interfaces. In general, the interface representation can be explicit ('moving, boundary conforming mesh') or implicit ('fixed mesh') or a combination of both.

Purely moving, boundary-conforming meshes are troublesome for simulating large numbers of arbitrarily shaped interfaces. Front-tracking methods are combinations of fixed and moving mesh methods. Although the interface is tracked by an interface grid, the flow is solved on a fixed grid. However, the interface grid will be difficult to evaluate when the interface has arbitrary shape and topology. Therefore an implicit interface definition by means of the Volume-of-Fluid (VOF) and Level-Set (LS) methodology is preferred in our applications. These methods have their drawbacks: advecting the so-called marker function in VOF is not straightforward whereas the LS is not mass-conserving. Van der Pijl *et al.* [30] have combined these two methods to obtain a mass-conserving Level-Set (MLS) method, where advecting the marker function is a lot simpler compared by the original VOF method. A short description of the MLS is given below, see [30] for details.

2.5.1 LS and VOF Method

In the original LS method, the interface is defined by the zero level-set of a marker function Φ: $\Phi = 0$ at the interface, $\Phi > 0$ inside fluid Υ_1 and $\Phi < 0$ elsewhere. The interface is implicitly advected by advecting Φ as if it were a material property:

$$\frac{\partial \Phi}{\partial t} + u \cdot \nabla \Phi = 0. \tag{2.17}$$

Although this results in an elegant method, the drawback is that it does not conserve mass. This means that additional effort is necessary to conserve mass or at least to improve mass conservation.

In the VOF method, a marker function Ψ indicates the fractional volume of a certain fluid Υ_1 in a computational cell τ:

$$\Psi = \frac{1}{\mathrm{vol}(\tau)} \int_\tau \chi \, \mathrm{d}\tau, \tag{2.18}$$

where χ is the characteristic function which is 1 in Υ_1 and 0 elsewhere. As a result, the value of Ψ will be 0 or 1 in cells without interface and $0 < \Psi < 1$ in cells containing the interface. There are algebraic and geometric methods available to advect Ψ. However, algebraic methods are straightforward but inaccurate, whereas geometric methods are elaborate to apply. Therefore advecting Ψ is rather complicated.

2.5.2 Mass-Conserving Level Set Method

In the MLS method, corrections to the LS function at time step n, say Φ^n, are made by considering the fractional volume Ψ of a certain fluid within a computational cell. First the usual LS advection is performed resulting in a tentative LS function $\Phi^{n+1,*}$. Since $\Phi^{n+1,*}$ will certainly not conserve mass, corrections are made such that it does conserve mass. This requires three steps at each time step:

1. the relative volume Ψ^n of a certain fluid in a computational cell is computed from an explicit formula f which depends on Φ^n and its gradient: Φ^n:
$$\Psi^n = f(\Phi^n, \nabla \Phi^n); \tag{2.19}$$
2. the VOF function Ψ^n has to be advected conservatively during a time step towards Ψ^{n+1}, using Φ^n;
3. corrections to $\Phi^{n+1,*}$ are sought such that
$$f(\Phi^{n+1}, \nabla \Phi^{n+1}) = \Psi^{n+1} \tag{2.20}$$
holds, resulting in a mass-conserving Φ^{n+1}.

The MLS method is described in detail in [30].

Chapter

3 CG Method, Preconditioning and Deflation Techniques

3.1 Introduction

Large and sparse linear systems occur in many scientific and engineering applications. These systems result often from a discretization of partial differential equations (PDE) where Finite Elements (FE), Finite Volumes (FV) or Finite Differences (FD) schemes are applied. The systems tend to become very large for 3-dimensional problems. Some models involve both time and space as independent parameters and therefore it is necessary to solve such a linear system efficiently at all time steps.

If we write a linear system as $Ax = b$, $A \in \mathbb{R}^{n \times n}$, which is used to discretize an elliptic PDE defined on Ω, then A appears regularly to be a symmetric and positive definite (SPD) coefficient matrix. In the previous chapter, we have defined the Poisson equation:

$$\nabla \cdot \left(\frac{1}{\rho} \nabla p \right) = f, \tag{3.1}$$

which is an elliptic PDE. Therefore, the resulting matrix A is SPD when appropriate boundary conditions are used. Presently, direct methods (such as a Cholesky decomposition) are available to solve such a linear system. However, fill-in causes a loss of efficiency for large and sparse matrix A. For such a case, iterative methods are a better alternative to reduce both memory requirements and computing time.

The most popular iterative method is the Conjugate Gradient (CG) method (see e.g. [14]). It is well-known that the convergence rate of the CG method is bounded as a function of the condition number of matrix A. After k iterations of the CG method, the error is bounded by (cf. Golub & Van Loan [14], Thm. 10.2.6):

$$||x - x_k||_A \leq 2||x - x_0||_A \left(\frac{\sqrt{\kappa - 1}}{\sqrt{\kappa + 1}} \right)^k, \tag{3.2}$$

where $\kappa = \kappa(A) = \lambda_n/\lambda_1$ is the spectral condition number of A and, moreover, the A-norm of x is given by $||x||_A = \sqrt{x^T Ax}$. Therefore, a smaller κ leads to a faster convergence of the CG method. Van der Sluis & Van der Vorst [34] noted that the convergence may be significantly faster than suggested in (3.2) if the eigenvalues λ_i of A are clustered.

In practice, it appears that the condition number κ is relatively large, so that solving $Ax = b$ applying the CG method shows slow convergence to the solution. Instead, a preconditioned system $M^{-1}Ax = M^{-1}b$ could be solved, where the SPD preconditioner M is chosen, such that $M^{-1}A$ has a more clustered spectrum or a smaller condition number than that of A. Furthermore, M must be chosen in such a way that the system $My = z$ for every vector z can be solved with less computational work than the original system $Ax = b$.

Probably the most general and effective SPD preconditioning strategy in common use is to take $M = LL^T$ which is an Incomplete Cholesky (IC) factorization of A, defined by Meijerink & Van der Vorst [22]. Since A is an SPD matrix, an IC decomposition always exists. We denote the preconditioned Conjugate Gradient method by PCG and the PCG with IC factorization by ICCG.

In simple practical applications, ICCG shows good convergence relative to other iterative methods (e.g., CG, Gauss-Seidel, SOR). However, it appears that ICCG still does not give satisfactory results in more complex models, for instance when the number of grid points becomes very large or when there are large jumps in the coefficients of the discretized PDE.

To remedy the bad convergence of ICCG in more complex models, (eigenvalue) deflation techniques are proposed, originally by Nicolaides [26]. This deflation technique has been exploited by several other authors, e.g., Mansfield [20, 21], Morgan [23], De Gersem & Hameyer [12], Kolotilina [18] and Waisman *et al.* [45, 46]. The deflation technique has also been exploited by Vuik *et al.* [11, 38, 42–44] where we will concentrate on in this report.

The purpose of deflation techniques is projecting very small eigenvalues of A near zero to exactly zero. These techniques might be successful due to Kaasschieter [16]. He notes that eigenvalues of a symmetric positive semi-definite (SPSD) matrix that are zero do not contribute to the convergence of the CG method.

In this chapter we first derive the projection matrix required for the deflation technique. Thereafter, some properties, choices of deflation vectors and the deflated preconditioned CG algorithm are presented. We end with some examples to illustrate the deflation technique.

3.2 Derivation Projection Matrix

To describe the deflation method in mathematical sense we need to define the so-called 'projection matrix' first. Therefore we follow the approach of Nicolaides [26].

Let the system to be solved be denoted by $Ax = b$, where A is $n \times n$ and SPD. For this system, denote the iterate residuals by $r_k = b - Ax_k$. In fact, the unpreconditioned CG algorithm is described by

$$\begin{aligned} x_0 &= 0; \\ x_{k+1} &= \mathcal{C}_k[x_k, x_{k-1}] + \mu_k r_k, \quad k = 0, 1, 2, \ldots, \end{aligned} \tag{3.3}$$

where the expression $\mathcal{C}_k[x_k, x_{k-1}]$ means a determined convex combination of x_k and x_{k-1}. Moreover, the parameter sequences μ_k and also the convex combinations $\mathcal{C}_k[x_k, x_{k-1}]$ are determined by requiring that r_{k+1} is orthogonal to both r_{k-1} and r_k. Clearly,

$$r_{k+1} = \mathcal{C}_k[r_k, r_{k-1}] - \mu_k A r_k, \quad k = 0, 1, 2, \ldots. \tag{3.4}$$

The above defined parameters μ_k and $\mathcal{C}_k[x_k, x_{k-1}]$ are determined as in the standard derivation of the CG method (see for instance Golub & Van Loan [14]) [1].

Next, we modify (3.3) by 'deflating' certain constituents from the residual r_{k+1}, i.e., by minimizing r_{k+1} in some sense. Let Z denote a fixed $n \times r$ matrix whose columns are a basis for an r-dimensional subspace of $\mathbb{R}^n$ and consider iterations:

$$\begin{aligned} x_0 &= 0; \\ x_{k+1} &= \mathcal{C}_k[x_k, x_{k-1}] + \mu_k(r_k - Z c_k), \quad k = 0, 1, 2, \ldots, \end{aligned} \tag{3.5}$$

with the still undefined vector sequences c_k. It follows that

$$r_{k+1} = \mathcal{C}_k[r_k, r_{k-1}] - \mu_k A(r_k - Z c_k), \quad k = 0, 1, 2, \ldots. \tag{3.6}$$

Now, we define the vector sequences c_k by choosing $\min_{c_k} ||r_k - Z c_k||_A$ for all k. This leads to minimizing

$$(A(r_k - Z c_k), r_k - Z c_k)\,, \tag{3.7}$$

which is equivalent with solving c_k from:

$$\frac{d}{dc_k}\left(r_k^T A r_k - 2c_k^T Z^T A r_k + c_k^T Z^T A Z c_k\right) = 0. \tag{3.8}$$

This yields

$$-2Z^T A r_k + 2Z^T A Z c_k = 0. \tag{3.9}$$

We can rewrite the latter expression in

$$E c_k = Z^T A r_k \;\rightarrow\; c_k = E^{-1} Z^T A r_k, \tag{3.10}$$

with $E \equiv Z^T A Z$. Now, Equation (3.6) becomes

$$r_{k+1} = \mathcal{C}_k[r_k, r_{k-1}] - \mu_k A(r_k - Z E^{-1} Z^T A r_k), \tag{3.11}$$

and, since the identity $AP^T = PA$ holds, it yields

$$r_{k+1} = \mathcal{C}_k[r_k, r_{k-1}] - \mu_k P A r_k, \tag{3.12}$$

with

$$P \equiv I - AZE^{-1}Z^T. \tag{3.13}$$

In the latter expression, P denoted the projection matrix required in the deflation method.

[1] By using the notation $\mathcal{C}_k[x_k, x_{k-1}] = \omega_k x_k + (1 - \omega_k) x_{k-1}$ with $0 \leq \omega_k \leq 1$ and $\omega_0 = 1$ and by setting $\mu_k = \alpha_k \omega_k$, we obtain $\alpha_k = (r_k, r_k)/(r_k, A r_k)$ and the recursion $w_k^{-1} = 1 - w_{k-1}^{-1} \left[\alpha_k (r_k, r_k)\right] / \left[\alpha_{k-1} (r_k, A r_k)\right]$ by direct calculations.

Thus, P is constructed in such a way that it minimizes r_k in each iterate. Notice that Z is arbitrarily chosen above. We interpret the columns of Z as being a basis for a subspace of certain slowly varying residual components. Expression (3.7) is used to 'deflate' such components form each residual. As earlier mentioned, the behavior of the residuals are related to the condition number of A, so that in fact we have to deflate some eigenvalues of A. This is called 'eigenvalue deflation', where we choose Z such that it approximates the eigenvectors of small eigenvalues in some sense.

Finally, we remark that it might be the case that the defined projection matrix P even minimizes $||r_{k+1}||_A$ where $r_{k+1} = \mathcal{C}_k[r_k, r_{k-1}] - \mu_k A(r_k - Zc_k)$, stronger than 'only' minimizing $||r_k - Zc_k||_A$, see Saad *et al.* [32].

3.3 Deflation Technique

We have seen that the projection matrix P is defined by

$$P = I - AZE^{-1}Z^T, \quad E = Z^T AZ, \quad Z \in \mathbb{R}^{n\times r}, \tag{3.14}$$

where I is the $n \times n$ identity matrix and where the column space of Z is the so-called 'deflation subspace', i.e., the space to be projected out of the residual. We assume both $r \ll n$ and Z having rank r, so that it guarantees E to be SPD and, moreover, E^{-1} exists and is relatively cheap to compute. Note that the inverse of E is never determined in practice. Instead, the system $Ey = z$ with arbitrary vector z is solved at low cost, for instance with a Cholesky decomposition.

Note further that projector P in (3.14) appears also in the multigrid setting. If Z is chosen to be a coarse-to-fine prolongation operator, then P is normally referred to as a coarse-grid correction operator. This is somewhat misleading since also the operator:

$$P_C = I - ZE^{-1}Z^T, \quad E = Z^T AZ, \quad Z \in \mathbb{R}^{n\times r}, \tag{3.15}$$

is often denoted by the coarse grid correction operator.

However, in other literature (see e.g. Giraud & Gratton [13]), two families are splitted depending on whether the extreme eigenvalues are moved exactly to one or are shift to close to one. The first class is referred to the deflation approach, while the latter is referred to the class of coarse grid preconditioners. In this case, it depends on the choice of Z whether we deal with deflation or coarse grid correction.

Recently, Waisman *et al.* [45,46] proposed the generalized global-basis (GBB) method. This GBB method is based on the global basis (GB) method which constructs an auxiliary coarse model from the largest eigenvalues of the iteration matrix. The GBB method projects these modes which would cause slow convergence to a coarse problem which is then used to eliminate these modes. This stabilizes and accelerates the iterative process, and yields rates of convergence similar to the application of the unaccelerated multigrid method applied to a positive definite system. It appears that this projection approach is identical to the original deflation method.

To avoid ambiguity in this report, we call P_C the coarse-grid correction operator and P the projection/deflation operator for arbitrary Z. An iterative

method using P is called a 'deflation method' rather than a coarse-grid correction or global-basis method. A comparison between P and P_C will be made in Section 3.6.

3.3.1 Properties of P

Using (3.14), a set of properties of P can be derived. These properties are summarized in Theorem 3.1 and their proofs can be found in for instance [38].

Theorem 3.1. *Let A be an $n \times n$ SPD matrix, let I be an $n \times n$ identity matrix and let P be the projection matrix as defined in (3.14). Then the following properties hold:*

1. $P^T Z = PAZ = Z^T P = 0$;
2. $AP^T = PA$;
3. $(I - P^T)Z = Z$;
4. $P^2 = P$;
5. *PA is SPSD and therefore PA is singular.*

If Z consists of arbitrary vectors, then

6. *PA has exactly r zero-eigenvalues.*

If Z consists of orthogonal eigenvectors, then even the following properties hold:

7. *for $j > r$ PA and A have the same eigenvalues λ_j;*
8. *for $j \leq r$ the corresponding eigenvalues λ_j of A are all zero for PA.*

If Z consists of orthonormal eigenvectors, then also the following property holds:

9. *$P = I - \sum_{i=1}^{r} z_i z_i^T$, where z_i are the columns of Z.*

The elimination of the small eigenvalues of A, i.e., projecting the small eigenvalues of A to zero, takes place by using projection matrix P. Therefore, we have to solve

$$PA\tilde{x} = Pb \tag{3.16}$$

instead of $Ax = b$. In this sense, P can be interpreted as a preconditioner matrix. However, PA is singular (Property 5 of Theorem 3.1), resulting in a solution $\tilde{x}$ which is not uniquely determined. This does not cause problems, since Kaasschieter [16] concludes that the singular system can be solved by the CG method as long as the deflated system (3.16) is consistent, i.e., as long as $Pb \in \mathrm{Col}(PA)$.

Theorem 3.2 (cf. [38], Thm. 5) can now be applied to find the unique solution of $Ax = b$ from $PA\tilde{x} = Pb$.

Theorem 3.2. *Let x be the unique solution of the SPD system $Ax = b$. Let $\tilde{x}$ be an arbitrary solution of the SPSD system $PA\tilde{x} = Pb$ which satisfies $Pb \in Col(PA)$. Then, x can be determined by $x = ZE^{-1}Z^T b + P^T \tilde{x}$.*

Since $ZE^{-1}Z^Tb$ can be immediately computed, we need only to determine $P^T\tilde{x}$ to construct the solution x. Therefore, we solve $\tilde{x}$ from (3.16) using the CG method and premultiply this solution with P^T and add it to $ZE^{-1}Z^Tb$.

Note that, due to Properties 6 and 7 of Theorem 3.1 and the observation (zero-eigenvalues do not contribute to the convergence of the CG method) of Kaasschieter [16], we obtain indeed a smaller condition number κ after deflation, i.e.,

$$\tilde{\kappa}(PA) = \frac{\lambda_n}{\lambda_{r+1}} \leq \frac{\lambda_n}{\lambda_1} = \kappa(A), \tag{3.17}$$

where $\tilde{\kappa}(PA)$ is called the *effective* condition number of PA. When there is no ambiguity, we omit the tilde and therefore, we write $\kappa(PA)$ instead of $\tilde{\kappa}(PA)$.

3.3.2 Bounds for $\kappa(PA)$

As mentioned above, a nice feature of the deflation method is that the convergence rate depends on the effective condition number $\kappa(PA) = \lambda_n/\lambda_{r+1}$. Nicolaides [26] proved the following bounds:

$$\lambda_n(PA) = \sup_{x \perp \mathrm{Col}(Z)} \frac{x^Tx}{x^TA^{-1}x}, \quad \lambda_n(PA) = \inf_{x \perp \mathrm{Col}(Z)} \frac{x^Tx}{x^TA^{-1}x}. \tag{3.18}$$

In fact, Z is a rectangular matrix constructed in such a way that the 'inverse' Rayleigh quotient $x^Tx/x^TA^{-1}x$ should not take extremely small/large values on the subspace orthogonal to the column space of Z. [2]

In Frank & Vuik [11], rather sharp bounds of a different flavor are introduced for $\kappa(PA)$, see Theorem 3.3.

Theorem 3.3. *Let A and P as defined above. Suppose there exists a splitting $A = C + R$ such that C and R are SPD with the null space of C equal to the span of Z, i.e., Null(C) = span{Z}. Then*

$$\lambda_i(C) \leq \lambda_i(PA) \leq \lambda_i(C) + \lambda_{\max}(PR). \tag{3.20}$$

Moreover, the effective condition number of PA is bounded by

$$\tilde{\kappa}(PA) \leq \lambda_n(A) + \lambda_{r+1}(C). \tag{3.21}$$

Proof. The proof can be found in Section 2 of [11]. □

The bounds given in Theorem 3.3 provide direction in choosing the deflation vectors such that it optimizes the convergence property. When we consider a discretized domain Ω divided in subdomains, we get the following result. If grid refinement is performed on Ω, keeping the grid resolution of each subdomain fixed, then the condition number is insensitive to the grid size. In this case, the convergence is governed by the 'worst' conditioned subdomain problem.

Other kind of bounds of the eigenvalues and the effective condition number of PA are given in Nabben & Vuik [24], see Theorem 3.4 and Corollary 3.1.

[2] In general, if H is symmetric and the vector $p \neq 0$, then the scalar $\frac{\langle p,Hp\rangle}{\langle p,p\rangle}$ is known as the Rayleigh quotient of p. The Rayleigh quotient is important because it has the following property:

$$\lambda_{min}(H) \leq \frac{\langle p, Hp\rangle}{\langle p, p\rangle} \leq \lambda_{max}(H). \tag{3.19}$$

See also Golub & Van Loan [14], Section 5.2.3.

Theorem 3.4. *Let A as defined above. Let $Z_1 \in \mathbb{R}^{n\times r}$ and $Z_2 \in \mathbb{R}^{n\times s}$ with rank $Z_1 = r$ and rank $Z_2 = s$. Define again $E_1 := Z_1^T AZ_1$ and $E_2 := Z_2^T AZ_2$. Define also: $P_1 := I - AZ_1E_1^{-1}Z_1^T$ and $P_2 := I - AZ_2E_2^{-1}Z_2^T$. If $Col(Z_1) \subseteq Col(Z_2)$ then:*

$$\lambda_n(P_1A) \geq \lambda_n(P_2A), \quad \lambda_{r+1}(P_1A) \leq \lambda_{s+1}(P_2A). \tag{3.22}$$

The latter theorem states that the effective condition number of PA decreases if we increase the number of deflation vectors, see also Corollary 3.1.

Corollary 3.1. *Let A, P_1, P_2 be as in Theorem 3.4. Then:*

$$\tilde{\kappa}(P_1A) \leq \tilde{\kappa}(P_2A). \tag{3.23}$$

3.3.3 Combined Preconditioning and Deflation Techniques

It is also possible to combine both a standard preconditioning and a deflation technique. Instead of solving $PA\tilde{x} = Pb$, we solve:

$$M^{-1}PA\tilde{x} = M^{-1}Pb. \tag{3.24}$$

If the IC decomposition is applied as preconditioner M, then this leads to the DICCG method which is described in the next section. This method is rather efficient relative to other general preconditioners. For instance, Theorem 2.11 and numerical experiments of [24] show that the DICCG method converges always faster than ICCG preconditioned by the coarse grid correction.

3.4 DICCG Algorithm

The linear system $Ax = b$ can be solved with DICCG as aforementioned. Then we solve actually the preconditioned-deflated linear system (3.24). Below the algorithm of is given in detail (cf. Algorithm 1 of [43]). Note that in the standard IC-decomposition the lower triangular matrix L and diagonal matrix D are used such that $A \approx LD^{-1}L^T$ is satisfied [3].

The algorithm is described in more detail in **Appendix A**. Note that if $Z = 0$, we obtain $P = I$ so that DICCG results in ICCG, i.e., the non-deflated ICCG method.

Moreover, notice that z_j in the above algorithm is the 'preconditioned' residual, i.e., $z_j = (LD^{-1}L^T)^{-1}r_j$. This gives no ambiguity in the context, because the columns of Z $(= z_1, z_2, \ldots, z_r)$ are not mentioned explicitly in the algorithm.

In literature, more variants of DICCG are known, see for instance Saad *et al.* [32] and Kolotilina [18]. These variants differ slightly from the above algorithm but they are mathematically equivalent.

[3] More detailed, lower triangular matrix L as given in the algorithm is determined with the IC decomposition such that it satisfies (cf. [22]):

- $L_{ij} = 0$ when $A_{ij} = 0$;
- $(LD^{-1}L^T)_{ij} = A_{ij}$ when $A_{ij} \neq 0$ and $L_{ii} = D_{ii}$.

Algorithm 1 DICCG Algorithm

Compute $\hat{r}_0 := Pb$, $z_1 := L^{-T}DL^{-1}\hat{r}_0$ and $p_1 := z_1$.
for $j := 1, \ldots,$ until convergence **do**
  $w_j := PAp_j$
  $\alpha_j := \frac{(\hat{r}_j, z_j)}{(p_j, w_j)}$
  $\tilde{x}_{j+1} := \tilde{x}_j + \alpha_j p_j$
  $\hat{r}_{j+1} := \hat{r}_j - \alpha_j w_j$
  $z_{j+1} := L^{-T}DL^{-1}\hat{r}_{j+1}$
  $\beta_j := \frac{(\hat{r}_j, z_j)}{(\hat{r}_{j-1}, z_{j-1})}$
  $p_{j+1} := z_{j+1} + \beta_j p_j$
end for

3.5 Choice of the Deflation Space

The deflation technique and the DICCG method are presented in the previous section. Recall that $P = I - AZE^{-1}Z^T$ where Z is an $n \times r$ matrix as defined in (3.14). Now, the remaining part left is to define the deflation space, i.e., the columns of Z.

3.5.1 Eigenvalue Deflation

We have earlier mentioned that deflation is applied to get rid of (extremely) small eigenvalues that delay the convergence of the iterative method. Due to Properties 6 and 7 of Theorem 3.1, the most natural choice of the columns of Z is exactly the eigenvectors corresponding to those small eigenvalues of A. We denote this choice of deflation by *eigenvalue deflation* (EV-Def). However, this is not applicable in practice because in general

- the involved eigenvectors are unknown. Therefore, these eigenvectors have to be approximated, which can be very expensive;
- Z is not sparse, leading to large work to construct E and computing with P in the DICCG method.

Hence, we have to look for other choices of Z instead of EV-Def. Generally, favorable requirements for the choice of Z are

- $r \ll n$;
- columns of Z can be constructed relatively easy;
- Z is sparse;
- the columns of Z represent approximately the eigenvectors of the small eigenvalues so that Properties 6 and 7 of Theorem 3.1 still hold 'in some sense'.

Many authors have proposed methods to approximate the eigenvectors, see e.g. Vuik *et al.* [43], Chapman & Saad [5], Burrage *et al.* [3] and Saad *et al.* [32].

Vuik *et al.* [43] proposed a scheme based on physical deflation (P-Def) in which the deflation vectors are continuous and satisfy the original partial differential equation on a subdomain. In this case, Z is more sparse than Z obtained with EV-Def.

Chapman & Saad [5] found approximate eigenvectors in the deflated-GMRES algorithm from the data generated during the GMRES iterations. Three projection techniques have been presented to obtain such approximations. The harmonic projection method (H-Def), originally suggested by Morgan [23], yielded the best results in finding eigenvalues nearest zero. The H-Def approach uses Ritz values and relies on solving generalized eigenvalue problems with a much lower dimensions than A. Note that this approach is successful to solve efficiently SPD systems with multiple right-hand sides of the form $Ax^{(s)} = b^s,\ s = 1, 2, \ldots, \nu$, see Section 5 of Saad *et al.* [32] for details. However, the deflation matrix obtained with H-Def is not sparse in general resulting in possible heavy computations with P and huge memory requirements.

Burrage *et al.* [3] have used a deflation technique applied on standard iterative methods (like Gauss-Seidel or Jacobi methods) based on

$$My^{(k+1)} = Ny^{(k)} + b, \tag{3.25}$$

with $A = M - N$ where M is non-singular. The deflation technique relies on computing so-called orthogonalized difference vectors and determining Schur vectors of a matrix with lower dimensions. This deflation technique provides a distinct advantage for ill-conditioned systems where the underlying scheme would either diverge or converge very slowly. Several numerical experiments demonstrated the efficiency of the method in [3]. However, for systems where the iterative scheme is already converging reasonably well, the accelerated convergence provided by deflation is not worth by considering the required extra work (see Section 6 of [3]).

In the next subsection we introduce another class of deflation techniques which is based on choosing subdomains rather than approximating eigenvectors.

3.5.2 Subdomain Deflation

A different variant involves the so-called algebraic deflation (A-Def) with discontinuous deflation vectors, see for instance Vuik *et al.* [38, 42] and Nicolaides [26]. In fact, A-Def is a domain decomposition method and therefore it is often called *subdomain deflation*.

It appears that A-Def is favorable relative to P-Def and H-Def due to the following observations:

- Z of A-Def is sparser than Z defined in P-Def and especially in H-Def;
- in numerical experiments, it has been shown that A-Def speeded the convergence up compared to P-Def.

A-Def is constructed in the following more mathematical sense. Let the computational domain denoted by Ω. This can be divided in open subdomains $\Omega_j,\ j = 1, 2, \ldots, r$, such that $\Omega = \cup_{j=1}^{r} \bar{\Omega}_j$ where $\bar{\Omega}_j$ is Ω_j including its boundaries, i.e., $\bar{\Omega}_j = \Omega_j \cup \Gamma$. Let $z_j,\ j = 1, 2, \ldots, r$ be the columns of Z. For each

subdomain Ω_j we introduce a deflation vector z_j as follows:

$$z_j(x,y) = \begin{cases} 0, & (x,y) \in \Omega \setminus \bar{\Omega}_j; \\ 1, & (x,y) \in \Omega_j. \end{cases} \tag{3.26}$$

Notice that we have not judged the value of $z_j(x,y)$ in any possible interface points inside Ω. A graphically representation of the deflation vectors can be found in Figure 3.1.

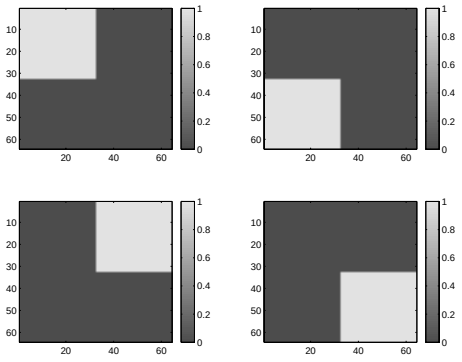


Figure 3.1: Representation of the subdomain deflation vectors with $r = 4$ in a square domain Ω.

Detailed treatment of interface points for Ω consisting high and low permeability regions has been done in [38]. In that case the A-Def technique appears to succeed when averaged or no overlap is applied in the deflation vectors. The worst variant is to choose total overlap deflation vectors.

Constant and Linear Deflation Vectors

When we consider a 1-dimensional case with two subdomains where n grid points are taken per subdomain, we obtain:

$$Z_{CD} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ \vdots & \vdots \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ \vdots & \vdots \\ 0 & 1 \end{bmatrix}. \tag{3.27}$$

Therefore, the deflation vectors in the A-Def method are also called constant deflation (CD) vectors. In this case, we add the underscript 'CD' to Z. Note that each column of Z can be scaled since the matrix PA only depends on span$\{Z\}$.

The underlying idea of choosing CD vectors was to approximate eigenvectors belonging to the smallest eigenvalues. Since the eigenvectors represent normally components of the solution which is often not-linear, these CD vectors give only a

rough approximation. This motivates several authors, among others Verkaik [36] and Fischer [10], to augment the subspace spanned by the CD vectors with linear deflation (LD) or even quadratic deflation (QD) vectors. In our 1-dimensional example with n grid points a subdomain, we obtain (cf. Eq. (3.28)):

$$Z_{LD} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & n & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 2 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 1 & n \end{bmatrix}, \tag{3.28}$$

for Z constructed with LD vectors. Hence, we have two deflation vectors per subdomain. Note that the values of the linear deflation vectors does not matter as long as the vector is linear, since only span$\{Z\}$ counts.

Generalization to 2-D and 3-D cases is straightforward for the CD vectors. For each subdomain, the elements of the vectors can simply be taken constant in the grid nodes. However, in contrast to these CD vectors, generalization to 2-D and 3-D cases of LD vectors is not trivial. In two dimensions one can take three vectors per subdomain: one constant and two linear vectors in each of the two dimensions. In similar way one needs four vectors per subdomain in three dimensions. When the number of subdomains are large, this can lead to proportionally many LD vectors making the deflation technique less cheap. Nevertheless, using LD or QD vectors can accelerate significantly the convergence of the iterative method, especially when the solution has a linear or quadratic form. This is left for further research.

3.5.3 Solution Deflation

Another class of deflation has been proposed by Clemens *et al.* [6]. They introduced a subspace projection extrapolation scheme for the starting vector generation of linear systems from implicit time integration schemes for electromagnetic discrete field formulations. The scheme yields optimal linear combinations from multiple available starting vectors. Similar to eigenvalue deflation, spectral components of the exact solution contained therein are optimally resolved which reduces the effective condition number.

Suppose $\{u_1, u_2, \ldots, u_{k-1}\}$ are solutions of the linear systems at time steps $1, 2, \ldots, k-1$. Then, to find solution u_k we use the deflation matrix:

$$Z = [u_1 \; u_2 \; \cdots \; u_{k-1}]. \tag{3.29}$$

Note that Z is dense in general. Therefore, k should be sufficiently small to preserve an efficient deflation method. This method is successful when

$$u_k \approx \text{span}\{Z\}. \tag{3.30}$$

Numerical experiments in [6] emphasized the improved convergence of the CG method combined with the solution deflation.

3.6 Comparison of Deflation and Some Other Preconditioners

Comparisons have been made between deflation and the well-known coarse grid correction (CGC) and the balancing Neumann-Neumann (BNN) preconditioner in Nabben & Vuik [24, 25], respectively. The results are summed up below, where we denote the deflation projection matrix by P_D instead of P to avoid ambiguity.

3.6.1 Deflation versus CGC preconditioner

The CGC preconditioner is defined by

$$P_C \equiv I - ZE^{-1}Z^T, \tag{3.31}$$

and in preconditioned case:

$$P_{CM^{-1}} \equiv M^{-1} - ZE^{-1}Z^T. \tag{3.32}$$

which have originally been introduced by Bramble *et al.* [2] and Dryja & Widlund [8]. They have showed, under mild conditions, that the convergence rate of the PCG method is independent of the grid sizes.

A more abstract analysis and further extension of this preconditioner have been given recently by Padiy *et al.* [28]. They note that instead of deflating the small eigenvalues as in original deflation methods, P_C 'moves' the small eigenvalues to the vicinity of the largest eigenvalue and using a preconditioner, which results in $P_{CM^{-1}}$, bounds the largest eigenvalues. One of the main advantages of the proposed algorithm is the possibility of avoiding exactly solving systems with E. This relaxes the restrictions posed on the choice of Z and often leads to more efficient implementations of the solver.

In the multigrid (MG) or domain decomposition (DDM) language, the matrices Z and Z^T are known as restriction and prolongation/interpolation operator, respectively. Moreover, the matrix $E \equiv Z^TAZ$ is called the Galerkin operator. In DDM, M^{-1} is the sum of the local solves in each domain, obtained by a direct or iterative way. To speed up convergence, a coarse grid correction $ZE^{-1}Z^T$ is added.

It can be proved that the effective condition number of the deflated preconditioned system $M^{-1}P_DA$ is always below the condition number of $P_{CM^{-1}}A$, i.e., the system preconditioned by CGC, see also Theorem 3.5.

Theorem 3.5. *Let $A, M \in \mathbb{R}^{n\times n}$ be SPD matrices. Let $Z \in \mathbb{R}^{n\times r}$ with rank $Z = r$. Then:*

$$\begin{aligned} \lambda_n(M^{-1}P_DA) &\leq \lambda_n(P_{CM^{-1}}A), \\ \lambda_{r+1}(M^{-1}P_DA) &\geq \lambda_1(P_{CM^{-1}}A). \end{aligned} \tag{3.33}$$

Proof. See the proof of Theorem 2.11 of Nabben & Vuik [24]. □

As a consequence, for all matrices $Z \in \mathbb{R}^{n\times r}$ and for all positive definite preconditioners M^{-1}, the CG method applied to the deflated preconditioned

system converges always faster than the CG method applied to the CGC preconditioned system, since

$$\kappa(P_{CM^{-1}}A) \geq \tilde{\kappa}(M^{-1}P_DA). \tag{3.34}$$

Numerical results for porous media flows and parallel preconditioners emphasized the theoretical results (see Section 4 of [24]). Note that in the above analysis we assume that systems with E are solved exactly since this is a requirement for the deflation method. In the CGC preconditioner the inner iterates with respect to E can be solved inaccurately. In this case, the iterative method needs more iterations to converge in general, but the required CPU time can be significantly lowered since the inner iterates can be computed cheap.

3.6.2 Deflation versus BNN preconditioner

The BNN preconditioner is given by

$$P_B \equiv (I - ZE^{-1}Z^TA)M^{-1}(I - AZE^{-1}Z^T) + ZE^{-1}Z^T, \tag{3.35}$$

where M is an SPD so-called Neumann-Neumann preconditioner and therefore P_B is also SPD. This preconditioner is originally proposed by Mandel [19].

The effective condition number of $M^{-1}P_DA$ is always below the condition number of $P_{CM^{-1}}A$, see Theorem 3.6.

Theorem 3.6. *Let $A, M \in \mathbb{R}^{n\times n}$ be SPD matrices. Let $Z \in \mathbb{R}^{n\times r}$ with rank $Z = r$. Then:*

$$\begin{array}{rcl} \lambda_n(M^{-1}P_DA) & \leq & \lambda_n(P_{B^{-1}}A), \\ \lambda_{r+1}(M^{-1}P_DA) & \geq & \lambda_1(P_{B^{-1}}A). \end{array} \tag{3.36}$$

Proof. See the proof of Theorem 2.7 of Nabben & Vuik [25]. □

As a consequence,

$$\kappa(P_BA) \geq \tilde{\kappa}(M^{-1}P_DA), \tag{3.37}$$

so the convergence bound based on the effective condition number implies that preconditioned deflated CG converges faster than CG preconditioned by the BNN preconditioner. However, the differences between both approaches are relatively small considering numerical experiments in [25], which can be explained by Theorem 3.7.

Theorem 3.7. *Suppose that the spectrum σ of $M^{-1}P_DA$ is given by:*

$$\sigma(M^{-1}P_DA) = \{0, \ldots, 0, \lambda_{r+1}, \ldots, \lambda_n\}, \tag{3.38}$$

then

$$\sigma(P_BA) = \{1, \ldots, 1, \lambda_{r+1}, \ldots, \lambda_n\}. \tag{3.39}$$

Proof. See the proof of Theorem 2.8 of Nabben & Vuik [25]. □

Thus both preconditioners lead to almost the same spectra with the same clustering: the zero eigenvalues of the deflation preconditioned system are replaced by eigenvalues which are one if the BNN preconditioner is used.

3.7 Examples

Various examples of A-Def can be found in [11, 38, 42–44]. Most of them are considering 2-dimensional geological applications with porous media, where the earth underground consists of layers with very large differences in permeability. The prediction of the fluid pressure history of this underground can be modelled by a time-dependent diffusion equation, leading to an SPD matrix A after discretization. Due to the large permeability differences, A has a very large condition number and the IC preconditioned matrix $L^{-1}AL^{-T}$ has a few very small eigenvalues near zero (cf. Thm. 2.2 of [44]). Several experiments have successfully been done with P-Def and A-Def for this type of problems. In most cases, A-Def gives the best results.

In this section we give two extra examples to illustrate the technique of deflation. In Example 1, we consider a 1-dimensional variant of the above mentioned geological application. In Example 2, a 2-dimensional Poisson equation on a homogeneous domain Ω is treated, where A-Def is applied to artificially layers and blocks in Ω.

Example 1: 1-D Layer Problem

The following diffusion problem with 9 layers is considered:

$$\begin{cases} -\nabla \cdot (\sigma(x)\nabla u(x)) = f(x), \quad x \in (0,1), \\ \\ u_x(0) = 0, \ u(1) = 1. \end{cases} \tag{3.40}$$

We denote the region with sand and shale by Ω_{sand} and Ω_{shale}, respectively. Now, we take $f(x) = 1$ and σ such that

$$\sigma(x) = \begin{cases} 1 & x \in \Omega_{sand}, \\ \epsilon & x \in \Omega_{shale}, \end{cases} \tag{3.41}$$

for some constant ϵ satisfying $0 < \epsilon \ll 1$.

Note that when $f(x) = 1$ is taken, then it can easily be verified that the solution is equal to the constant function $u(x) = 1$ for all ϵ.

By central differences, we obtain the following linear system:

$$Au = f, \tag{3.42}$$

where A is a symmetric positive definite (SPD) matrix with sizes $n \times n$, u is the unknown vector of size n and f is the source vector and some elements due to the boundary conditions.

The linear system (3.42) has been solved with the diagonal-preconditioned CG (DCG) and the deflated diagonal-preconditioned CG (DDCG) method with tolerance $\epsilon = 10^{-6}$. Note that in a 1-dimensional problem, the Incomplete Cholesky decomposition is identical to the standard Cholesky decomposition. Therefore, it makes no sense to apply this as preconditioner. Instead of this, we use the standard diagonal preconditioner D defined by

$$D = \text{diag}(A) \tag{3.43}$$

and hence,

$$\widetilde{A} = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}. \tag{3.44}$$

In fact, we take $L = D^{\frac{1}{2}}$ in Algorithm 1. Furthermore, eigenvectors of $\tilde{A}$ are chosen as deflation vectors.

We take 10 grid points in each layer, resulting in $n = 90$. Solution u of this problem can be seen in Figure 3.2.

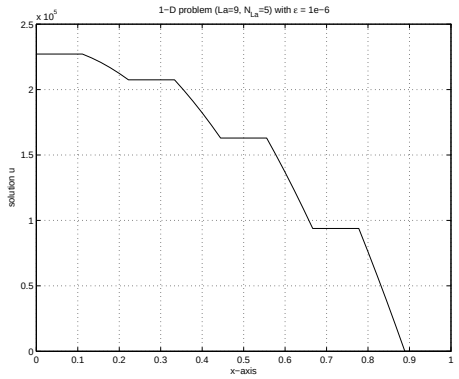


Figure 3.2: Solution of Example 1 ($\epsilon = 10^{-6}$) with 9 layers and 10 gridpoints per layer.

The results of using CG, DCG and DDCCG can be found in Table 3.1.

Method	# Iterations
CG	516
DCG	107
DDCG ($k = 1$)	94
DDCG ($k = 2$)	73
DDCG ($k = 3$)	44
DDCG ($k = 4$)	25
DDCG ($k = 5$)	21
DDCG ($k = 6$)	21

Table 3.1: Number of iterations for CG, DCG and DDCG with different values of k.

First we note that the number of iterations of CG (516) is very large with respect to the number of grid points (91). Moreover, it can be observed in Table 3.1 that DDCG with sufficient large k is more than twice as fast as DCG. In Figure 3.3, one can find the four eigenvectors which are used in DDCG with $k = 4$. It can be seen that the eigenvectors are constant in the sand layers.

Considering Table 3.1, we can also note that increasing k leads to less iterations and from $k = 5$ the decrease of iterations is relatively restricted. The explanation is given in [38].

In Figure 3.4, one can find the residuals (inside the iterative method) of each iterate for CG, DCG and DDCG. It can be concluded that the erratic behavior, observed in CG and DCG, has been disappeared in DDCG.

Example 2: Poisson Problem

The 2-dimensional Poisson equation is considered with constant density ρ:

$$\Delta u = 1, \tag{3.45}$$

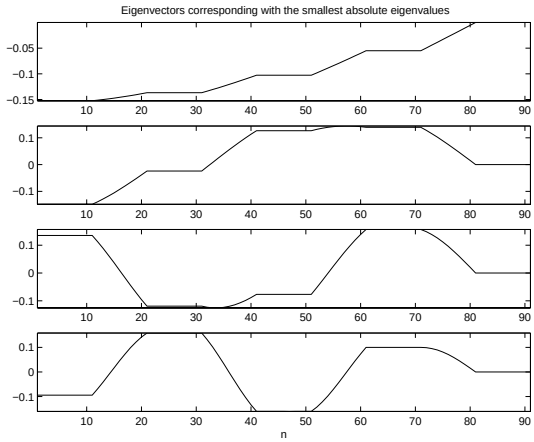


Figure 3.3: Eigenvectors corresponding by the four smallest eigenvalues of $D^{-1}A$.

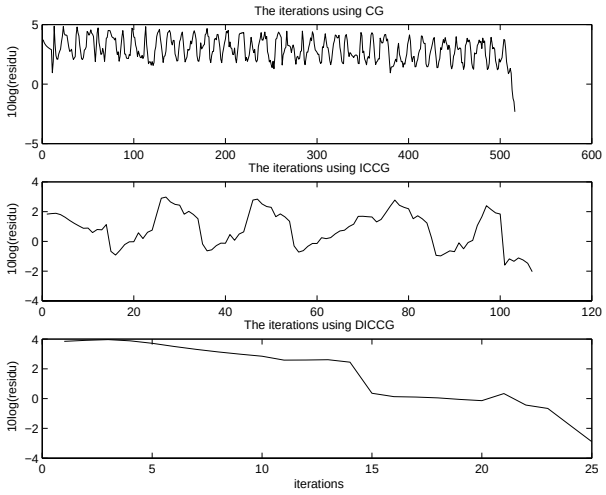


Figure 3.4: Residuals (r_i of the iterative method) during the iterates of CG, DCG and DDCG.

We solve this problem with the grid sizes $n_x = n_y = 64$ by applying a standard finite difference stencil and DICCG. The plot of the solution can be found in Figure 3.7.

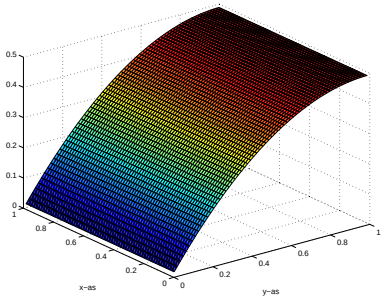


Figure 3.5: Solution of Example 2.

In DICCG, we apply A-Def where artificially horizontal layers and thereafter blocks with equal sizes are taken as subdomains Ω_j.

Deflation with Horizontal Layers as Subdomain Vectors

We start with a deflation technique where we take horizontal layers as subdomains. The results can be found in Table 3.7. We observe some improvements in the results by enlarging the number of layers, but these results are not satisfactory.

# Deflation Vectors	# Iterations
0	85
2	65
4	61
16	60
64	60

Table 3.2: Number of DICCG-iterations using 'horizontal layers deflation' in Example 2.

Deflation with Blocks as Subdomain Vectors

The results of deflation with blocks as subdomain vectors are given in Table 3.3. From this table, we notice a relatively large improvement by enlarging the number of deflation blocks. The number of iterations is reduced by over 80% comparing the first and last row, while the amount of work to construct P is still sufficiently small. However, the efficiency of block deflation is not only determined by the number of iterations. To illustrate this we look also at the number of flops needed in the method. In **Appendix B**, an estimate is given of the number of flops required in both methods. The total number of flops using ICCG and DICCG are denoted by F_{iccg} and F_{diccg}, respectively. The value ϕ is defined by $\phi = F_{diccg} - F_{iccg}$.

# Block Deflation	# Iterations	F_{iccg} ($\times 10^3$)	F_{diccg} ($\times 10^3$)	ϕ ($\times 10^3$)
No	85	198	198	–
1	77	198	201	3
4	58	198	153	−45
16	38	198	107	−91
64	25	198	103	−95
256	17	198	298	100

Table 3.3: Number of DICCG-iterations and ϕ using 'block deflation' in the Poisson problem at unit domain with Neumann and Dirichlet boundary conditions.

Obviously, it can be concluded that DICCG performs (much) better than ICCG by applying $r = 4, 16, 64$. For the case of $r = 64$, DICCG seems to be optimal in this example.

Chapter

4 Parallel Computing and Domain Decomposition Methods

4.1 Introduction

In Chapter 1 we have considered the Poisson equation. After discretization this has led to a linear system which can be huge, especially in the 3-D case. The required work to solve this with an iterative solver can be enormous and the available memory can be insufficient, even when a supercomputer is used.

Parallel computing is fast becoming an inexpensive alternative to the standard supercomputer approach for solving large linear problems that arise from PDE's like the Poisson equation. Much recent literature with wide overviews is available, see for instance Demmel *et al.* [7], Duff & Van der Vorst [9] and Petersen & Arbenz [29].

The iterative techniques can be parallelized by dividing the work over a set of computers in a certain way. Several forms of parallelism are available (see e.g. Saad [31], Chapter 11). A well-known implementation of parallelism is that shared memory computers are implemented with a switching network, see Figure 4.1. In this setup, no 'master-computer' is required controlling the 'slave-computers'. Main benefits of shared memory models are that access to data depends very little on its location in memory and that a large global memory is readily accessible to any processor.

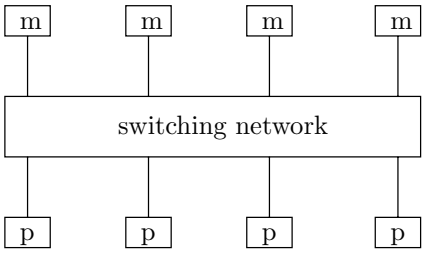


Figure 4.1: Parallel implementation with a switch based shared memory system. 'p' = processor, 'm' = memory.

The DICCG method has been introduced in the previous chapter to solve problems like the Poisson problem. To do this in a parallel environment, we have to distinguish the main types of operations in DICCG, which are:

(a) matrix-vector multiplications;

(b) vector updates;

(c) dot products;

(d) preconditioning setup and operations;

(e) deflation setup and operations;

In the above list, the potential bottlenecks are setting up the preconditioner and solving linear systems with the preconditioner (step (d)). On the other hand, deflation setup and operations (step (e)) can be parallelized well, which can be seen in the next section. Moreover, we note that the dot product operation (c) may be troublesome in computational applications, since all the processors must synchronize and perform communication before computations can be continued, see Section 6 of Hagger [15].

Since step (d) is the most complicated one in the parallel approach, we have to pay attention to this. In Section 4.3 we consider preconditioners based on the Schur complement and based on non-overlapping additive Schwarz (also known as the additive Schwarz with minimum overlap), which have attracted a lot of attention in the numerical analysis community. Equivalently, instead of DICCG we consider the deflated CG method preconditioned with a block-Jacobi preconditioner:

$$M_{jac} = \begin{pmatrix} M_{11} & & & \varnothing \\ & M_{22} & & \\ & & \ddots & \\ \varnothing & & & M_{ss} \end{pmatrix}. \tag{4.1}$$

In Section 4.4 we treat the preconditioner in more detail. Practically, solving the linear system $M_{jac}z = y$ accurately is algebraically equivalent to the Schur complement approach. However, each submatrix M_{ii} can be extremely large, so that accurately solving can be expensive. Instead, solving $M_{jac}z = y$ inaccurately, by applying an iterative method, becomes attractive. This latter approach is algebraically equivalent to the additive Schwarz preconditioner. Since there is no overlap, the linear subsystems $M_{ii}z_i = y_i$ for all $i = 1, 2, \ldots, s$ can be solved in parallel.

In fact, these parallel ideas are all based on domain decomposition methods (DDM), see also Smith *et al.* [35] and Wilders *et al.* [47] for a wide overview of these methods.

4.2 Parallel Deflation

The $n \times n$ projection matrix as given in the previous chapter has been defined as:

$$P = I - AZE^{-1}Z^T, \quad E = Z^TAZ, \quad Z \in \mathbb{R}^{n\times r}, \tag{4.2}$$

where we take r the number of subdomains (i.e., $r = s$). Recall that $\Omega_m, m = 1, 2, \ldots, s$ are non-overlapping subdomains such that $\cup_{m=1}^{s} \overline{\Omega}_m = \Omega$.

As in the coarse grid correction methods, the columns z_m of Z can be chosen in the following way:

$$z_m(i) = \begin{cases} 1, & i \in \Omega_m; \\ 0, & i \notin \Omega_m, \end{cases} \tag{4.3}$$

for all $m = 1, 2, \ldots, s$. Consequently, Z is a sparse matrix where each row consists of one non-zero element.

In parallel, first the unknowns have to be distributed according to subdomain across available processors. For convenience, one subdomain per processor is assumed. The coupling with neighboring domains is realized by the use of virtual cells added to the local grids. In this way, a block-row of $Ax = b$ corresponding to the subdomain ordering

$$A = \begin{pmatrix} A_{11} & \cdots & A_{1s} \\ \vdots & & \vdots \\ A_{s1} & \cdots & A_{ss} \end{pmatrix} \tag{4.4}$$

can be represented locally on one processor: the diagonal block A_{ii} represents coupling between local unknowns of subdomain i and the off-diagonal blocks of block-row i represent coupling between local unknowns and the virtual cells.

For the operations with deflation in parallel (see also Vuik *et al.* [41] and Frank & Vuik [11]), first we compute and store successively the matrices E and E^{-1} on each processor, whereas locally AZ is computed and stored. Use of the projection matrix P within a Krylov subspace method involves premultiplying a vector p by PA:

$$x \equiv PAp = (I - AZE^{-1}Z^T)Ap. \tag{4.5}$$

Then, to compute $x = PAp$, the following operations have to be done:

- the matrix-vector multiplication $w := Ap$, requiring nearest neighbor communications;
- the local contribution to the restriction $x_1 := Z^T w$ which has to be distributed to all processors;
- a coarse grid operation $x_2 := E^{-1}x_1$, which is locally determined;
- finally $x := I - AZx_2$ which is also locally determined.

The total communications involved in the matrix-vector multiplication and deflation are a nearest neighbor communication of the length of the interface.

From a 2-D problem (with a five-point discretization), it can be verified that the added iteration expense of deflation to be less expensive than an ILU(0) factorization and that the method will parallelize efficiently on a distributed memory computer (see Frank & Vuik [11], Section 5). Moreover, the overlapping of subdomains makes the parallel iterative method more or less independent of the subdomain grid size, but overlapping is not easy to implement on top of an existing software package. To make the parallel iterative method robust, one can also apply the deflation technique, since only a slow increase of the number of iterations can be observed when the subdomain grid size is constant and the number of subdomains increases (see also Vuik & Frank [39]). Additionally,

for a fixed global grid, it appears that the number of iterations decreases when the number of processors increases. This result has also been obtained in the numerical experiments of the previous chapter.

4.3 Domain Decomposition Methods

Before we treat preconditioners based on DDM's, first some of these DDM's are described shortly in this section to gain insight. Domain decomposition techniques are distinguished by four features (Saad [31], Chapter 13):

- type of partitioning;
- degree and way of overlap;
- processing of interface values;
- accuracy of subdomain solution.

In this report we consider only two DDM approaches:

1. Schur complement methods (SCM);
2. additive Schwarz methods (ASM).

These methods are related to each other, since they algebraically leads to the (accurate and inaccurate, respectively) block-Jacobi preconditioner.

4.3.1 Schur Complement Method (SCM)

Let $Ax = b$ be an $n \times n$ linear system derived from the Poisson equation which holds in domain Ω. Assume that Ω can be partitioned into s subdomains $\Omega_i, i = 1, 2, \ldots, s$. This linear system, associated with the problem, has the following structure:

$$\begin{pmatrix} B_1 & & \varnothing & & G_1 \\ & B_2 & & & G_2 \\ \varnothing & & \ddots & & \vdots \\ & & & B_s & G_s \\ F_1 & F_2 & \ldots & F_s & C \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_s \\ y \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_s \\ g \end{pmatrix}, \tag{4.6}$$

where each x_i represents the subvector of unknowns that are interior to subdomain Ω_i and y represents the vector of all interface unknowns. Assume further that $x \equiv (x_1; x_2; \ldots; x_s)$ has dimension n_1 and y has dimension $n_2 = n - n_1$, i.e., we assume that Ω consists of n_2 interface points and n_1 non-interface points. To facilitate we express the above system in the simpler form:

$$\begin{pmatrix} B & G \\ F & C \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix}. \tag{4.7}$$

Thus, G represents the subdomain to interface coupling seen from the subdomains, whereas F can be associated with the interface to subdomain coupling seen from the interface nodes. From the first block row, the unknown vector x can be computed from

$$Bx = \tilde{f}, \tag{4.8}$$

with $\tilde{f} \equiv f - Gy$. Upon substituting this into the second block equation, the following reduced system is obtained

$$Sy = \tilde{g}, \tag{4.9}$$

with $S \equiv C - FB^{-1}G$ and $\tilde{g} \equiv g - FB^{-1}f$. The latter system (4.9) is also known as the Schur complement system. If the components of this Schur complement system can be formed and the solution exists, all the interface variables y will become available. Once these variables are known, the remaining unknowns can be computed via (4.8).

The above described method of decoupling interior and interface equations is called the Schur complement method. Matrix S is called the Schur complement matrix associated with the variable y. One possible solution method for solving the Schur complement system is the deflation method applied by Mansfield [21].

A practical implementation of the whole solution method based on this SCM approach can be obtained with a Block-Gaussian elimination, see for instance Saad [31] (Section 13.2, Algorithm 13.1). Each subsystem, which has to be solved, can be treated in a direct or iterative way, depending on the size of the subproblems and interfaces. Because of the particular block structure of B, observe that any linear system solution with it decouples in s separate systems which is obviously well-parallelizable.

The above described method of decoupling into block equations is only successful when the number of interface points is limited, i.e., $n_1 \gg n_2$. Therefore, the configuration of the subdomains determines the attractiveness of the SCM. In our 3-D bubbly-flow applications, the method may fail by assuming that each bubble corresponds to a subdomain. This is a consequence of the possible appearance of many bubbles and a possible finer grid around the interfaces. On the other hand, by taking an alternative configuration, where the subdomains are considered to be fixed parts of Ω not depending on the location of the bubbles, can be more attractive to apply the SCM. Note that in the latter approach, each subdomain can consist of several bubbles. Both configurations are graphically given in Figure 4.2.

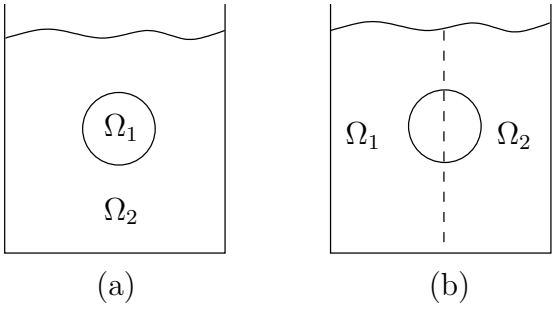


Figure 4.2: Simple problem with one bubble: (a) subdomains based on the bubble and (b) fixed subdomains independent of the bubble.

4.3.2 Additive Schwarz Method (ASM)

The additive Schwarz method is a variant of the original alternating procedure described by Schwarz in 1870. It consists of three parts:

- alternating between two overlapping domains;
- solving the Dirichlet problem on one domain at each iteration;
- taking boundary conditions based on the most recent solution obtained from the other domain.

This procedure is also called the multiplicative Schwarz procedure which is strongly related to the block Gauss-Seidel iterations. The analogue of the block-Jacobi procedure is known as the additive Schwarz procedure which is described below.

Let A_{ii} be the corresponding submatrix of the subdomain Ω_i with certain boundary conditions which holds for all $i = 1, 2, \ldots, s$. Moreover, let R_i an $n_i \times n$ restriction matrix formed by some rows of the identity matrix related to subdomain Ω_i. The transpose R_i^T is a prolongation operator which takes a variable from Ω_i and extends it to the equivalent variable in Ω. With this interpretation we can write:

$$A_{ii} = R_i A R_i^T, \tag{4.10}$$

where A_{ii} is of dimension $n_i \times n_i$ and defines a restriction of A to Ω_i. A problem associated with A_{ii} can be solved which would update the unknowns in the domain Ω_i.

The ASM consists of updating all the new block components from the same residual. It differs from the multiplicative procedure only because the components in each subdomain are not updated until a whole cycle of updates through all domains are completed. The basic additive Schwarz iteration would therefore be as follows (Saad [31], Section 13.3.3):

1. **For** $i = 1, 2, \ldots, s$ **Do**
2. Compute $\delta_i := R_i^T A_{ii}^{-1} R_i (b - Ax)$
3. **EndDo**
4. $x_{new} := x + \sum_{i=1}^{s} \delta_i$

Each instance of the loop redefines different components of the new approximation x_{new} and there is no data dependency between the subproblems involved in the loop. Consequently, the subproblems are well-parallelizable.

It can be easily verified that the above described basic additive Schwarz iteration is equivalent with the block-Jacobi iteration (cf. Algorithm 4.1 of Saad [31]).

Analogous to the SCM, when the bubbles are chosen to be the subdomains, the additive Schwarz method may also be difficult applicable in problems of 3-D bubbly flows, due to the possibly large number of bubbles (i.e., $s \gg 1$) and the non-stationary position of the bubbles. However, the ASM can be still useful by considering fixed subdomains (cf. Figure 4.2(b)).

4.4 Parallel Preconditioners

In DICCG the incomplete Cholesky preconditioner $M_{IC} = LD^{-1}L^T$ is used as preconditioner. However, this preconditioner is difficult to parallelize, since

linear systems with the preconditioner can only be solved sequentially due to the triangular form of L.

Alternatively, many parallel preconditioners are proposed, see e.g. Demmel *et al.* [7] for a survey. A number of approaches to obtain parallelism in the preconditioning part are (cf. [7], Section 8.1):

1. reordering the computations;
2. reordering the unknowns;
3. forcing parallelism.

We restrict ourselves to the latter class: we force parallelism by simply neglecting couplings to unknowns residing in other processors. This is like variants of the ILU preconditioner and the block Gauss-Jacobi (briefly: block-Jacobi) preconditioner. The latter preconditioner is treated in more detail below.

4.4.1 Block-Jacobi preconditioner

In the previous section we have mentioned that the SCM and ASM lead to the preconditioning matrix, which is the block-Jacobi matrix M_{jac}. In case of the SCM approach, the linear system $M_{jac}z = y$ is solved accurately, since this can theoretically be reduced to a system only involving unknowns near the interfaces.

However, since the subdomain problems have often a similar non-zero structure as the original matrix and since they may still be quite large, it is reasonable to solve them (called: the inner iterations) using a Krylov subspace method like the CG method. In this case, this is the second iterative method because the outer iterations are also obtained with a Krylov subspace method. A question which arises naturally, address the tolerance to which the inner iterations should converge . It seems senseless, for example, to solve the subdomain problems with a much smaller tolerance than is desired for the global solution. The influence of the accuracy of the subdomain solution on the convergence has been investigated in e.g. Brakkee *et al.* [1]. Although the number of outer iterations increased, a large gain in CPU time has been observed on a sequential computer, when the subdomain accuracy is relatively low.

The inaccurate solving of subdomain problems is in fact the ASM approach. Algebraically, the ASM preconditioner M_{ASM} satisfies

$$M_{ASM}^{-1} = \sum_{i=1}^{s} R_i^T \tilde{A}_{ii}^{-1} R_i, \tag{4.11}$$

where $\tilde{A}_{ii}^{-1}$ is the inaccurate solution of $\tilde{A}_{ii} z_i = y_i$ obtained with the CG method. Equivalently, we can write:

$$M_{SC2}^{-1} = \begin{pmatrix} \tilde{A}_{11}^{-1} & & & \varnothing \\ & \tilde{A}_{22}^{-1} & & \\ & & \ddots & \\ \varnothing & & & \tilde{A}_{ss}^{-1} \end{pmatrix}, \tag{4.12}$$

which is suitable for parallelization due to the block-diagonal structure of the preconditioner. It is known from numerical experiments that the number of

outer iterations decreases compared by applying the standard incomplete Cholesky preconditioner. However, the overall computational time can be (drastically) lower by using the parallel inaccurate block-Jacobi preconditioner.

Note that since the number of inner iterations may vary in each outer iteration, the effective preconditioner (4.12) is non-linear and varies in each outer iteration. In this case we have to be careful with applying the preconditioned CG method, see e.g. Notay [27]. He introduced flexible CG to overcome the slightly varying preconditioner in each iterate. The main difference with the standard CG method lies in the explicit orthogonalization of the search direction vectors. More concretely, let p_i be the search direction in the i-th iterate. Then p_i is orthogonalized with respect to the $(\cdot,\cdot)_A$ inner product against the previous vectors $p_1,\ldots,p_{i-1}$. Restarting and truncation techniques have also been proposed in [27] to avoid memory problems.

From Vuik *et al.* [40] one can conclude that the block-Jacobi preconditioner is perfectly parallel applicable and gives better performance than a simple diagonal scaling. However, the convergence rate degrades substantially as the number of blocks increases. Fortunately, recently several remedies have been proposed to overcome this and are shortly described below.

The first remedy is applying a deflation technique as described in Section 4.2. In fact, this is a form of coarse grid technique leading to a two-level ASM preconditioner M_{ASM2}:

$$M_{ASM2}^{-1} = R_H^T \tilde{A}_H^{-1} R_H + \sum_{i=1}^{s} R_i^T \tilde{A}_{ii}^{-1} R_i, \tag{4.13}$$

where $R_H^T \tilde{A}_H^{-1} R_H$ is the coarse grid correction or a form of deflation, see e.g. [39]).

Other remedies are based on overlap are proposed to make the block-Jacobi preconditioner more robust, see e.g. Cai & Sarkis [4]. However, drawbacks of methods with overlapping subdomains is that the amount of work increases proportionally to the overlap and that these methods are not easy to implement on top on existing software packages ([39], Section 3). Therefore, in this report we restrict ourselves to the non-overlapping approaches of coarse grid correction and deflation and neglect overlapping methods.

4.5 Example

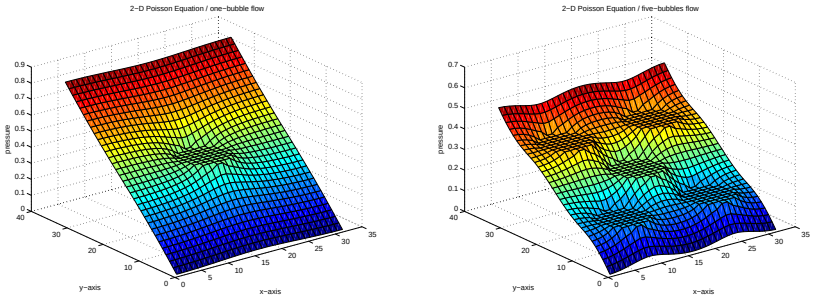
We consider an $n \times n$ SPD linear system $Ax = b$ derived from the 2-D Poisson equation:

$$\nabla \cdot \left(\frac{1}{\rho} \nabla \phi \right) = f, \quad \phi = \phi(\mathbf{x}) \in \Omega, \tag{4.14}$$

where implicitly Neumann conditions are imposed on $\partial\Omega$. This means that solution ϕ is uniquely determined up to a constant vector. The equation (4.14), the source term f and the artificially boundary conditions are derived from the pressure-correction method which is applied on the Navier-Stokes (NS) equations, see Chapter 2. In this case, ρ is piecewise constant with a relatively high contrast ϵ:

$$\rho = \begin{cases} \rho_0 = 1, & \mathbf{x} \in \Lambda_0, \\ \rho_1 = \epsilon \ll 1, & \mathbf{x} \in \Lambda_1, \end{cases} \tag{4.15}$$

where Λ_0 is the main fluid of the flow around the bubbles and Λ_1 is the region inside the bubbles. The geometry of the problem can be seen in Figure 4.3 where plots are shown with different number of bubbles. We start with one bubble with radius γ in the middle of the 2-D unit domain. In future more bubbles will be treated.



(a) Flow with one bubble.

(b) Flow with five bubble.

Figure 4.3: Two examples of bubbly flows.

Since artificial Neumann conditions holds on $\partial\Omega$, matrix A and therefore also E are singular. The singularity of A does not cause problems for the CG method since it can deal with singular systems provided that $b \in \text{Col}(A)$. However, E^{-1} should be determined with a direct solver. Therefore, E has to be modified such that it is invertible. For the sake of simplicity, we assume temporarily that A is non-singular which can easily be achieved by doubling the first diagonal element of A. In this case E^{-1} can be determined in a direct way.

We take 64 grid poins each direction, i.e., $n = n_x \times n_y = 64 \times 64$. Moreover, the zero starting vector, contrast $\epsilon = 10^{-3}$ and CG-tolerance $||M^{-1}Pr_k||_2/||M^{-1}Pb||_2 < 10^{-7}$ are chosen.

Next, some MATLAB experiments are presented for the 2-D bubbly flows. We compare the original and deflated block-Jacobi preconditioner with the original and deflated diagonal and incomplete Cholesky preconditioners. The non-deflated versions are denoted by BJCG_j, DCG and ICCG, respectively. Furthermore, the deflated versions are denoted by $\text{DBJCG}_j - k$, DDCG$-k$ and DICCG$-k$, respectively. Parameter j is the number of blocks and k the number of deflation vectors.

Results

Some results considering the above problem setting can be found in Table 4.1. The number of iterations and the CPU time are given to find the solution. Note that the CPU time measured in MATLAB is only a rough indication of efficiency. Note that the computations have been done on a sequential computer. Later on we will switch to a parallel environment.

For the non-deflated methods, we conclude from the table that BJCG_4 needs the fewest iterations, but ICCG is the most efficient method considering the

Method	# Iterations	CPU Time (sec)
DCG	394	2.1
DDCG−4	286	2.3
DDCG−16	115	0.9
DDCG−64	58	0.5
DDCG−256	31	0.3
ICCG	127	1.4
DICCG−4	100	1.3
DICCG−16	54	0.7
DICCG−64	28	0.4
DICCG−256	17	0.3
BJCG_4	75	4.8
BJCG_{16}	142	7.5
BJCG_{64}	292	2.3
$\text{DBJCG}_4 - 4$	56	3.9
$\text{DBJCG}_4 - 16$	39	2.6
$\text{DBJCG}_4 - 64$	26	1.8

Table 4.1: Number of iterations for (D)DCG, (D)ICCG and (D)BJCG in the Poisson problem with one bubble.

CPU times. For the deflated methods, the most efficient results are achieved by the DICCG method while the fewest iterations are reached with DBJCG.

Obviously, the number of iterations of (D)BJCG increases when the number of blocks j in the BJ-preconditioner grows. Moreover, the CPU times of (D)BJCG are somewhat disappointing but later on these can be lowered when parallel computers are used.

The main conclusion we can draw is that incorporating subdomain deflation accelerates the convergence of the iterative methods DCG, ICCG and BJCG significantly and these deflated versions are relatively efficient considering the CPU times. The largest differences can be observed by comparing DCG with DDCG$-k$.

Varying the grid sizes

We vary n_x and n_y and investigate the influence on the number of iterations. Moreover, the number of deflation vectors is chosen equal to the number of grid points in $n_x (= n_y)$ The results can be found in Table 4.2.

n_x, n_y, k	DCG	DDCG	ICCG	DICCG	BJCG_4	DBJCG_4
16	90	33	34	19	38	29
64	394	115	127	28	75	39
256	> 1000	464	529	53	–	–

Table 4.2: Number of iterations for the iterative methods where $n_x = n_y = k$, i.e., where the number of deflation vectors is equal to the number of grid points in each direction. Note that '–' means 'out of memory'.

From the non-deflated methods of the table one observes that the number of

iterations grows very fast when the grid sizes are increased. The deflation techniques remedy this partially, since the number of iterations of DDCG, DICCG and DBJCG increases slowly by enlarging the grid sizes.

Note that the deflated variants of DCG and ICCG for $m > 16$ and $n_x = n_y = 64$ converge with more or less the same CPU time (see also 4.1). However by enlarging the grid sizes, DICCG becomes much more attractive.

We end with the remark that (D)BJCG can be made more robust (i.e., independent of the grid sizes) by increasing the number of blocks proportionally to n_x and n_y. This is left for future.

Chapter

5 Conclusions & Outlook

In this report we have presented an overview of the literature considering the parallel deflated-preconditioned CG method applied to moving boundary problems. Some main results are listed below.

- The Navier-Stokes equations can be treated with the pressure-correction method. The Poisson equation with piecewise-constant density is the difficult one to solve in this method.
- Many variants and namings of deflation/projection methods are available which are comparable with each other.
- Also many methods are available to choose the deflation vectors which are based on approximate eigenvectors, solutions at previous time steps or divided subdomains.
- The original CG method and the deflation technique are well-parallellizable. In a parallel environment the block-Jacobi preconditioner is frequently embedded in the deflated CG method rather than preconditioners based on incomplete Cholesky decomposition.

Moreover, in the examples of the overview, we have also given results of some small numerical experiments. The conclusions of these results are drawn below.

- Applying the subdomain deflation technique (resulting in the DICCG method) in the 2-D Poisson problem with constant density leads to favorable results. Choosing blocks as subdomains give namely a significant decrease of the number of iterations and also the CPU time compared with ICCG.
- In the 1-D layer problem (1-D Poisson problem with piecewise constant density) we have used algebraic vectors to investigate the deflation method. A few algebraic vectors have been required to reduce the number of iterations significantly.

- The 2-D Poisson problem with two densities in the domain derived from the pressure correction method has been tested. Again, the subdomain deflation technique has performed very well in combination with diagonal, incomplete Cholesky or block-Jacobi preconditioners in the CG method.

We have ideas for further research. Some of these main ideas are given below.

- The examples will be generalized to the 3-D case. Also, the parallel aspects will be considered.
- In the subdomain deflation approach we have chosen for constant deflation vectors. We will investigate the possibilities of embedding linear and quadratic vectors in this approach.
- The different variants of the deflation techniques will be compared in more detail.
- More research will be done in applying deflation techniques in moving boundary problems.

Bibliography

[1] E. Brakkee, C. Vuik, P. Wesseling, *Domain decomposition for the incompressible Navier-Stokes equations: solving subdomain problems accurately and inaccurately*, Int. J. for Num. Meth. Fluids, **26**, pp. 1217–1237, 1998.

[2] J.H. Bramble, J.E. Pasciak, A.H. Schatz, *The construction of preconditioners for elliptic problems by substructuring*, I, Math. Comp, **47** , pp. 103–134, 1986.

[3] K. Burrage, J. Erhel, B. Pohl, A. Williams. *A deflation technique for linear systems of equations*, SIAM Journal on Science and Computation, 19(4), pp. 1245–1260, 1998.

[4] X. C. Cai and M. Sarkis, *A restricted additive Schwarz preconditioner for general sparse linear systems*, SIAM J. Sci. Comput., **21** , pp. 792–797, 1999.

[5] A. Chapman, Y. Saad, *Deflated and augmented Krylov subspace techniques*, Numer. Linear Algebra Appl., **4**, pp. 43–66, 1997.

[6] M. Clemens, M. Wilke, R. Schuhmann, T. Weiland, *Subspace Projection Extrapolation Scheme for Transient Field Simulations*, IEEE Transactions on Magnetics, **40**(2), pp. 934–937, 2004.

[7] J.W. Demmel, M. T. Heath, H.A. van der Vorst, *Parallel Numerical Linear Algebra*, Computer Science Division Technical Report UCB–CSD–92–703, U.C. Berkeley, 1992.

[8] M. Dryja, O. Widlund, *Multilevel additive methods for elliptic finite element problems*, in *Parallel Algorithms for Partial Differential Equations*, Proceedings of the Sixth GAMM-Seminar, Vieweg & Son Braunscweig Germany, pp. 58–69, 1991.

[9] I. S. Duff, H. A. van der Vorst, *Developments and trends in the parallel solution of linear systems*, Parallel Computing, **25**(13-14), pp. 1931–1970, 1999.

[10] P. F. Fischer, *An overlapping Schwarz method for spectral element solution of the incompressible Navier-Stokes equations*, Journal of Computational Physics, **133**(1), pp. 84–101, 1997.

[11] J. Frank, C. Vuik, *On the construction of deflation-based preconditioners*, SIAM Journal on Scientific Computing, **23**, pp. 442–462, 2001.

[12] H. de Gersem, K. Hameyer, *A deflated iterative solver for magnetostatic finite element models with large differences in permeability*, Eur. Phys. J. Appl. Phys., **13**, pp. 45–49, 2000.

[13] L. Giraud, S. Gratton, *On the sensitivity of some spectral preconditioners*, Technical Report TR/PA/04/108 (see `http://www.cerfacs.fr/~giraud/recentPub.html`), CERFACS, Toulouse, France, 2004.

[14] G.H. Golub, C.F. van Loan, *Matrix Computations*, Third Edition, The John Hopkins University Press, Baltimore, Maryland 21218, 1996.

[15] M.J. Hagger, *Automatic domain decomposition on unstructured grids (DOUG)*, Advances in Computational Mathematics, **9**, pp. 281–310, 1998.

[16] E.F. Kaasschieter, *Preconditioned conjugate gradients for solving singular systems*, Journal of Computational and Applied Mathematics , **24**, pp. 265–275, 1988.

[17] J.J.I.M. van Kan, *A second-order accurate pressure correction method for viscous incompressible flow*, SIAM J. Sci. Stat. Comp., **7**, pp. 870–891, 1986.

[18] L. Yu. Kolotilina, *Preconditioning of systems of linear algebraic equations by means of twofold deflation*, I. Theory, J. Math. Sci., **89**, pp. 1652–1689, 1998.

[19] J. Mandel, *Balancing Domain Decomposition*, Comm. Numer. Meth. Engrg., **9**, pp. 233–241, 1993.

[20] L. Mansfield, *Damped Jacobi preconditioning and coarse grid deflation for Conjugate Gradient iteration on parallel computers*, SIAM J. Sci. Stat. Comput., **12**, pp. 1314–1323, 1991.

[21] L. Mansfield, *On the Conjugate Gradient Solution of the Schur Complement System Obtained from Domain Decomposition*, SIAM J. Num. Anal., **27**, pp. 1612-1620, 1990.

[22] J.A. Meijerink, 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**, pp. 148–162, 1977.

[23] R.B. Morgan, *A restarted GMRES method augmented with eigenvectors*, SIAM J. Matrix Analysis and Applications, **16**, pp. 1154–1171, 1995.

[24] R. Nabben, C. Vuik, *A comparison of Deflation and Coarse Grid Correction applied to porous media flow*, Delft University of Technology, Department of Applied Mathematical Analysis, Report 03-10, ISSN 1389-6520, 2003.

[25] R. Nabben, C. Vuik, *A comparison of Deflation and the balancing Neumann-Neumann preconditioner*, Delft University of Technology, Department of Applied Mathematical Analysis, Report 04-09, ISSN 1389-6520, 2004.

[26] R.A. Nicolaides, *Deflation of Conjugate Gradients with applications to boundary value problems*, SIAM J. Matrix Anal. Appl., **24**, pp. 355–365, 1987.

[27] Y. Notay, *Flexible conjugate gradients*, SIAM J. Sci. Comp., **22**, pp. 1444–1460, 2000.

[28] A. Padiy, O. Axelsson, B. Polman, *Generalized augmented matrix preconditioning approach and its application to iterative solution of ill-conditioned algebraic systems*, SIAM J. Matrix Anal. Appl., **22**, pp. 793–818, 2000.

[29] W. P. Petersen, P. Arbenz, Introduction to Parallel Computing, Oxford University Press, 2004.

[30] S. P. van der Pijl, A. Segal, C. Vuik, P. Wesseling, *A mass-conserving Level-Set method for modelling of multi-phase flows*, Int. J. Num. Meth. in Fluids, **47**(4), pp. 339–361, 2005.

[31] Y. Saad, *Iterative methods for sparse linear systems*, PWS Publishing Company, Boston, 1996.

[32] Y. Saad, M. Yeung, J. Erhel, F. Guyomarc'h, *A deflated version of the Conjugate Gradient Algorithm*, SIAM J. Sci. Comput., **21** (5), pp. 1909–1926, 2000.

[33] A. Segal, C. Vuik, F. J. Vermolen, *A conserving discretization for the free boundary in a two-dimensional Stefan problem*, J. Comp. Phys., **141**, pp. 1–21, 1998.

[34] A. van der Sluis, H.A. van der Vorst, *The rate of convergence of Conjugate Gradients*, Num. Math., **48**, pp. 543–560, 1999.

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

[36] J. Verkaik, *Deflated Krylov-Schwarz Domain Decomposition for the Incompressible Navier-Stokes Equations on a Colocated Grid*, Master's thesis (see `http://ta.twi.tudelft.nl/nw/users/vuik/numanal/verkaik_afst.pdf`), TU Delft, 2003.

[37] F.J. Vermolen and C. Vuik, *A mathematical model for the dissolution of particles in multi-component alloys*, J. Comp. Appl. Math., **126**, pp. 233–254, 2000.

[38] F. Vermolen, C. Vuik, A. Segal, *Deflation in preconditioned Conjugate Gradient methods for finite element problems*, In: Conjugate Gradient and Finite Element Methods (Ed: M. Krizek and P. Neittaanmaki and R. Glowinski and S. Korotov), Springer, Berlin, pp. 103–129, 2004.

[39] C. Vuik, J. Frank, *Coarse grid acceleration of a parallel block preconditioner*, Future Generation Computer Systems, **17**, pp. 933–940, 2001.

[40] C. Vuik, J. Frank, A. Segal, *A parallel block-preconditioned GCR method for incompressible flow problems*, Future Generation Computer Systems, **18**, pp. 31–40, 2001.

[41] C. Vuik, J. Frank, F.J. Vermolen, *Parallel Deflated Krylov methods for incompressible flow*, in: Parallel Computational Fluid Dynamics: Practice and Theory (see [47]), pp. 381–388, 2002.

[42] C. Vuik, A. Segal, L. El Yaakoubi, E. Dufour, *A comparison of various deflation vectors applied to elliptic problems with discontinuous coefficients*, Applied Numerical Mathematics, **41**, pp. 219–233, 2002.

[43] C. Vuik, A. Segal, J.A. Meijerink, *An efficient preconditioned CG method for the solution of a class of layered problems with extreme contrasts in the coefficients*, J. Comp. Phys., **152**, pp. 385–403, 1999.

[44] C. Vuik, A. Segal, J.A. Meijerink, G.T. Wijma, *The construction of projection vectors for a Deflated ICCG method applied to problems with extreme contrasts in the coefficients*, Journal of Computational Physics, **172**, pp. 426–450, 2001, (see also Shell Report EP2000-8019).

[45] H. Waisman, J. Fish, R. S. Tuminaro, J. Shadid, *The generalized global basis (GGB) method*, International Journal for Numerical Methods in Engineering, **61**(8), pp. 1243–1269, 2004.

[46] H. Waisman, J. Fish, R. S. Tuminaro, J. Shadid, *Acceleration of the generalized global basis method for nonlinear problems*, Submitted to Journal of Computational Physics, 2004.

[47] P. Wilders, A. Ecer, J. Periaux, N. Satofuka, P. Fox (*eds.*), *Parallel Computational Fluid Dynamics: Practice and Theory*, Proceedings of the Parallel CFD 2001 Conference (Egmond aan Zee, The Netherlands, May 21-23), North-Holland, 2002.

Appendix

A DICCG Algorithm

The DICCG with $M = LD^{-1}L^T$ is described in more detail below.

Algorithm 2 DICCG Algorithm

Choose x_0 arbitrarily;
$r_0 := b - Ax_0$;
Make Z;
$E^{-1} := (Z^T AZ)^{-1}$;
$P := I - AZE^{-1}Z^T$;
$u := ZE^{-1}Z^T b$;
$\hat{r}_0 := Pr_0$;
Make L and D so that $A \approx LD^{-1}L^T$;
Solve z_0 from $LD^{-1}L^T z_0 = r_0$;
$p_1 := z_0$;
$j := 0$;
Choose ϵ and $It_{\max}$;
while $j < It_{\max}$ **and** $||z_j||_2 > \epsilon$ **do**
    $j := j + 1$;
    $w_j := PAp_j$;
    $\alpha_j := \frac{(\hat{r}_{j-1}, z_{j-1})}{(p_j, w_j)}$;
    $\tilde{x}_j := \tilde{x}_{j-1} + \alpha_j p_j$;
    $\hat{r}_j := \hat{r}_{j-1} - \alpha_j w_j$;
    Solve z_j from $LD^{-1}L^T z_j = r_j$;
    $\beta_j := \frac{(\hat{r}_j, z_j)}{(\hat{r}_{j-1}, z_j)}$;
    $p_{j+1} := z_j + \beta_j p_j$;
end while
$x := u + P^T \tilde{x}_j$.

Appendix

B Comparison of Flops

To compare the number of flops between DICCG and ICCG we assume that:

- A has sizes $n \times n$ and consists of 5 non-zero diagonals (derived from e.g. the 2-D Poisson or Helmholtz problem);
- b has length n;
- Z has sizes $n \times r$;
- Z consists of vectors such that each row has exactly one non-zero element and each column has approximately n/r non-zero elements. Thus in total there are n non-zero elements in Z. In this appendix, we consider Z which is obtained with subdomain deflation with blocks.

As a consequence, Z and AZ have approximately the same form and the same number of non-zeros. Moreover, considering the latter assumption, we note that Z consists of orthogonal columns, i.e.,

$$Z^T Z = \alpha I, \tag{B.1}$$

where $\alpha = n/r$ and I is the identity matrix, whereas

$$E \equiv Z^T A Z, \tag{B.2}$$

with E is an SPD matrix with five diagonals and E has bandwidth $\sqrt{r}$. When we want to solve x from the system $Ex = y$ with y a known vector, we can do that by applying a band Cholesky algorithm instead of computing the inverse E^{-1}. Since E is SPD, we can write $E = LD^{-1}L^T$. Therefore a careful flop count of the steps:

$$E = LD^{-1}L^T,\ Lx_1 = y,\ D^{-1}x_2 = x_1,\ L^T x = x_2, \tag{B.3}$$

reveals that

$$r\left(r + 8\sqrt{r} + 1\right)\ \text{flops} \tag{B.4}$$

are required (see Golub & Van Loan [14], p. 156, by taking $p = \sqrt{r}$ and $n = r$).

We make some assumptions with respect to the (approximated) number of flops in the table below, where x, y, z_1, z_2 are vectors of length n. In the analysis, we shall neglect $\mathcal{O}(1)$ terms.

	Operation	**# Flops**	**Explanation**
(a)	(x, y)	$2n$	
(b)	$x + y$	n	
(c)	Ax	$9n$	A consists of 5 non-zero diagonals.
(d)	Making L and D	$8n$	Constructing L and D as in the standard IC-decomposition.
(e)	Solving x from $LD^{-1}L^T x = y$	$11n$	First solving $L^T x = z_1$ ($5n$ flops), thereafter $D^{-1}z_1 = z_2$ (n flops), and $Lz_2 = y$ ($5n$ flops).

Now we can derive the number of flops for constructing E^{-1} and AZ.

	Operation	**# Flops**	**Explanation**
(f)	AZ	$9n$	Z is sparse and consists of n non-zeros.
(g)	$E := Z^T(AZ)$	$2n$	AZ has almost the same form as Z.
(h)	E^{-1}	$r\,(r + 8\sqrt{r} + 1)$	Note: the inverse is not computed explicitly. Expression (B.4) is used.

In total there are

$$11n + r\left(r + 8\sqrt{r}\right) \text{ flops} \tag{B.5}$$

required in preparation for creating P. However, in practice the projection matrix P is never computed. The operation Px for arbitrary vector x is computed in the following sense.

	Operation	**# Flops**	**Explanation**
(i)	$x_1 := Z^T x$	$2n$	Each row-vector product costs $2n/r$ flops
(j)	$x_2 := E^{-1}x_1$	$(2\sqrt{r} + 1)r$	$Ex_2 = x_1$ is solved with (h) using Choleski decomposition. E has a band structure with bandwidth $\sqrt{r}$.
(k)	$x_3 := (AZ)x_2$	n	AZ has already computed and is almost of the same form as Z.
(l)	$Px := x - x_3$	n	

Thus, computing Px costs in total

$$2r\sqrt{r} + 4n + r \text{ flops.} \tag{B.6}$$

Extra Flops Applying Deflation in ICCG

Compared to ICCG, DICCG requires a few extra computations, namely:

	Operation	**# Flops**	**Explanation**
(m)	Make Z	n	
(n)	$u := ZE^{-1}Z^T b$	$4n + (2\sqrt{r} + 1)r$	Using (i), (j) and $u := Zx_2$ costs $2n$ flops.
(o)	$\hat{r}_0 := Pr_0$	$2r\sqrt{r} + 4n + r$	Using Expression (B.6).
(p)	$x := u + P^T\tilde{x}_j$	$2r\sqrt{r} + 4n + r$	Using (B.6).

Define F_{def} to be the extra flops required in the computations when we apply deflation in ICCG, i.e., F_{def} is the summation of the flops of the latter table and including the preparation flops (B.5). Then:

$$F_{def} = 23n + (11r + 2)\sqrt{r} + r^2 + 3r. \tag{B.7}$$

Basis Flops

Before entering the WHILE-loop, the following steps are required in both DICCG and ICCG:

	Operation	**# Flops**	**Explanation**
(q)	Set x_0	n	
(r)	$r_0 := b - Ax_0$	$10n$	Using (c).
(s)	Make L and D	$8n$	Using (d).
(t)	Solve z_0	$11n$	Using (e).
(u)	$p_1 := z_0$	n	
(v)	$j := 0$	–	
(w)	Set ϵ and $It_{\max}$	–	

F_{basis} is the summation of the flops of the latter table. Then:

$$F_{basis} = 31n. \tag{B.8}$$

Flops in each iterate of ICCG and DICCG

We compute the number of flops required in each iterate of the WHILE-loop of both ICCG and DICCG.

Operation	**# Flops**	**Explanation**
$j < It_{\max}$ and $\|\|z_j\|\|_2 > \epsilon$	$2n$	
$j := j + 1$	–	
$w_j := PAp_j$	$9n$ DICCG: $13n + 2r\sqrt{r} + r$	Note that $PA = A$ in ICCG. Using (c) and (B.6).
$\alpha_j := \frac{(\hat{r}_{j-1}, z_{j-1})}{(p_j, w_j)}$	$4n$	
$\tilde{x}_j := \tilde{x}_{j-1} + \alpha_j p_j$	$2n$	
$\hat{r}_j := \hat{r}_{j-1} - \alpha_j w_j$	$2n$	
Solve z_j from $LD^{-1}L^T z_j = r_{j+1}$	$11n$	Using (e).
$\beta_j := \frac{(\hat{r}_j, z_j)}{(\hat{r}_{j-1}, z_j)}$	$4n$	
$p_{j+1} := z_j + \beta_j p_j$	$2n$	

Therefore, DICCG needs approximately $40n + 2r\sqrt{r}$ flops each iterate, while ICCG needs $36n$ flops each iterate. We denote the number of iterates of ICCG and DICCG by I_{iccg} and I_{diccg}, respectively. Then the total number of flops in and outside the WHILE-loop (Lines 1–10 in DICCG algorithm) is approximately:

$$F_{iccg} = 36nI_{iccg} + F_{basis} \quad \text{and} \quad F_{diccg} = (40n + 2r\sqrt{r})I_{diccg} + F_{def} + F_{basis},$$

where in F_{diccg} the extra flops F_{def} from the computations with deflation (B.7) are added. Moreover, in both variants the basis steps F_{basis} are also added. We obtain:

$$F_{iccg} = 36nI_{iccg} + 31n$$

and

$$F_{diccg} = (40n + 2r\sqrt{r})I_{diccg} + 54n + (11r + 2)\sqrt{r} + r^2 + 3r.$$

Comparison ICCG and DICCG

A comparison of the number of flops between ICCG and DICCG can be made in the following way. Therefore, we define $\phi(r, n)$ as follows [1]

$$\phi(r, n) = F_{diccg} - F_{iccg}, \tag{B.9}$$

resulting in:

$$\phi(r, n) = (40n + 2r\sqrt{r})I_{diccg} + 25n + 7r^2 + r\left(8\sqrt{r} + 1\right) - 36nI_{iccg}. \tag{B.10}$$

This function $\phi(r, n)$ determines the difference in flops between the DICCG and ICCG, for given r and n. If $\phi(r, n) > 0$, then ICCG requires less computational work relative to DICCG, while $\phi(r, n) < 0$ means that DICCG is more efficient than ICCG. [2]

[1] We can also choose for $\tilde{\phi}(r, n) = F_{diccg}/F_{iccg}$. In this case, the deflation technique is efficient when $\tilde{\phi}(r, n) < 1$.

[2] Note that if $I_{diccg} \gg 1$, then $(40n + 2r\sqrt{r})I_{diccg} \gg 25n + 7r^2 + r\left(8\sqrt{r} + 1\right)$. As a consequence:

$$\phi(r, n) \approx (40n + 2r\sqrt{r})I_{diccg} - 36nI_{iccg}. \tag{B.11}$$