DELFT UNIVERSITY OF TECHNOLOGY

REPORT 12-10

# Fast linear solver for pressure computation in layered domains

P. van Slingerland and C. Vuik

ISSN 1389-6520

Reports of the Department of Applied Mathematical Analysis

Delft 2012

# Fast linear solver for pressure computation in layered domains

P. van Slingerland C. Vuik

August 14, 2012

**Abstract**

Accurate simulation of fluid pressures in layered reservoirs with strong permeability contrasts is a challenging problem. For this purpose, the Discontinuous Galerkin (DG) method can be particularly suitable. This discretization scheme uses discontinuous piecewise polynomials to combine the best of both classical finite element methods and finite volume methods. However, due to the relatively large number of unknowns per mesh elements, and the extreme variations in the permeability in the layered structure, standard linear solvers usually yield long computational times and/or low accuracy of the approximated fluid pressures.

To increase the efficiency of the Conjugate Gradient (CG) method for linear systems resulting from Symmetric Interior Penalty (discontinuous) Galerkin (SIPG) discretizations, we have cast an existing two-level preconditioner into the deflation framework. The main idea is to use coarse corrections based on the DG solution with polynomial degree p=0. This report provides a numerical comparison of the performance of both two-level methods in terms of scalability and overall efficiency. Furthermore, it studies the influence of the SIPG penalty parameter, the smoother, damping of the smoother, and the strategy for solving the coarse systems.

We have found that the penalty parameter can best be chosen diffusion-dependent. In that case, both two-level methods yield fast and scalable convergence. Coarse systems can be solved efficiently by applying the CG method again in an inner loop with low accuracy. Whether preconditioning or deflation is to be favored depends on the choice for the smoother and on the damping of the smoother. Altogether, both two-level methods can contribute to faster and more accurate fluid pressure simulations.

## 1 Introduction

Layered reservoirs often exhibit very strong permeability contrasts with typical values between $10^{-1}$ and $10^{-7}$. Solving for the pressure in such a system can be numerically challenging. The governing equation is a mildly nonlinear diffusion equation with time-varying coefficients obtained by combining mass conservation and Darcy's law.

To discretize this equation in space, the Discontinuous Galerkin (DG) method can be particularly suitable (cf. Rivière et al. (2000), Sun and Wheeler (2007), Rivière (2008), Arnold et al. (2002) and references therein). This discretization scheme can be interpreted as a finite volume method that uses piecewise polynomials of degree $p$ rather than piecewise constant functions. As such, it combines the best of both classical finite element methods and finite volume methods, making it particularly suitable for handling non-matching grids and designing hp-refinement strategies.

However, after linearization of the discretized equations, the resulting linear systems are usually larger than those for the aforementioned classical discretization schemes. This is due to the larger number of unknowns per mesh element. At the same time, the condition number typically increases with the number of mesh elements, the polynomial degree, and the stabilization factor (Castillo (2002); Sherwin et al. (2006)). The strong permeability contrasts in the problem sketched above pose an extra challenge. Altogether, standard linear solvers often result in long computational times and/or low accuracy of the approximated fluid pressures.

In search of efficient and scalable algorithms (for which the number of iterations does not increase with e.g. the number of mesh elements), much attention has been paid to subspace correction methods (Xu (1992)). Examples include classical geometric (h-)multigrid (Brenner and Zhao (2005); Gopalakrishnan and Kanschat (2003)), spectral (p-)multigrid (Fidkowski et al. (2005); Persson and Peraire (2008)), algebraic multigrid (Prill et al. (2009); Saad and Suchomel (2002)), and Schwarz domain decomposition (Antonietti and Ayuso (2007); Feng and Karakashian (2001)). Usually, these methods can either be used as a standalone solver, or as a preconditioner in an iterative Krylov method. The latter tends to be more robust for problems with a few isolated 'bad' eigenvalues, as is the case for the strongly varying problems of our interest.

An alternative for preconditioning is the method of deflation, originally proposed by Nicolaides (1987). This method has been proved effective for layered problems with extreme contrasts in the coefficients by Vuik et al. (1999). Deflation is related to multigrid in the sense that it also makes use of a coarse space that is combined with a smoothing operator at the fine level. This relation has been considered from an abstract point of view by Tang et al. (2009, 2010).

This research seeks to extend this comparison between preconditioning and deflation in the context of DG schemes. In particular, it is focused on the Conjugate Gradient (CG) method for linear systems resulting from Symmetric Interior Penalty (discontinuous) Galerkin (SIPG) discretizations for stationary diffusion problems with extreme contrasts in the coefficients.

Starting point of this research is one of the two-level methods proposed by Dobrev et al. (2006). This method uses coarse corrections based on the DG discretization with polynomial degree $p = 0$. Using the analysis by Falgout et al. (2005), they have shown theoretically (for $p = 1$) that this preconditioner yields scalable convergence of the CG method, independent of the mesh element diameter. Another nice property is that the use of only two levels offers an appealing simplicity. More importantly, the coefficient matrix that is used for the coarse correction is quite similar to a matrix resulting from a central difference discretization, for which very efficient solution techniques are readily available.

To extend the work of Dobrev et al. (2006), we have cast the two-level preconditioner into the deflation framework, using the abstract analysis of Tang et al. (2009). Furthermore, we have conducted several numerical experiments to compare the scalability and the overall efficiency of both two-level methods. These results (including $p > 1$) complement the theoretical analysis for the preconditioning variant for $p = 1$ by Dobrev et al. (2006). Additionally, we have investigated how the efficiency of the CG method is influenced by the SIPG penalty parameter, the smoother, damping of the smoother, and the strategy for solving the coarse systems.

The outline of this report is as follows. Section "*Discretization*" discusses the SIPG method for diffusion problems with large jumps in the coefficients. To solve the resulting linear systems, Section "*Linear solver*" discusses the existing two-level preconditioner and the resulting deflation variant. The performance of both two-level methods is compared numerically in Section "*Numerical experiments*". Section "*Conclusion*" summarizes the main conclusions.

## 2 Discretization

We consider the linearized form of the spatially discretized pressure equation, which can be interpreted as a stationary diffusion equation. This section summarizes the SIPG discretization method for this model problem. Furthermore, it discusses the influence of the so-called penalty parameter that characterizes this DG method.

## 2.1 SIPG method

We study the following model problem on the spatial domain $\Omega$ (with Dirichlet boundary conditions):

$$-\nabla \cdot (K \nabla u) = f. \tag{1}$$

The diffusion (or permeability) coefficient $K$ is a scalar that typically contains large jumps across the domain. The function $f$ is a source term.

The SIPG approximation for the model above can be constructed in the following manner. First, choose a mesh with elements $E_1, ..., E_N$. The numerical experiments in this report are for uniform Cartesian meshes on the domain $\Omega = [0,1]^2$, although our solver can be applied for a wider range of problems.

Next, define the test space $V$ that contains each function that is a polynomial of degree $p$ or lower within each mesh element, and that may be discontinuous at the element boundaries. The SIPG approximation $u_h$ is now defined as the unique element in this test space that satisfies the relation

$$B(u_h, v) = L(v), \qquad \text{for all test functions } v \in V, \tag{2}$$

where $B$ and $L$ are (bi)linear forms that characterize the SIPG method (cf. details below). For a large class of (sufficiently smooth) problems, the SIPG method converges with order $p+1$ (Rivière (2008)).

Details ((Bi)linear forms $B$ and $L$) For uniform Cartesian meshes on the domain $\Omega = [0,1]^2$, the (bi)linear forms $B$ and $L$ in (2) can be specified as follows. First, we require the following additional notation: the Dirichlet boundary conditions are specified as $u = g_D$ on the domain boundary $\partial\Omega$; the vector $\mathbf{n}_i$ denotes the outward normal of mesh element $E_i$; the set $\Gamma_h$ denotes the collection of all interior edges $e = \partial E_i \cap \partial E_j$ in the mesh shared by two adjacent elements; and the set $\Gamma_D$ denotes the collection of all boundary edges $e = \partial E_i \cap \partial \Omega$.

Finally, to deal with the inter-element discontinuities in the test space $V$, we also need to introduce the usual trace operators for jumps and averages at the mesh element boundaries: at each interior element boundary $\partial E_i \cap \partial E_j \in \Gamma_h$, we define:

$$[v] := v_i \mathbf{n}_i + v_j \mathbf{n}_j, \qquad \{v\} := \frac{v_i + v_j}{2},$$

where $v_i$ denotes the trace of the function $v$ along the side of $E_i$. Observe that $[v]$ is a vector, while $v$ is a scalar. Analogously, we define for a vector-valued function $\tau$:

$$[\tau] := \tau_i \cdot \mathbf{n}_i + \tau_j \cdot \mathbf{n}_j, \qquad \{\tau\} := \frac{1}{2}(\tau_i + \tau_j).$$

Observe that $[\tau]$ is a scalar, while $\tau$ is a vector. Similarly, at the domain boundary, we define at each element boundary $\partial E_i \cap \partial\Omega \in \Gamma_D$:

$$[v] := v_i \mathbf{n}_i, \qquad \{v\} := v_i, \qquad [\tau] := \tau_i \cdot \mathbf{n}_i, \qquad \{\tau\} := \tau_i.$$

Now that the required notation has been introduced, the forms $B$ and $L$ in (2) can be defined as:

$$\begin{aligned} L(v) &= \int_\Omega f v - \sum_{e \in \Gamma_D} \int_e \Big( [K \nabla v] + \frac{\sigma}{h} v \Big) g_D, \\ B(u_h, v) &= \sum_{i=1}^N \int_{E_i} K \nabla u_h \cdot \nabla v + \sum_{e \in \Gamma_h \cup \Gamma_D} \int_e \Big( - \{K \nabla u_h\} \cdot [v] - [u_h] \cdot \{K \nabla v\} + \frac{\sigma}{h} [u_h] \cdot [v] \Big), \end{aligned}$$

where $\sigma$ is the so-called penalty parameter. This positive parameter penalizes the inter-element jumps to enforce weak continuity and ensure convergence. Although it is presented as a constant here, its value may vary throughout the domain. This is further discussed in Section "*Penalty parameter*" later on. ⌟

## 2.2 Linear systems

In order to compute the SIPG approximation defined by (2), it needs to be rewritten as a linear system. To this end, we choose basis functions for the test space $V$. More specifically, for each mesh element $E_i$, we define the basis function $\phi_1^{(i)}$ which is zero in the entire domain, except in $E_i$, where it is equal to one. Similarly, we define higher-order basis functions $\phi_2^{(i)}, ..., \phi_m^{(i)}$, which are higher-order polynomials in $E_i$ and zero elsewhere. In this report, we use monomial basis functions (cf. details below).

Details (Monomial basis functions) For uniform Cartesian meshes on the domain $\Omega = [0,1]^2$, the monomial basis functions are defined as follows. In the mesh element $E_i$ with center $(x_i, y_i)$ and size $h \times h$, the function $\phi_k^{(i)}$ reads:

$$\phi_k^{(i)}(x,y) = \left(\frac{x - x_i}{\frac{1}{2}h}\right)^{k_x} \left(\frac{y - y_i}{\frac{1}{2}h}\right)^{k_y},$$

where $k_x$ and $k_y$ are selected as follows:

| $k$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $k_x$ | 0 | 1 | 0 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | ... |
| $k_y$ | 0 | 0 | 1 | 0 | 1 | 2 | 0 | 1 | 2 | 3 | ... |
| | $p = 0$ | $p = 1$ | | $p = 2$ | | | $p = 3$ | | | | ... |

The dimension of the basis within one mesh element is equal to $m = \frac{(p+1)(p+2)}{2}$ for two-dimensional problems (for one-dimensional problems, we would have $m = p + 1$). ⌟

Next, we express $u_h$ as a linear combination of the basis functions:

$$u_h = \sum_{i=1}^{N} \sum_{k=1}^{m} u_k^{(i)} \phi_k^{(i)}. \tag{3}$$

The new unknowns $u_k^{(i)}$ in (3) can be determined by solving a linear system $A\mathbf{u} = \mathbf{b}$ of the form:

$$\begin{bmatrix} A_{11} & A_{12} & \dots & A_{1N} \\ A_{21} & A_{22} & & \vdots \\ \vdots & & \ddots & \\ A_{N1} & \dots & & A_{NN} \end{bmatrix} \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \vdots \\ \mathbf{u}_N \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_N \end{bmatrix},$$

where the blocks all have dimension $m$:

$$A_{ji} = \begin{bmatrix} B(\phi_1^{(i)}, \phi_1^{(j)}) & B(\phi_2^{(i)}, \phi_1^{(j)}) & \dots & B(\phi_m^{(i)}, \phi_1^{(j)}) \\ B(\phi_1^{(i)}, \phi_2^{(j)}) & B(\phi_2^{(i)}, \phi_2^{(j)}) & & \vdots \\ \vdots & & \ddots & \\ B(\phi_1^{(i)}, \phi_m^{(j)}) & \dots & & B(\phi_m^{(i)}, \phi_m^{(j)}) \end{bmatrix}, \quad \mathbf{u}_i = \begin{bmatrix} u_1^{(i)} \\ u_2^{(i)} \\ \vdots \\ u_m^{(i)} \end{bmatrix}, \quad \mathbf{b}_j = \begin{bmatrix} L(\phi_1^{(j)}) \\ L(\phi_2^{(j)}) \\ \vdots \\ L(\phi_m^{(j)}) \end{bmatrix},$$

for all $i, j = 1, ..., N$. This system is obtained by substituting the expression (3) for $u_h$ and the basis functions $\phi_\ell^{(j)}$ for $v$ into (2). Once the unknowns $u_k^{(i)}$ are solved from the system $A\mathbf{u} = \mathbf{b}$, the final SIPG approximation $u_h$ can be obtained from (3).

Example (SIPG matrix) For a Laplace problem on the domain $[0,1]^2$ with $p = 1$, a uniform Cartesian mesh with $2 \times 2$ elements, and penalty parameter $\sigma = 10$ we obtain the following SIPG matrix $A$:

$$A = \left[\begin{array}{rrr|rrr||rrr|rrr} 40 & 1 & 1 & -10 & 9 & 0 & -10 & 0 & 9 & 0 & 0 & 0 \\ 1 & 25 & 0 & -9 & 8 & 0 & 0 & -3 & 0 & 0 & 0 & 0 \\ 1 & 0 & 25 & 0 & 0 & -3 & -9 & 0 & 8 & 0 & 0 & 0 \\ \hline -10 & -9 & 0 & 40 & -1 & 1 & 0 & 0 & 0 & -10 & 0 & 9 \\ 9 & 8 & 0 & -1 & 25 & 0 & 0 & 0 & 0 & 0 & -3 & 0 \\ 0 & 0 & -3 & 1 & 0 & 25 & 0 & 0 & 0 & -9 & 0 & 8 \\ \hline\hline -10 & 0 & -9 & 0 & 0 & 0 & 40 & 1 & -1 & -10 & 9 & 0 \\ 0 & -3 & 0 & 0 & 0 & 0 & 1 & 25 & 0 & -9 & 8 & 0 \\ 9 & 0 & 8 & 0 & 0 & 0 & -1 & 0 & 25 & 0 & 0 & -3 \\ \hline 0 & 0 & 0 & -10 & 0 & -9 & -10 & -9 & 0 & 40 & -1 & -1 \\ 0 & 0 & 0 & 0 & -3 & 0 & 9 & 8 & 0 & -1 & 25 & 0 \\ 0 & 0 & 0 & 9 & 0 & 8 & 0 & 0 & -3 & -1 & 0 & 25 \end{array}\right]. \tag{4}$$

Note that $A$ has the same structure as a central difference matrix, except that it consists of small dense blocks rather than scalars. ⌟

## 2.3 Penalty parameter

The SIPG method involves the penalty parameter $\sigma$ which penalizes the inter-element jumps to enforce weak continuity (cf. Section "*SIPG method*"). This parameter should be selected carefully: on the one hand, it needs to be sufficiently large to ensure that the SIPG method converges and the coefficient matrix $A$ is Symmetric and Positive Definite (SPD) (Rivière (2008)). At the same time, it needs to be chosen as small as possible, since the condition number of $A$ increases rapidly with the penalty parameter (Castillo (2002)).

Computable theoretical lower bounds have been derived for a large variety of problems by Epshteyn and Rivière (2007). For one-dimensional diffusion problems, it suffices to choose $\sigma \geq 2\frac{k_1^2}{k_0}p^2$, where $k_0$ and $k_1$ are the *global* lower and upper bound respectively for the diffusion coefficient $K$. However, while this lower bound for $\sigma$ is sufficient to ensure convergence (assuming the exact solution is sufficiently smooth), it can be unpractical for problems with strong variations in the coefficients. For instance, if the diffusion coefficient $K$ takes values between 1 and $10^{-3}$, we obtain $\sigma \geq 2000p^2$, which is inconveniently large. For this reason, it is common practice to choose e.g. $\sigma = 20$ rather than $\sigma = 20\,000$ for such problems (Dobrev et al. (2006); Proft and Rivière (2009)).

An alternative strategy is to choose the penalty parameter based on *local* values of the diffusion-coefficient $K$, e.g. choosing $\sigma = 20K$ rather than $\sigma = 20$. Such a strategy has been analyzed theoretically by Dryja (2003) for piecewise linear continuous basis functions in the context of Schwarz domain decomposition. Furthermore, it has been demonstrated numerically by Dobrev et al. (2008) that a diffusion-dependent penalty parameter can benefit the efficiency of a linear solver. This will be discussed further in Section "*Numerical experiments*" later on.

In this report, we study the effects of both a constant and a diffusion-dependent penalty parameter.

# 3 Linear solver

The SIPG discretization discussed in the previous section requires the solution of a linear system of the form $A\mathbf{u} = \mathbf{b}$, where $A$ is SPD. This section discusses preconditioning strategies for solving this system by means of the CG method. The main idea is to make use of coarse corrections based on the lower order SIPG method with polynomial degree $p = 0$. This concept can be incorporated in a two-level preconditioner, as proposed by Dobrev et al. (2006), but also in a deflation variant, using the abstract analysis of Tang et al. (2009). This section discusses both strategies.

## 3.1 Coarse correction operator $Q$

Both the two-level preconditioner and the corresponding deflation variant are defined in terms of a coarse correction operator $Q \approx A^{-1}$ that switches from the original DG test space to a coarse subspace, then performs a correction that is now simple in this coarse space, and finally switches back to the original DG test space. In this report, we study the coarse subspace that consists of all piecewise constant functions.

More specifically, the coarse correction operator $Q$ is defined as follows: let $A_0$ denote the SIPG matrix for polynomial degree $p = 0$. This matrix is also referred to as the 'coarse' matrix. Next, define the restriction operator $R$ such that $A_0 := RAR^T$. More specifically, the matrix $R$ is defined as the following $N \times N$ block matrix:

$$R = \begin{bmatrix} R_{11} & R_{12} & \dots & R_{1N} \\ R_{21} & R_{22} & & \vdots \\ \vdots & & \ddots & \\ R_{N1} & \dots & & R_{NN} \end{bmatrix},$$

where the blocks all have size $1 \times m$:

$$R_{ii} = \begin{bmatrix} 1 & 0 \dots & 0 \end{bmatrix}, \qquad\qquad R_{ij} = \begin{bmatrix} 0 & \dots & 0 \end{bmatrix},$$

for all $i, j = 1, ..., N$ and $i \neq j$.

Example (Coarse matrix) For the matrix $A$ in (4), we obtain the following coarse matrix $A_0$ and restriction operator $R$:

$$A_0 = \left[\begin{array}{r|r||r|r} 40 & -10 & -10 & 0 \\ \hline -10 & 40 & 0 & -10 \\ \hline\hline -10 & 0 & 40 & -10 \\ \hline 0 & -10 & -10 & 40 \end{array}\right], \quad R = \left[\begin{array}{ccc|ccc||ccc|ccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline\hline 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{array}\right]$$

Observe that $A_0$ has the same structure as a central difference matrix (aside from a factor $\sigma = 10$). Furthermore, every element in $A_0$ is also present in the upper left corner of the corresponding block in the matrix $A$. This is because the piecewise constant basis functions are assumed to be in any polynomial basis. As a consequence, the matrix $R$ contains elements equal to 0 and 1 only, and does not need to be stored explicitly: multiplications with $R$ can be implemented by simply extracting elements or inserting zeros. ⌟

Using this notation, the coarse correction operator $Q$ is defined as follows:

$$Q := \underbrace{R^T}_{\text{prolongation}} \quad \underbrace{A_0^{-1}}_{\text{coarse correction}} \quad \underbrace{R}_{\text{restriction}} .$$

This operator forms the basis of both the two-level preconditioner and the corresponding deflation variant, as is discussed hereafter.

## 3.2 Two-level preconditioner

Now that the coarse correction operator $Q$ has been defined, we can formulate the two-level preconditioner. To this end, let $\omega \leq 1$ denote a damping parameter (damping will be discussed further in Section "*Damping*" later on), and let $M^{-1} \approx A^{-1}$ denote an invertible smoother, such as block Jacobi. Basically, the result $\mathbf{y} = P_{\text{prec}}\mathbf{r}$ of applying this preconditioner to a vector $\mathbf{r}$ can be computed in three steps:

$$\begin{aligned} \mathbf{y}^{(1)} &= \omega M^{-1}\mathbf{r} && \text{(pre-smoothing)}, \\ \mathbf{y}^{(2)} &= \mathbf{y}^{(1)} + Q(\mathbf{r} - A\mathbf{y}^{(1)}) && \text{(coarse correction)}, \\ \mathbf{y} &= \mathbf{y}^{(2)} + \omega M^{-T}(\mathbf{r} - A\mathbf{y}^{(2)}) && \text{(post-smoothing)}. \end{aligned} \tag{5}$$

If the smoother is chosen such that

$$M + M^T - \omega A \text{ is SPD,} \tag{6}$$

then the operator $P_{\text{prec}}$ is SPD (Vassilevski (2008)). As a consequence, the two-level preconditioner can be implemented in a standard preconditioned CG algorithm (see details below). Requirement (6) also implies that the two-level preconditioner yields scalable convergence of the CG method (independent of the mesh element diameter) for a large class of problems. This has been shown for polynomial degree $p = 1$ by Dobrev et al. (2006), using the analysis of Falgout et al. (2005).

Details (CG implementation) For a given preconditioning operator $P$ and start vector $x_0$, the CG method can be implemented as follows:

```
1.  $\mathbf{r}_0 := \mathbf{b} - A\mathbf{x}_0$
2.  $\mathbf{y}_0 := P\mathbf{r}_0$
3.  $\mathbf{p}_0 := \mathbf{y}_0$
4.  for $j = 0, 1, ...$ until convergence do
5.      $\mathbf{w}_j := A\mathbf{p}_j$
6.      $\alpha_j := (\mathbf{r}_j, \mathbf{y}_j)/(\mathbf{p}_j, \mathbf{w}_j)$
7.      $\mathbf{x}_{j+1} := \mathbf{x}_j + \alpha_j \mathbf{p}_j$
8.      $\mathbf{r}_{j+1} := \mathbf{r}_j - \alpha_j \mathbf{w}_j$
9.      $\mathbf{y}_{j+1} = P\mathbf{r}_{j+1}$
10.     $\beta_j := (\mathbf{r}_{j+1}, \mathbf{y}_{j+1})/(\mathbf{r}_j, \mathbf{y}_j)$
11.     $\mathbf{p}_{j+1} := \mathbf{y}_{j+1} + \beta_j \mathbf{p}_j$
12. end
```
⌟

## 3.3 Two-level deflation

The two-level preconditioner can be reformulated as a deflation method. To this end, there are several strategies, leading to different types of deflation (Tang et al. (2009)). In this report, we study the so-called 'ADEF2' deflation scheme, as this type can be implemented relatively efficiently, and allows for inexact solving of coarse systems. The latter will be discussed further in Section "*Numerical experiments*" later on.

Basically, the ADEF2 deflation variant is obtained by skipping the last smoothing step in (5). In other words, the result $\mathbf{y} = P_{\text{defl}}\mathbf{r}$ of applying the two-level deflation technique to a vector $\mathbf{r}$ can be computed as:

$$\begin{aligned} \mathbf{y}^{(1)} &:= \omega M^{-1}\mathbf{r} && \text{(pre-smoothing)}, \\ \mathbf{y} &:= \mathbf{y}^{(1)} + Q(\mathbf{r} - A\mathbf{y}^{(1)}) && \text{(coarse correction)}. \end{aligned} \tag{7}$$

The operator $P_{\text{defl}}$ is not symmetric in general. As such, it seems unsuitable for the standard preconditioned CG method. Interestingly, it can still be implemented successfully in its current asymmetric form, as long as the smoother $M^{-1}$ is SPD (requirement (6) is not needed), and the start vector $\mathbf{x}_0$ is pre-processed according to:

$$\mathbf{x}_0 \mapsto Q\mathbf{b} + (I - AQ)^T\mathbf{x}_0. \tag{8}$$

Other than that, the CG implementation remains as discussed in Section "*Two-level preconditioner*". Indeed, it has been shown by Tang et al. (2009) that, under the conditions above, $P_{\text{prec}}$ yields the same CG iterates as an alternative operator (called 'BNN') that actually *is* SPD.

## 3.4 Comparison of computational costs

Because the deflation variant skips one of the two smoothing steps, its costs per CG iteration are lower than for the preconditioning variant. In this section, we compare the differences in terms of FLoating point OPerationS (FLOPS).

Table 1 displays the costs for a two-dimensional diffusion problem with polynomial degree $p$, a Cartesian mesh with $N = n \times n$ elements, and polynomial space dimension $m := \frac{(p+1)(p+2)}{2}$ (cf. Section "*SIPG method*"). Using the preconditioning variant, the CG method requires per iteration $(27m^2 + 14m)N$ flops, plus the costs for two smoothing steps and one coarse solve. Using the two-level deflation method, the CG method requires per iteration $(18m^2 + 12m)N$ flops, plus the costs for only one smoothing step and one coarse solve.

| operation | flops (rounded) | # defl. | # prec. |
|---|---|---|---|
| mat-vec ($Au$) | $9m^2N$ | 2 | 3 |
| inner product ($u^Tv$) | $2mN$ | 2 | 2 |
| scalar multiplication ($\alpha u$) | $mN$ | 3 | 3 |
| vector update ($u \pm v$) | $mN$ | 5 | 7 |
| smoothing ($M^{-1}u$) | variable | 1 | 2 |
| coarse solve ($A_0^{-1}u_0$) | variable | 1 | 1 |

Table 1: Comparing the computational costs per CG iteration for A-DEF2 deflation and the two-level preconditioner for our applications.

A block Jacobi smoothing step with blocks of size $m$ requires $(2m^2 - m)N$ flops, assuming that an $LU$-decomposition is known. In this case, the smoothing costs are low compared to the costs for a matrix-vector product, and the deflation variant is roughly 30% faster (per iteration). For more expensive smoothers, this factor becomes larger. A block Gauss-Seidel sweep (either forward or backward) requires the costs for one block Jacobi step, plus the costs for the updates based on the off-diagonal elements, which are approximately $4m^2N$ flops.

### 3.5 Damping

Damping often benefits the convergence of multigrid methods (Yavneh (2006)). At the same time, deflation may not be influenced by damping at all. The latter has been observed theoretically by Tang et al. (2010) for the so-called 'DEF' variant (recall that we study the alternative ADEF2 variant).

For multigrid methods with smoother $M = I$, a "typical choice of $[\omega]$ is close to $\frac{1}{||A||_2}$", although a "better choice of $[\omega]$ is possible if we make further assumptions on how the eigenvectors of A associated with small eigenvalues are treated by coarse-grid correction" —— Tang et al. (2010). In that reference, the latter is established for a coarse space that is based on a set of orthonormal eigenvectors of $A$. However, such a result does not seem available yet for the coarse space (and smoothers) currently under consideration.

Altogether, it is an open question how the damping parameter can best be selected in practice. For this reason, we use an emprical approach in this report, and study the effects for several values of $\omega \leq 1$.

## 4 Numerical experiments

In this section, we compare the two-level preconditioner and the corresponding deflation variant discussed in the previous section through numerical experiments. In particular, we study the scalability of both methods, and the influence of the SIPG penalty parameter, the smoother, damping of the smoother, and the strategy for solving the coarse systems.

### 4.1 Experimental setup

We consider four diffusion problems of the form (1) on the domain $[0, 1]^2$: a Poisson problem ($K = 1$) and the three test cases specified in Figure 1 (if we subdivide the domain into $10 \times 10$ equally sized squares, the diffusion coefficient is constant within each square). The test cases in Figure 1 are largely inspired by Vuik et al. (2001), and mimic the presence of layers of sandstone and shale, the occurrence of sand inclusions within a layer of shale, and ground water flow. In all cases, the Dirichlet boundary conditions and the source term $f$ are chosen such that the exact solution reads $u(x, y) = \cos(10\pi x) \cos(10\pi y)$. We stress that this choice does not impact the matrix or the performance of the linear solver, as we use random start vectors (see below).

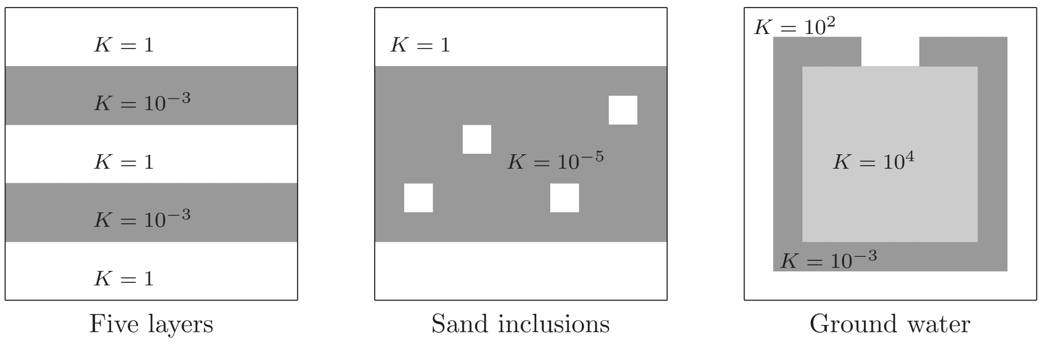

Figure 1: Illustration of the three test cases

All model problems are discretized by means of the SIPG method as discussed in Section "*Discretization*". We use a uniform Cartesian mesh with $N = n \times n$ elements, where $n = 20, 40, 80, 160, 320$. Furthermore, we use monomial basis functions with polynomial degree $p = 2, 3$ (results for $p = 1$ are similar though). Unless specified otherwise, the penalty parameter is chosen diffusion-dependent, $\sigma = 20K$. The factor 20 has been chosen emperically. We have verified that the matrix $A$ is SPD and that the SIPG method converges for this particular choice. For $p = 3$ and $n = 320$, the number of degrees of freedom is a little over $10^6$.

The linear systems resulting from the SIPG discretizations are solved by means of the CG method, combined with either the two-level preconditioner or the corresponding deflation variant, as discussed in Section "*Linear solver*". Unless specified otherwise, damping is not used. For the smoother $M^{-1}$, we use block Jacobi with small blocks of size $m \times m$ (recall that $m = \frac{(p+1)(p+2)}{2}$). For the preconditioning variant, we also consider block Gauss-Seidel with the same block size (deflation requires a symmetric smoother). It can be shown that the aforementioned smoothers satisfy (6) for all $\omega \leq 1$ for the problems under consideration.

Diagonal scaling is applied as a pre-processing step in all cases, and the same random start vector $\mathbf{x}_0$ is used for all problems of the same size. Pre-processing of the start vector according to (8) is applied for deflation only, as it makes no difference for the preconditioning variant. For the stopping criterion we use:

$$\frac{\|r_k\|_2}{\|b\|_2} \leq \text{TOL}, \tag{9}$$

where $\text{TOL} = 10^{-6}$, and $r_k$ is the residual after the $k^{\text{th}}$ iteration.

Coarse systems, involving the SIPG matrix $A_0$ with polynomial degree $p = 0$, are solved directly. However, a more efficient strategy is also provided and tested in Section "*Coarse systems*" later on. In any case, the coarse matrix $A_0$ is quite similar to a central difference matrix, for which very efficient solvers are readily available.

## 4.2 Diffusion-dependent penalty parameter

This section studies the influence of the SIPG penalty parameter on the convergence of the CG and the SIPG method. We compare the differences between using a constant penalty parameter ($\sigma = 20$), and a diffusion-dependent value ($\sigma = 20K$). Similar experiments have been considered by Dobrev et al. (2008) for the two-level preconditioner for $p = 1$, a single mesh, and symmetric Gauss-Seidel smoothing (solving the coarse systems using geometric multigrid). They found that "proper weighting", i.e. a diffusion-dependent penalty parameter, "is essential for the performance". In this section, we consider $p = 2, 3$, and both preconditioning and deflation (using block Jacobi smoothing). Furthermore, we analyze the scalability of the methods by considering multiple meshes. Our results are consistent with those of Dobrev et al. (2008).

Table 2 displays the number of CG iterations required for convergence for the Poisson problem with $\sigma = 20$. Because the diffusion coefficient is constant ($K = 1$), a diffusion-dependent value ($\sigma = 20K$) would yield the same results. We observe that both the two-level preconditioner (TL prec.) and the deflation variant (TL defl.) yield fast and scalable convergence (independent of the mesh element diameter). For comparison, the results for standard Jacobi and block Jacobi preconditioning are also displayed (not scalable). Interestingly, the two-level deflation method requires fewer iterations than the preconditioning variant, even though its costs per iteration are lower (cf. Section "*Comparison of computational costs*").

| degree | p=2 | | | | p=3 | | | |
|---:|:-:|:-:|:-:|:-:|:-:|:-:|:-:|:-:|
| mesh | N=$20^2$ | N=$40^2$ | N=$80^2$ | N=$160^2$ | N=$20^2$ | N=$40^2$ | N=$80^2$ | N=$160^2$ |
| Jacobi | 301 | 581 | 1049 | 1644 | 325 | 576 | 1114 | 1903 |
| Block Jacobi (BJ) | 205 | 356 | 676 | 1190 | 206 | 357 | 696 | 1183 |
| TL Prec., 2x BJ | 36 | 38 | 39 | 40 | 49 | 52 | 53 | 54 |
| TL Defl., 1x BJ | 32 | 33 | 33 | 34 | 36 | 37 | 37 | 38 |

Table 2: Poisson: # CG iterations for a constant penalty $\sigma = 20$

Table 3 considers the same test, but now for the problem with five layers (using a constant $\sigma = 20$). It can be seen that the convergence is no longer fast and scalable for this problem with jumps in the coefficients. The deflation method is significantly faster than the preconditioning variant, but neither produce satisfactory results.

Table 4 addresses this issue by switching to a diffusion-dependent penalty parameter ($\sigma = 20K$). Now, the results are similar to those for the Poisson problem, and both two-level methods yield fast and scalable convergence.

| degree | p=2 | | | | p=3 | | | |
|---|---|---|---|---|---|---|---|---|
| mesh | N=$20^2$ | N=$40^2$ | N=$80^2$ | N=$160^2$ | N=$20^2$ | N=$40^2$ | N=$80^2$ | N=$160^2$ |
| Jacobi | 1671 | 4311 | 9069 | 15924 | 2569 | 5070 | 9083 | 15656 |
| Block Jacobi (BJ) | 933 | 2253 | 4996 | 9656 | 1398 | 2960 | 5660 | 9783 |
| TL Prec., 2x BJ | 415 | 1215 | 2534 | 3571 | 1089 | 2352 | 4709 | 8781 |
| TL Defl., 1x BJ | 200 | 414 | 531 | 599 | 453 | 591 | 667 | 698 |

Table 3: Five layers: # CG iterations for a constant penalty $\sigma = 20$

| degree | p=2 | | | | p=3 | | | |
|---|---|---|---|---|---|---|---|---|
| mesh | N=$20^2$ | N=$40^2$ | N=$80^2$ | N=$160^2$ | N=$20^2$ | N=$40^2$ | N=$80^2$ | N=$160^2$ |
| Jacobi | 975 | 1264 | 1567 | 2314 | 1295 | 1490 | 1921 | 3110 |
| Block Jacobi (BJ) | 243 | 424 | 788 | 1285 | 244 | 425 | 697 | 1485 |
| TL Prec., 2x BJ | 46 | 43 | 43 | 44 | 55 | 56 | 56 | 57 |
| TL Defl., 1x BJ | 43 | 45 | 45 | 46 | 47 | 48 | 48 | 48 |

Table 4: Five layers: # CG iterations for a diffusion-dependent penalty $\sigma = 20K$

These results motivate the use of a diffusion-dependent penalty parameter, provided that that this strategy does not worsen the accuracy of the SIPG discretization compared to a constant penalty parameter. In Figure 2, it is verified that a diffusion-dependent penalty parameter actually improves the accuracy of the SIPG approximation (for $p = 3$). Similar results have been observed for other test cases and for $p = 1, 2$ (not displayed). The higher accuracy can be explained by the fact that the discretization contains more information of the underlying physics for a diffusion-dependent penalty parameter. Altogether, the penalty parameter can best be chosen diffusion-dependent, and we will do so in the remaining of this report.

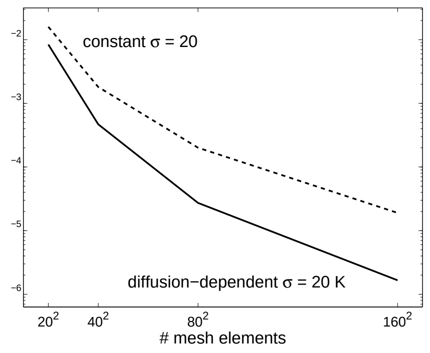

Figure 2: Five layers (p=3): SIPG accuracy

## 4.3 Smoothers & Damping

This section discusses the influence of the smoother and damping on both two-level methods. In particular, we consider multiple damping values $\omega = 0.5, 0.6, 0.7, 0.8, 0.9, 1$, and both Jacobi and (block) Gauss-Seidel smoothing. The latter is applied for the preconditioning variant only, as deflation requires a *symmetric* smoother.

Table 5 displays the number of CG iterations required for convergence for the problem with five layers. For the deflation variant (Defl.), we have found that damping makes no difference for the CG convergence, so the outcomes for $\omega < 1$ are not displayed. Such a result has also been observed theoretically by Tang et al. (2010) for an alternative deflation variant (known as 'DEF'). For the preconditioning variant (Prec.), damping can both improve and worsen the efficiency: for block Jacobi (BJ) smoothing, choosing e.g. $\omega = 0.7$ can reduce the number of iterations by 37%; for block Gauss-Seidel (BGS) smoothing, choosing $\omega < 1$ has either no influence or a small negative impact in most cases. We have also performed the experiment for standard (non-block) Gauss-Seidel (not displayed in the table). However, this did not lead to satisfactory results: over

250 iterations in all cases. We speculate that this is due to the fact that the block structure is not well-presented in that case.

| degree | p=2 | | | | p=3 | | | |
|---|---|---|---|---|---|---|---|---|
| mesh | N=$40^2$ | N=$80^2$ | N=$160^2$ | N=$320^2$ | N=$40^2$ | N=$80^2$ | N=$160^2$ | N=$320^2$ |
| Prec., 2X BJ ($\omega = 1$) | 43 | 43 | 44 | 44 | 56 | 56 | 57 | 58 |
| ($\omega = 0.9$) | 34 | 34 | 37 | 37 | 40 | 40 | 42 | 43 |
| ($\omega = 0.8$) | 33 | 34 | 34 | 35 | 36 | 37 | 39 | 39 |
| ($\omega = 0.7$) | 33 | 33 | 33 | 34 | 35 | 36 | 36 | 37 |
| ($\omega = 0.6$) | 32 | 33 | 34 | 35 | 35 | 36 | 36 | 37 |
| ($\omega = 0.5$) | 34 | 34 | 35 | 35 | 36 | 37 | 38 | 39 |
| Prec., 2x BGS ($\omega = 1$) | 33 | 33 | 34 | 35 | 34 | 35 | 35 | 37 |
| ($\omega = 0.9$) | 33 | 33 | 34 | 34 | 34 | 34 | 35 | 36 |
| ($\omega = 0.8$) | 33 | 33 | 35 | 35 | 35 | 34 | 36 | 37 |
| ($\omega = 0.7$) | 32 | 34 | 36 | 36 | 35 | 36 | 37 | 38 |
| ($\omega = 0.6$) | 33 | 35 | 36 | 37 | 36 | 37 | 38 | 39 |
| ($\omega = 0.5$) | 34 | 35 | 37 | 38 | 37 | 39 | 39 | 40 |
| Defl., 1x BJ ($\omega = 1$) | 45 | 45 | 46 | 46 | 48 | 48 | 48 | 49 |

Table 5: Five layers: # CG Iterations

Based on the results in Table 5, it appears that that the preconditioning variant with either block Jacobi (with optimal damping) or block Gauss-Seidel is the most efficient choice. However, the costs per iteration also need to be taken into account. As expected, we have found that block Gauss-Seidel yields lower overall efficiency than block Jacobi (when the number of iterations is approximately the same).

Figure 3 compares the overall computational time in seconds for both two-level schemes with block Jacobi smoothing (with and without damping for the preconditioner). Without damping, the deflation variant turns out to be the fastest. This is due to the fact that it requires only one smoothing step per iteration, instead of two. When an optimal damping parameter is known, the preconditioning variant performs comparable to deflation. However, it is an open question how the damping parameter can best be selected in practice (cf. Section "*Damping*").

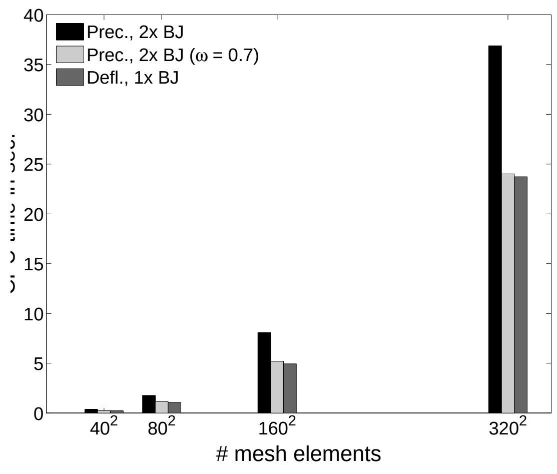

Figure 3: Five layers: CPU time in seconds

In Table 6 and Table 8, the experiment is repeated for the problems 'sand inclusions' and 'ground water' respectively (cf. Figure 1). Only the best candidates are displayed (regarding damping, the results are similar to those for 'five layers'). The corresponding computational times are displayed in Table 7 and Table 9. As before, both two-level methods yield fast and scalable convergence. Without damping, deflation is the most efficient. When an optimal damping value is known, the preconditioning variant performs comparable to deflation.

### 4.4 Coarse systems

To obtain the results in the previous sections, a direct solver has been used for the coarse systems with coefficient matrix $A_0$. In practice, this is usually not feasible, since $A_0$ has the same structure and size ($N \times N$) as a central difference matrix. To improve on the efficiency of the coarse solver, we have investigated the cheaper alternative of applying the CG method again in an inner loop. This section discusses the results using the algebraic multigrid preconditioner *MI_20* in the HSL software package[1]. The inner loop uses a stopping criterion of the form (9).

Table 10 and Table 11 display the number of outer CG iterations required for convergence using the two-level preconditioner and deflation variant respectively (for the problem with five layers). Different values of the inner tolerance TOL are considered in these tables. For comparison, the results for the direct solver are also displayed. We observe that low accuracy in the inner loop is sufficient to reproduce the latter. In both tables, the inner tolerance can be $10^4$ times as large as the outer tolerance. For the highest acceptable inner tolerance TOL $= 10^{-2}$, the number of iterations in the inner loop is between 2 and 5 in all cases (not displayed in the tables).

We remark that, in terms of computational time, the difference between the direct solver and the inexact AMG-CG solver is negligible for the problems under consideration. However, for large three-dimensional problems, it can be expected that the inexact coarse solver is much faster, and thus crucial for the overall efficiency of the linear solver.

## 5 Conclusion

This report compares the two-level preconditioner proposed by Dobrev et al. (2006) and the resulting ADEF2-deflation variant for linear systems resulting from SIPG discretizations. We have found that both two-level methods yield fast and scalable convergence for diffusion-problems with large jumps in the coefficients. This result is obtained provided that the SIPG penalty parameter is chosen diffusion-dependent. Coarse systems can be solved efficiently by applying the CG method again in an inner loop with low accuracy.

Whether preconditioning or deflation is to be favored depends on the choice for the smoother and on the damping of the smoother. Without damping, we have found that deflation (with block Jacobi smoothing) yields faster convergence of the CG method at lower costs per iteration. When an optimal damping factor is known, the preconditioner (with block Jacobi smoothing) yields comparable computational times. However, it remains an open question how the damping parameter can best be selected in practice.

Altogether, both two-level methods can contribute to faster and more accurate pressure simulations for layered systems with strong permeability contrasts. Future research could focus on theoretical support for these findings and more advanced, larger-scale test cases.

## References

Antonietti, P.F. and Ayuso, B. [2007] Schwarz domain decomposition preconditioners for discontinuous Galerkin approximations of elliptic problems: non-overlapping case. *M2AN Math. Model. Numer. Anal.*, **41**(1), 21–54.

Arnold, D.N., Brezzi, F., Cockburn, B. and Marini, L.D. [2002] Unified analysis of discontinuous Galerkin methods for elliptic problems. *SIAM J. Numer. Anal.*, **39**(5), 1749–1779 (electronic).

Brenner, S.C. and Zhao, J. [2005] Convergence of multigrid algorithms for interior penalty methods. *Appl. Numer. Anal. Comput. Math.*, **2**(1), 3–18.

Castillo, P. [2002] Performance of discontinuous Galerkin methods for elliptic PDEs. *SIAM J. Sci. Comput.*, **24**(2), 524–547.

[1]HSL, a collection of Fortran codes for large-scale scientific computation. See http://www.hsl.rl.ac.uk/

| degree | p=2 | | | | p=3 | | | |
|---|---|---|---|---|---|---|---|---|
| mesh | N=$40^2$ | N=$80^2$ | N=$160^2$ | N=$320^2$ | N=$40^2$ | N=$80^2$ | N=$160^2$ | N=$320^2$ |
| Prec., 2x BJ | 44 | 48 | 46 | 46 | 53 | 56 | 58 | 59 |
| Prec., 2x BJ ($\omega = 0.7$) | 30 | 30 | 30 | 31 | 32 | 34 | 34 | 34 |
| Defl., 1x BJ | 42 | 42 | 42 | 43 | 46 | 47 | 48 | 48 |

Table 6: Sand Inclusions: # CG Iterations

| degree | p=2 | | | | p=3 | | | |
|---|---|---|---|---|---|---|---|---|
| mesh | N=$40^2$ | N=$80^2$ | N=$160^2$ | N=$320^2$ | N=$40^2$ | N=$80^2$ | N=$160^2$ | N=$320^2$ |
| Prec., 2x BJ | 0.10 | 0.69 | 3.35 | 16.85 | 0.37 | 1.77 | 8.22 | 38.06 |
| Prec., 2x BJ ($\omega = 0.7$) | 0.07 | 0.44 | 2.19 | 11.47 | 0.22 | 1.08 | 4.86 | 22.16 |
| Defl., 1x BJ | 0.07 | 0.47 | 2.44 | 13.23 | 0.22 | 1.06 | 4.94 | 23.76 |

Table 7: Sand Inclusions: CPU time in seconds

| degree | p=2 | | | | p=3 | | | |
|---|---|---|---|---|---|---|---|---|
| mesh | N=$40^2$ | N=$80^2$ | N=$160^2$ | N=$320^2$ | N=$40^2$ | N=$80^2$ | N=$160^2$ | N=$320^2$ |
| Prec., 2x BJ | 54 | 52 | 52 | 52 | 67 | 68 | 68 | 69 |
| Prec., 2x BJ ($\omega = 0.7$) | 38 | 38 | 38 | 40 | 41 | 42 | 42 | 42 |
| Defl., 1x BJ | 54 | 54 | 54 | 55 | 59 | 59 | 60 | 60 |

Table 8: Ground water: # CG Iterations

| degree | p=2 | | | | p=3 | | | |
|---|---|---|---|---|---|---|---|---|
| mesh | N=$40^2$ | N=$80^2$ | N=$160^2$ | N=$320^2$ | N=$40^2$ | N=$80^2$ | N=$160^2$ | N=$320^2$ |
| Prec., 2x BJ | 0.12 | 0.75 | 3.75 | 18.57 | 0.46 | 2.15 | 9.53 | 43.71 |
| Prec., 2x BJ ($\omega = 0.7$) | 0.08 | 0.56 | 2.77 | 14.53 | 0.28 | 1.35 | 5.88 | 27.15 |
| Defl., 1x BJ | 0.08 | 0.61 | 3.19 | 16.57 | 0.28 | 1.32 | 6.09 | 29.01 |

Table 9: Ground water: CPU time in seconds

| degree | p=2 | | | | p=3 | | | |
|---|---|---|---|---|---|---|---|---|
| mesh | N=$40^2$ | N=$80^2$ | N=$160^2$ | N=$320^2$ | N=$40^2$ | N=$80^2$ | N=$160^2$ | N=$320^2$ |
| direct | 43 | 43 | 44 | 44 | 56 | 56 | 57 | 58 |
| TOL = $10^{-4}$ | 43 | 43 | 44 | 44 | 56 | 56 | 57 | 58 |
| TOL = $10^{-3}$ | 43 | 43 | 44 | 44 | 56 | 56 | 57 | 58 |
| TOL = $10^{-2}$ | 43 | 43 | 44 | 44 | 56 | 57 | 58 | 58 |
| TOL = $10^{-1}$ | 48 | 62 | 55 | 57 | 59 | 65 | 70 | 78 |

Table 10: Five layers: # outer iterations for the preconditioning variant

| degree | p=2 | | | | p=3 | | | |
|---|---|---|---|---|---|---|---|---|
| mesh | N=$40^2$ | N=$80^2$ | N=$160^2$ | N=$320^2$ | N=$40^2$ | N=$80^2$ | N=$160^2$ | N=$320^2$ |
| direct | 45 | 45 | 46 | 46 | 48 | 48 | 48 | 49 |
| TOL = $10^{-4}$ | 45 | 45 | 46 | 46 | 48 | 48 | 48 | 49 |
| TOL = $10^{-3}$ | 45 | 45 | 46 | 46 | 48 | 48 | 48 | 49 |
| TOL = $10^{-2}$ | 45 | 45 | 46 | 46 | 48 | 48 | 48 | 49 |
| TOL = $10^{-1}$ | 60 | 81 | 51 | 300 | 48 | 54 | 79 | 67 |

Table 11: Five layers: # outer iterations for the deflation variant

Dobrev, V.A., Lazarov, R.D., Vassilevski, P.S. and Zikatanov, L.T. [2006] Two-level preconditioning of discontinuous Galerkin approximations of second-order elliptic equations. *Numer. Linear Algebra Appl.*, **13**(9), 753–770.

Dobrev, V.A., Lazarov, R.D. and Zikatanov, L.T. [2008] Preconditioning of symmetric interior penalty discontinuous Galerkin FEM for elliptic problems. In: *Domain decomposition methods in science and engineering XVII.* Springer, Berlin, vol. 60 of *Lect. Notes Comput. Sci. Eng.*, 33–44.

Dryja, M. [2003] On discontinuous Galerkin methods for elliptic problems with discontinuous coefficients. *Comput. Methods Appl. Math.*, **3**(1), 76–85 (electronic), dedicated to Raytcho Lazarov.

Epshteyn, Y. and Rivière, B. [2007] Estimation of penalty parameters for symmetric interior penalty Galerkin methods. *J. Comput. Appl. Math.*, **206**(2), 843–872.

Falgout, R.D., Vassilevski, P.S. and Zikatanov, L.T. [2005] On two-grid convergence estimates. *Numer. Linear Algebra Appl.*, **12**(5-6), 471–494.

Feng, X. and Karakashian, O.A. [2001] Two-level additive Schwarz methods for a discontinuous Galerkin approximation of second order elliptic problems. *SIAM J. Numer. Anal.*, **39**(4), 1343–1365 (electronic).

Fidkowski, K.J., Oliver, T.A., Lu, J. and Darmofal, D.L. [2005] p-Multigrid solution of high-order discontinuous Galerkin discretizations of the compressible Navier-Stokes equations. *J. Comput. Phys.*, **207**(1), 92–113.

Gopalakrishnan, J. and Kanschat, G. [2003] A multilevel discontinuous Galerkin method. *Numer. Math.*, **95**(3), 527–550.

Nicolaides, R.A. [1987] Deflation of conjugate gradients with applications to boundary value problems. *SIAM J. Numer. Anal.*, **24**(2), 355–365.

Persson, P.O. and Peraire, J. [2008] Newton-GMRES preconditioning for discontinuous Galerkin discretizations of the Navier-Stokes equations. *SIAM J. Sci. Comput.*, **30**(6), 2709–2733.

Prill, F., Lukáčová-Medviďová, M. and Hartmann, R. [2009] Smoothed aggregation multigrid for the discontinuous Galerkin method. *SIAM J. Sci. Comput.*, **31**(5), 3503–3528.

Proft, J. and Rivière, B. [2009] Discontinuous Galerkin methods for convection-diffusion equations for varying and vanishing diffusivity. *Int. J. Numer. Anal. Model.*, **6**(4), 533–561.

Rivière, B. [2008] *Discontinuous Galerkin methods for solving elliptic and parabolic equations*, vol. 35 of *Frontiers in Applied Mathematics.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, ISBN 978-0-898716-56-6, theory and implementation.

Rivière, B., Wheeler, M. and Banaš, K. [2000] Part ii. discontinuous galerkin method applied to a single phase flow in porous media. *Comput. Geosci*, **4**, 337–349.

Saad, Y. and Suchomel, B. [2002] ARMS: an algebraic recursive multilevel solver for general sparse linear systems. *Numer. Linear Algebra Appl.*, **9**(5), 359–378.

Sherwin, S.J., Kirby, R.M., Peiró, J., Taylor, R.L. and Zienkiewicz, O.C. [2006] On 2D elliptic discontinuous Galerkin methods. *Internat. J. Numer. Methods Engrg.*, **65**(5), 752–784.

Sun, S. and Wheeler, M. [2007] Local problem-based a posteriori error estimators for discontinuous galerkin approximations of reactive transport. *Computational Geosciences*, **11**(2), 87–101.

Tang, J.M., MacLachlan, S.P., Nabben, R. and Vuik, C. [2010] A comparison of two-level preconditioners based on multigrid and deflation. *SIAM J. Matrix Anal. Appl.*, **31**(4), 1715–1739.

Tang, J.M., Nabben, R., Vuik, C. and Erlangga, Y.A. [2009] Comparison of two-level preconditioners derived from deflation, domain decomposition and multigrid methods. *J. Sci. Comput.*, **39**(3), 340–370.

Vassilevski, P.S. [2008] *Multilevel block factorization preconditioners.* Springer, New York, ISBN 978-0-387-71563-6, matrix-based analysis and algorithms for solving finite element equations.

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

Vuik, C., Segal, A., Meijerink, J. and Wijma, G. [2001] The construction of projection vectors for a Deflated ICCG method applied to problems with extreme contrasts in the coefficients. *Journal of Computational Physics*, **172**, 426–450.

Xu, J. [1992] Iterative methods by space decomposition and subspace correction. *SIAM Rev.*, **34**(4), 581–613.

Yavneh, I. [2006] Why multigrid methods are so efficient. *Computing in Science & Engineering*, **8**(6), 12–22.