Computer Assisted Mechanics and Engineering Sciences, 11: 47–62, 2004.

Preconditioning GMRES for discontinuous Galerkin approximations

Krzysztof Banaś[1,2], Mary F. Wheeler[1]

[1] *The Center for Subsurface Modeling, Texas Institute for Computational and Applied Mathematics, The University of Texas, Austin TX 78712, U.S.A.*

[2] *Section of Applied Mathematics, Institute of Computer Modeling, Cracow University of Technology, Warszawska 24, 31-155 Kraków, Poland*
email: Krzysztof.Banas@pk.edu.pl

(Received May 2, 2002)

The paper presents an implementation and the performance of several preconditioners for the discontinuous Galerkin approximation of diffusion dominated and pure diffusion problems. The preconditioners are applied for the restarted GMRES method and test problems are taken mainly from subsurface flow modeling. Discontinuous Galerkin approximation is implemented within an *hp*-adaptive finite element code that uses hierarchical 3D meshes. The hierarchy of meshes is utilized for multi-level (multigrid) preconditioning. The results of numerical computations show the necessity of using multi-level preconditioning and insufficiency of simple stationary preconditioners, like Jacobi or Gauss–Seidel. Successful preconditioners comprise a multi-level block ILU algorithm and a special multi-level block Gauss–Seidel method.

1. Introduction

In recent years the range of application of the discontinuous Galerkin method has been extended to diffusion dominated and pure diffusion problems. Several formulations have been proposed (see the comparison in [1]). All of them lead to the solution of a large, sparse system of linear equations, with special structure and properties. Efficient solution techniques have to be designed and implemented for these systems to make the application of discontinuous Galerkin approximations feasible in practice (see e.g. [2, 3]).

In the present article formulations developed in [4] and [5] are considered, both related to the interior penalty methods from the 80's [6, 7]. The formulations lead to non-symmetric systems of linear equations, hence the GMRES method is chosen as a solver, due to its popularity and robustness for flow problems. The main subject of the paper is the comparison of the performance of several well known preconditioners adapted to discontinuous Galerkin approximation.

The paper is organized as follows. First, discontinuous Galerkin formulations for diffusion equations are presented in Sec. 2. In Sec. 3 a general setting for the restarted GMRES method with multi-level (multigrid) preconditioning is described. Implementations of different preconditioners are presented in Sec. 4. Section 5 consist of the description of numerical examples and the performance of preconditioners. Several remarks conclude the paper in Sec. 6.

2. Discontinuous Galerkin formulations

The following notation is used for a model problem, diffusion equations for a vector variable $\mathbf{u} = [u_1, u_2, ..., u_{N_u}]$:

$$\sum_{i=1}^{N_d} \left(\sum_{j=1}^{N_d} \mathbf{A}^{ij} \mathbf{u}_{,j} \right)_{,i} = 0 . \tag{1}$$

All coefficients are assumed to be functions, possibly discontinuous, of space coordinates x_i, within a domain $\Omega \in R^{N_d}$. The coefficients $\mathbf{A}^{ij}$ have the form of matrices of dimension N_u. The summation extends over the number of space dimensions N_d, with "$,i$" indicating differentiation with respect to the i-th space coordinate. In the rest of the chapter the summation signs will be dropped, assuming Einstein's summation convention.

The system (1) is completed with Dirichlet, Neumann or mixed (Robin) boundary conditions. For the Dirichlet boundary, Γ_D, essential boundary conditions are specified:

$$\mathbf{u} = \mathbf{f}_D(\mathbf{x}, t)$$

for the Neumann boundary, Γ_N, natural conditions hold:

$$\mathbf{A}^{ij} \mathbf{u}_{,j} n^i = \mathbf{g}(\mathbf{x}, t)$$

and for the Robin boundary, Γ_R, mixed boundary conditions are given:

$$\mathbf{A}^{ij} \mathbf{u}_{,j} n^i = (\mathbf{u} - \mathbf{f}_R(\mathbf{x}, t))\, \mathbf{K}_R(\mathbf{x}, t) .$$

Above, $\mathbf{n} = [n^1, .., n^{N_d}]$ is an outward unit vector, normal to domain boundary $\Gamma = \partial\Omega$ and $\mathbf{f}_D$, $\mathbf{g}$, $\mathbf{f}_R$ and $\mathbf{K}_R$ are problem dependent data.

For the purpose of discontinuous Galerkin approximations the computational domain Ω is divided into finite elements Ω_e. For all inter-element boundaries, $\Gamma_{ef} = \partial\Omega_e \cap \partial\Omega_f$ ($\bigcup \Gamma_{ef} = \Gamma_{\text{int}}$), a unique normal vector $\mathbf{n}$ is specified (for external boundaries $\mathbf{n}$ coincides with the outward normal introduced earlier, hence the same notation). The jump and average operators are defined for any function $\mathbf{v}$ on Γ_{int}:

$$[\mathbf{v}] = \mathbf{v}^L - \mathbf{v}^R = \mathbf{v}|_{\partial\Omega_e \cap \Gamma_{ef}} - \mathbf{v}|_{\partial\Omega_f \cap \Gamma_{ef}} ,$$

$$<\mathbf{v}> = 0.5 * (\mathbf{v}^L + \mathbf{v}^R) = 0.5 * (\mathbf{v}|_{\partial\Omega_e \cap \Gamma_{ef}} + \mathbf{v}|_{\partial\Omega_f \cap \Gamma_{ef}}) .$$

A finite dimensional space V of element-wise polynomial functions is introduced.

The first presented formulation is the one developed in [4], called here the non-symmetric discontinuous Galerkin (NDG) formulation:

Find $\mathbf{u} \in V$ such that for all test functions $\mathbf{w} \in V$

$$a_{\text{NDG}}(\mathbf{u}, \mathbf{w}) = l_{\text{NDG}}(\mathbf{w}) \tag{2}$$

where

$$a_{\text{NDG}}(\mathbf{u}, \mathbf{w}) =$$

$$\int_\Omega \mathbf{A}^{ij} \mathbf{w}_{,i} \mathbf{u}_{,j} d\Omega$$

$$+ \int_{\Gamma_{\text{int}}} \left(<\mathbf{A}^{ij} n^i \mathbf{w}_{,j}> [\mathbf{u}] - [\mathbf{w}] <\mathbf{A}^{ij} n^i \mathbf{u}_{,j}> \right) d\Gamma$$

$$+ \int_{\Gamma_D} \left(\mathbf{A}^{ij} n^i \mathbf{w}_{,j} \mathbf{u} - \mathbf{A}^{ij} n^i \mathbf{w} \mathbf{u}_{,j} \right) d\Gamma - \int_{\Gamma_R} \mathbf{K}_R \mathbf{w} \mathbf{u} d\Gamma$$

$$l_{\mathrm{NDG}}(\mathbf{v}) =$$
$$\int_{\Gamma_D} \mathbf{A}^{ij} n^i \mathbf{w}_{,j} \mathbf{f}_D \mathrm{d}\Gamma + \int_{\Gamma_N} \mathbf{w} \mathbf{g} \mathrm{d}\Gamma - \int_{\Gamma_R} \mathbf{K}_R \mathbf{w} \mathbf{f}_R \mathrm{d}\Gamma .$$

At the infinite dimensional limit the formulation (2) is consistent with the original problem (1) in the sense that the solutions to (1) are also solutions to (2), and that sufficiently smooth solutions to (2) solve (1) as well.

Another consistent discontinuous Galerkin formulation is obtained by adding to the bilinear and linear forms a_{NDG} and l_{NDG} stabilizing terms that aim at directly enforcing the continuity of solution across inter-element boundaries through penalties. The resulting formulation is the Non-symmetric Interior Penalty Galerkin (NIPG) method [5] and reads as follows:

Find $\mathbf{u} \in V$ such that for all test functions $\mathbf{w} \in V$

$$a_{\mathrm{NIPG}}(\mathbf{u}, \mathbf{w}) = l_{\mathrm{NIPG}}(\mathbf{w}) \tag{3}$$

where

$$a_{\mathrm{NIPG}}(\mathbf{u}, \mathbf{w}) = a_{\mathrm{NDG}}(\mathbf{u}, \mathbf{w}) + \int_{\Gamma_{\mathrm{int}}} \frac{\sigma}{h} [\mathbf{w}][\mathbf{u}] \mathrm{d}\Gamma + \int_{\Gamma_D} \frac{\sigma}{h} \mathbf{w} \mathbf{u} \mathrm{d}\Gamma$$

and

$$l_{\mathrm{NIPG}}(\mathbf{w}) = l_{\mathrm{NDG}}(\mathbf{w}) + \int_{\Gamma_D} \frac{\sigma}{h} \mathbf{w} \mathbf{f}_D \mathrm{d}\Gamma .$$

Above, $\sigma > 0$ is a parameter, constant over each face, and h is a linear size of a face (the integrals involving h should be understood as sums of integrals over element faces). The discussion on the choice of σ can be found in [8].

For all examples in this paper, σ is taken as equal to the degree of approximation p, the value suggested in [8] and found to be optimal for the convergence of the GMRES linear solver. For hp approximations, neighboring elements can have different sizes, due to h-refinements. For such cases, the face integrals are computed over the smaller faces (faces of more refined elements), using their sizes for h.

REMARK. For pure diffusion problems the NDG formulation is theoretically proven and experimentally confirmed to be stable only for higher order approximations [4] (in practice the degree of approximation p has to be ≥ 2). The NIPG formulation is stable also for piecewise linear approximation [5], however, the strict conservation property of the NDG formulation does not hold for NIPG.

2.1. Implementation

Both discontinuous Galerkin formulations presented above are implemented in an hp-adaptive finite element code. In all examples presented in Sec. 5 the code is used with 3D elements, obtained as linear transformations of the master prismatic element shown in Fig. 1. There are two sets of element shape functions, both defined for the master element. The functions from the first set, called "tensor" shape functions, are tensor products of monomials in xy plane (1, x, y, x^2, xy, y^2, etc.), forming a basis for complete polynomials, with standard monomials in z direction. The functions from the second set, "complete" shape functions, form, for a given degree of approximation p, a basis for a space of complete polynomials of degree p in 3D. The numerical comparison of both sets of shape functions can be found in [9].

The code implements h and p adaptivity. p adaptivity consists in specifying the order of approximation separately for each element, an obvious solution for discontinuous Galerkin discretizations. h adaptivity is implemented using uniform divisions of a "father" element into eight "son" elements. The hierarchical character of meshes is exploited in multi-level preconditioning.

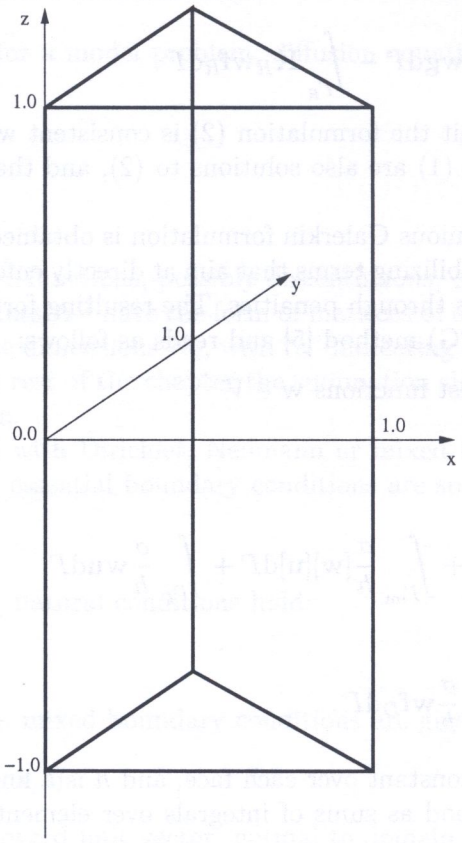


Fig. 1. Master prismatic element

2.2. System of linear equations

The standard finite element procedures applied to the NDG or NIPG formulations lead to the system of linear equations:

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \tag{4}$$

where $\mathbf{A}$ denotes the global stiffness matrix, $\mathbf{x}$ is the vector of unknowns and $\mathbf{b}$ is the global right hand side vector (load vector). The system matrix is semi-positive definite for the NDG formulation and positive definite for the NIPG formulation.

Since supports of global basis functions are limited to single elements, each element in the mesh possess a separate small vector of degrees of freedom (in contrast to classical linear continuous finite elements where degrees of freedom are associated with nodes). This induces the splitting of the global vector of unknowns into non-overlapping blocks $\mathbf{x}_l$ and the corresponding splittings of the right hand side vector $\mathbf{b}$ into blocks $\mathbf{b}_l$ and the system matrix $\mathbf{A}$ into blocks $\mathbf{A}_{lk}$, according to the formula:

$$\mathbf{b}_l = \sum_{k=1}^{N_{bl}} \mathbf{A}_{lk}\mathbf{x}_k \,.$$

The diagonal blocks $\mathbf{A}_{lk}$, $l = k$ contain entries obtained by integrating over elements' interiors and faces while the off-diagonal blocks $\mathbf{A}_{lk}$, $k \neq l$ are results of integration over elements' faces exclusively. The blocks $\mathbf{A}_{lk}$ are called the elementary blocks and are the basis for the storage scheme of the stiffness matrix adopted in the code. The blocks are full, possibly not symmetric, but positioned symmetrically within $\mathbf{A}$. They are usually square, except for the case of different degrees

```
compute the initial residual of the system: r_0 := B^{-1}(b - Ax^0)
normalize the residual: r̄_0 := r_0/||r_0||
for i = 1, 2, ..., N_ksv
  compute preconditioned matrix-vector product: r_i := B^{-1}Ar̄_{i-1}
  orthonormalize r_i wrt all previous r̄_j, j = 1, .., i - 1 by the modified
    Gram-Schmidt procedure obtaining r̄_i
  solve the GMRES minimization problem and check convergence
  if convergence achieved form the approximate solution and leave GMRES
end for
form the approximate solution and substitute it for the initial guess x^0
start from the beginning
```

Fig. 2. Restarted GMRES method

of approximation used in neighboring elements when the off-diagonal blocks are rectangular. The whole matrix has symmetric non-zero structure but is non-symmetric.

To solve the system (4) the restarted preconditioned GMRES method [10], schematically presented in Fig. 2, is used. The first initial guess $\mathbf{x}^0$ is zero for the considered elliptic problems and N_{ksv} is the number of Krylov space basis vectors of the Krylov space

$$\text{span}\left\{\mathbf{r}_0, \mathbf{B}^{-1}\mathbf{A}\mathbf{r}_0, (\mathbf{B}^{-1}\mathbf{A})^2\mathbf{r}_0, ..., (\mathbf{B}^{-1}\mathbf{A})^{(N_{\text{ksv}}-1)}\mathbf{r}_0\right\}$$

in which the solution is sought. $\mathbf{B}^{-1}$ represents the action of a preconditioner, the construction of which will be the subject of the next section.

3. Preconditioners for the GMRES method

In general, the purpose of preconditioning is to make the product $\mathbf{B}^{-1}\mathbf{A}$ close to the identity matrix, in order to speed up the convergence of an iterative solver [11]. Hence any approximate solver, e.g. one or few iterations of another iterative solver, can be used as a preconditioner. In practical implementations a preconditioner for the GMRES has to deliver either the solution to the auxiliary system

$$\mathbf{B}\mathbf{r}_i = \mathbf{z}_i \tag{5}$$

or the product

$$\mathbf{B}^{-1}\mathbf{A}\bar{\mathbf{r}}_{i-1}\,. \tag{6}$$

In the following, both approaches will be considered and implementations for typical preconditioners presented.

3.1. ILU(0)

Incomplete factorization preconditioners are very popular and several variations have been developed [12]. The version adopted for discontinuous Galerkin approximations is the standard ILU(0) method acting on elementary blocks of the stiffness matrix (see the definition in Sec. 2.2) instead of the individual entries. In the implementation, the algorithm performs separately two operations: the product with the stiffness matrix and the solution to the auxiliary system (5). Hence, it requires the storage for two (equal size) matrices: the original matrix $\mathbf{A}$ and its incompletely factorized form.

3.2. Stationary methods

One iteration of a stationary method can be formally written as [11]:

$$\mathbf{x}^m = \mathbf{x}^{m-1} + \mathbf{D}^{-1}(\mathbf{b} - \mathbf{A}\mathbf{x}^{m-1}) \tag{7}$$

with $\mathbf{D}^{-1}$ denoting an operator, usually some approximation to the inverse $\mathbf{A}^{-1}$, implied by the actual algorithm. Any algorithm of the form (7), with suitable choices for the initial guess and the right hand side and followed by suitable subtractions, can be used to provide directly the product (6) [13]. Block Jacobi and block Gauss–Seidel methods acting on elementary blocks of the stiffness matrix have been adopted as preconditioners. For the block Jacobi method the explicit formula for $\mathbf{B}^{-1}$ reads [14]:

$$\mathbf{B}_{BJ}^{-1} = \mathbf{D}_{BJ}^{-1} = \sum_{l=1}^{N_{bl}} \mathbf{R}_l^T \mathbf{A}_{ll}^{-1} \mathbf{R}_l \tag{8}$$

while for the block Gauss–Seidel:

$$\mathbf{B}_{GS}^{-1} = \mathbf{D}_{GS}^{-1} = \left(I - \prod_{l=1}^{N_{bl}} (I - \mathbf{R}_l^T \mathbf{A}_{ll}^{-1} \mathbf{R}_l \mathbf{A}) \right) \mathbf{A}^{-1} . \tag{9}$$

The operators $\mathbf{R}_l$ and $\mathbf{R}_l^T$ denote the restriction of the whole vector $\mathbf{x}$ to a single block $\mathbf{x}_l$ and the prolongation from $\mathbf{x}_l$ to $\mathbf{x}$, respectively. In practice, these operators are not formed explicitly. In the code, restriction and prolongation are always performed using a special algorithm and the storage based on elementary blocks.

The implementation consists of a loop over blocks (elements) and the solution of local problems, with matrices $\mathbf{A}_{ll}$ as system (stiffness) matrices. Each local problem corresponds to a small PDE, with the same weak formulation as the global problem, the domain consisting of a single element and the Dirichlet boundary conditions provided by the current values of solution in the neighboring elements. Since the product $\mathbf{B}^{-1}(\mathbf{b} - \mathbf{A}\mathbf{x}^{m-1})$ is obtained without the separate product $\mathbf{A}\mathbf{x}^{m-1}$, the implementation uses only the storage for the stiffness matrix $\mathbf{A}$. The blocks $\mathbf{A}_{ll}$ are inverted using a direct solver (e.g. a LAPACK library procedure). The inverted diagonal blocks $\mathbf{A}_{ll}^{-1}$ replace the original blocks $\mathbf{A}_{ll}$.

Single element subdomains do not provide effective smoothing by stationary iterations, since the error on the inter-element boundary is not sufficiently reduced [15]. The extension of stationary methods is given by single level, overlapping Schwarz preconditioners [14]. If blocks in (8) and (9) are larger than elementary blocks induced by the approximation (these larger, possibly overlapping blocks will be denoted by $\widehat{\mathbf{x}}_l$ and $\widehat{\mathbf{A}}_{ll}$), than the implementation of (7) still produces a preconditioner, with even stronger effect than the original stationary methods. However, now a separate storage is required for the inverted blocks $\widehat{\mathbf{A}}_{ll}^{-1}$ and this storage is larger than for the blocks $\widehat{\mathbf{A}}_{ll}$. Blocks $\widehat{\mathbf{A}}_{ll}$ are sparse while their inverses $\widehat{\mathbf{A}}_{ll}^{-1}$ are full. The additional storage required depends upon the size of blocks and will be the subject of investigations in the section concerned with numerical examples.

To fulfill the requirements of the domain decomposition theory, the larger blocks of unknowns correspond to subdomains within the computational domain. Since blocks $\widehat{\mathbf{A}}_{ll}$ are inverted by a direct solver, the size of subdomains is limited to several elements. To remove errors on inter-element boundaries, each face should be internal for at least one subdomain. One possibility is to create a subdomain, comprised of two adjacent elements, for each face. Another is to create, for each element, a subdomain consisting of the element and all its neighbors across faces (see Fig. 3). Both designs lead to significantly increased storage. The solution adopted in the code, is to create, subsequently, a subdomain for each element, but to add to it only those neighbors connected across the faces, that are not yet inside any other subdomain. Such subdomains are called "large" subdomains, contrary to "small", single element, subdomains.

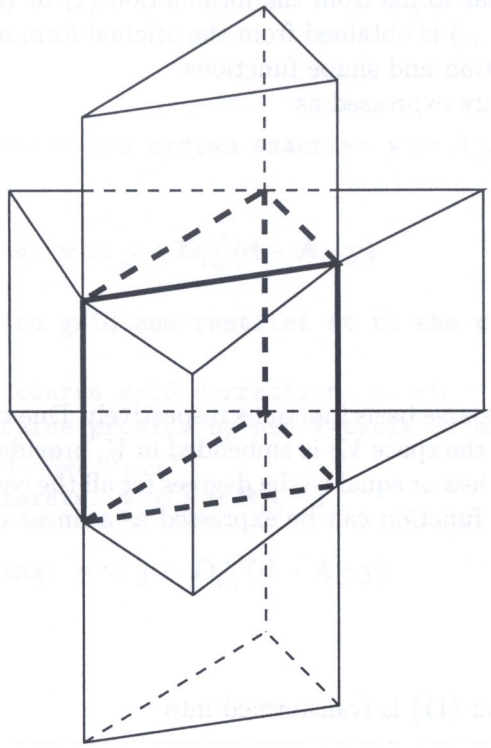


Fig. 3. An example "large" subdomain, a prismatic element (thick lines) and all its neighbors across faces (thin lines)

3.3. Multi-level preconditioning

Iteration (7) can be interpreted as a sequence of local projections of the error on the local spaces related to the blocks $\mathbf{x}_l$ [14]. Since projections are only local, stationary methods act mainly as a smoother – an algorithm that reduces high frequency components of the error. In order to reduce low frequency components, an additional coarse grid correction is introduced [16]. Algebraically, the coarse grid correction has the same form as the correction on the fine level:

$$\overline{\mathbf{x}} = \widetilde{\mathbf{x}} + \widetilde{\mathbf{e}}_C = \widetilde{\mathbf{x}} + \mathbf{R}_C^T \mathbf{A}_C^{-1} \mathbf{R}_C (\mathbf{b} - \mathbf{A}\widetilde{\mathbf{x}}) \tag{10}$$

with $\overline{\mathbf{x}}$ – the corrected solution, $\widetilde{\mathbf{x}}$ – the current iterate and $\widetilde{\mathbf{e}}_C$ the approximation of the error on the coarse grid. In our implementation, the definitions of the coarse space and the operators $\mathbf{R}_C$ and $\mathbf{A}_C$ utilize the hierarchical structure of the meshes produced by adaptivity.

The computational mesh for which the solution to the problem is sought is considered as a fine mesh with the corresponding fine space V. For every element active in the given mesh it is checked whether it has resulted from a refinement of a father element or not. If the element does not result from a refinement it is included in the coarse mesh as well, otherwise the father element is added to the coarse mesh. This means, for example, that for uniform refinements of a prismatic mesh, the coarse mesh has eight time less elements (and respectively degrees of freedom) than the fine mesh.

The coarse space V_C is spanned by the basis functions corresponding to elements of the coarse mesh. Given the spaces V and V_C, the construction of the restriction and prolongation operators follows from the variational statement of the coarse grid correction problem [14]:

Find $\widetilde{e}_C \in V_C$ such that

$$a_C(\widetilde{e}_C, v_C) = a(u^* - \widetilde{u}, v_C') = f(v_C') - a(\widetilde{u}, v_C') \qquad \forall v_C \in V_C \tag{11}$$

with $\widetilde{e}_C$ the approximation to the error on the coarse grid (corresponding to the vector $\widetilde{\mathbf{e}}_C$), u^* the exact solution to the original problem, $\widetilde{u}$ the solution corresponding to the current iterate $\widetilde{\mathbf{x}}$, $a(.,.)$

and $f(.)$ the bilinear and linear forms from the formulation (2) or (3) and v'_C the extension of v_C into V. The bilinear form $a_C(.,.)$ is obtained from the original form $a(.,.)$ by changing properly the respective domains of integration and shape functions.

Functions in the space V are expressed as

$$v = \sum_i v^i \phi^i$$

while functions in V_C as

$$v_C = \sum_i v^i_C \phi^i_C$$

where ϕ^i and ϕ^i_C are fine and coarse basis functions respectively. Due to the fact that the coarse mesh is embedded in the fine mesh, the space V_C is embedded in V, provided the degree of approximation for each coarse grid element is less or equal to the degrees for all the corresponding fine grid elements. Hence, any coarse grid shape function can be expressed as a linear combination of fine grid shape functions:

$$\phi^i_C = \sum_j r_{ij} \phi^j \,.$$

Coarse grid correction problem (11) is transformed into

$$a_C(\widetilde{e}_C, v_C) = \sum_i v^i_C \sum_j r_{ij} \left(f(\phi^j) - a(\widetilde{u}, \phi^j) \right) \,.$$

The stiffness matrix $\mathbf{A}_C$ corresponding to the bilinear form $a_C(\widetilde{e}_C, v_C)$ is obtained by integrating over coarse elements. The term $f(\phi^j) - a(\widetilde{u}, \phi^j)$ corresponds to computing the residual of the fine grid problem for the current iterate $\widetilde{u}$. The coefficients r_{ij} are identified with the entries of restriction operator $\mathbf{R}_C$. Given $\mathbf{R}_C$ and its transpose $\mathbf{R}^T_C$, all the ingredients necessary for the implementation of the coarse grid correction step (10) are then defined.

To obtain the coefficients r_{ij} for each pair of father-son elements, a local L_2 projection problem is solved. Each father's shape function is projected onto son's local space. Since elements are geometrically linear the coefficients r_{ij} depend only on the degrees of approximation for the father and for the son and on the refinement type of the father. For each combination of these parameters appearing in the mesh, the coefficients are precomputed, stored and than used in actual operations.

REMARK. For higher orders of approximation, solving L_2 projection problems for each shape function of the master element may involve substantial computational effort. For these cases different shape functions can be considered, such as e.g. L_2 orthogonal shape functions described in [15], that make mass matrices for L_2 problems diagonal and reduce the computational effort.

To fully exploit the hierarchical structure of adaptive meshes, the classical V–cycle multigrid algorithm [16], presented in Fig. 4, has been implemented. $\mathbf{R}_{C_i}$ and $\mathbf{R}^T_{C_i}$ denote restriction and prolongation operators for a given coarse grid, belonging to the hierarchy of grids, $\Omega_{C_0} \subset \Omega_{C_1} \subset \ldots \subset \Omega_{C_i} \ldots \subset \Omega_{C_{\max}} \equiv \Omega_f$, obtained by coarsening the finest grid Ω_f. In a smoother algorithm $\mathbf{y} := \mathbf{y} + \mathbf{D}^{-1}_{C_i}(\mathbf{d} - \mathbf{A}_{C_i}\mathbf{y})$, $\mathbf{D}^{-1}_{C_i}$ denotes a linear operator implied by the actual algorithm. Parameters N_{pre} and N_{post} are the numbers of pre-smoothing and post-smoothing steps. Coarse grid correction problems are solved approximately, using, recursively, the V–cycle algorithm. The whole algorithm acts for the finest grid as a multi-level smoother, so when called with parameters $\texttt{MG}(\widetilde{\mathbf{x}}, \mathbf{b}, C_{\max}, N_{\text{pre}}, N_{\text{post}})$ it returns a vector $\widetilde{\mathbf{x}} := \widetilde{\mathbf{x}} + \mathbf{D}^{-1}_{MG}(\mathbf{b} - \mathbf{A}\widetilde{\mathbf{x}})$, when now $\mathbf{D}^{-1}_{MG}$ includes smoothing at all grid levels. Similarly to the stationary methods, the V–cycle multigrid algorithm is used as a (multi-level) preconditioner with $\mathbf{B}^{-1}_{MG} = \mathbf{D}^{-1}_{MG}$.

```
V-cycle(y, d, C_i, N_pre, N_post)
begin
  if (C_i = C_0)
    on the coarsest mesh solve the system exactly: y := A_{C_0}^{-1} d;
  else
    for j = 1, 2, ..., N_pre
      perform pre-smoothing: y := y + D_{C_i}^{-1}(d - A_{C_i} y)
    endfor
    compute residual on fine grid and restrict it to the coarser grid:
      g := R_{C_{i-1}}(d - A_{C_i} y)
    set initial guess for coarse grid correction: z := 0
    solve approximately coarse problem by one V-cycle of multigrid:
      z := V-cycle(z, g, C_{i-1}, N_pre, N_post)
    correct the current iterate: y := y + R_{C_{i-1}}^T z
    for j = 1, 2, ..., N_post
      perform post-smoothing: y := y + D_{C_i}^{-1}(d - A_{C_i} y)
    endfor
  endif
end
```

Fig. 4. One V–cycle of the classical multigrid algorithm

3.4. ILU(0) as a smoother

Apart from the stationary methods – the most popular smoothers, represented in the code by the block Gauss–Seidel algorithm – incomplete factorization can also be used within multigrid algorithm [17]. If D_{ILU}^{-1} denotes an operator for solving system (4) using incompletely factorized stiffness matrix $\mathbf{A}$ then ILU(0) algorithm can be made part of a smoother algorithm that produces:

$$\overline{\mathbf{x}} := \widetilde{\mathbf{x}} + \mathbf{D}_{\mathrm{ILU}}^{-1}(\mathbf{b} - \mathbf{A}\widetilde{\mathbf{x}})$$

ILU smoothing is applied at each grid level within the multigrid V–cycle algorithm, forming a multi-level block ILU(0) preconditioner.

4. Numerical examples

All computations in this section were performed on an SGI Origin 200 computer with MIPS R10000 (180 MHz) processor ($\approx$ 100MFLOPS, SPECfp_base95 = 14.4).

4.1. An example with a smooth solution on a regular grid

The first example is Laplace's equation

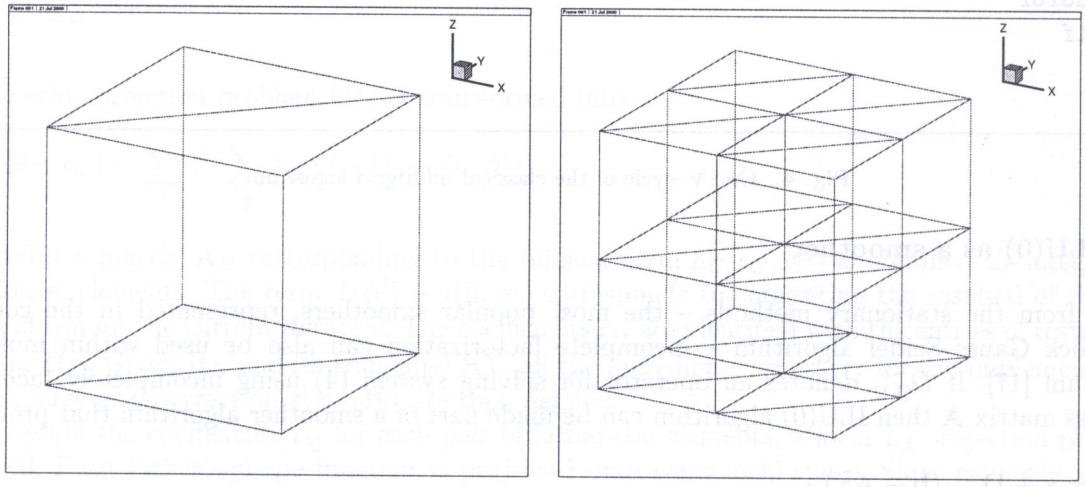
$$\Delta u = \Delta u_{ex}$$

where u_{ex} is the known exact solution:

$$u_{ex} = \exp\left(-x^2 - y^2 - z^2\right)$$

The computational domain consist of the box $[0,1] \times [0,1] \times [0,1]$ and boundary conditions are chosen to match the exact solution. The initial mesh consist of two prismatic elements shown in Fig. 5. The consecutive meshes are obtained by uniform refinements of the initial mesh and labeled using the number of refinement levels (element generations).

The problem is solved using both formulations, NDG and NIPG, "complete" element shape functions and the restarted GMRES method with 20 Krylov vectors. The following GMRES preconditioners are compared: single level block Gauss–Seidel – SBGS, multi-level block Gauss–Seidel – MBGS (classical multigrid), single level block ILU(0) – SBILU and multi-level block ILU(0) – MBILU (multigrid with block ILU(0) smoother). For stationary methods (Schwarz preconditioners) two types of preconditioner blocks are considered: single element blocks ("-s" after the preconditioner symbol) and larger blocks, discussed above ("-l" after the preconditioner symbol). For all preconditioners only one smoothing iteration (preconditioner solve for single level preconditioners and post-smoothing iteration for multigrid) is applied at the finest mesh level, to point out the effect of the coarse grid correction. For multi-level preconditioners, one pre- and one post-smooth iteration is applied at intermediate grid levels. The restarted GMRES is used to solve the problem on the coarsest level. There is no special ordering of blocks applied. Additional experiments, not reported here, revealed lack of strong dependence of the convergence rates on the ordering of blocks.



(a) initial mesh (b) first refined mesh

Fig. 5. Test example with smooth solution – initial mesh with 2 prismatic elements and first refined mesh with 16 elements

The results are presented for NDG and NIPG discretizations and two degrees of approximation, $p = 2$ and $p = 5$. For $p = 2$ and five mesh levels (element generations) the number of degrees of freedom $N_{\text{dof}} = 81920$ and the L_2 norm of the error $\|e\|_{L_2} = 0.98 \cdot 10^{-4}$, while the H^1 seminorm of the error $|e|_{H^1} = 0.93 \cdot 10^{-3}$. The error norms are computed as the square root of the sum of integrals over elements' interiors of the square of the difference between the approximate and the exact solution (or its derivative). For $p = 5$ and only two mesh levels the respective numbers are: $N_{\text{dof}} = 896$, $\|e\|_{L_2} = 0.56 \cdot 10^{-5}$, $|e|_{H^1} = 0.13 \cdot 10^{-3}$. For $p = 5$ and three mesh levels with $N_{\text{dof}} = 7168$ the results are: $\|e\|_{L_2} = 0.93 \cdot 10^{-7}$, $|e|_{H^1} = 0.43 \cdot 10^{-4}$.

The first three tables present, for each mesh and preconditioner, the number of GMRES iterations to achieve the reduction of the initial residual by 10^{-6}, the necessary storage (the same for both formulations) and the total CPU time for execution, including the creation of the stiffness matrix. "NO" in the place "#iter" indicates that the convergence has not been obtained after 100 GMRES iterations. The next two figures (Fig. 6 and Fig. 7) show graphically the dependence of the execution time and the storage required by the solver on the number of degrees of freedom. NIPG formulation and $p = 2$ are chosen for this comparison (the results for NDG formulation are qualitatively the

Table 1. Test example with smooth solution – the number of GMRES iterations to reach convergence, the necessary storage (in MBytes) and the total execution time (in seconds) for different preconditioners, NDG discretization and the degree of approximation $p = 2$

Preconditioner		Number of grid levels			
		2	3	4	5
SBGS-s	#iter	17	40	91	NO
	storage	0.09	0.81	6.87	56.57
	CPU time	0.041	0.49	7.89	—
SBGS-l	#iter	9	13	18	37
	storage	0.19	1.94	17.01	142.98
	CPU time	0.044	0.51	5.29	69.71
SBILU	#iter	10	15	25	57
	storage	0.15	1.44	12.08	99.56
	CPU time	0.046	0.54	11.29	914.83
MBGS-s	#iter	17	NO	NO	NO
	storage	0.20	1.22	9.58	77.77
	CPU time	0.075	—	—	—
MBGS-l	#iter	8	11	12	13
	storage	0.27	2.19	19.00	160.45
	CPU time	0.067	0.65	6.12	54.09
MBILU	#iter	8	10	12	13
	storage	0.24	1.66	13.54	111.57
	CPU time	0.067	0.63	8.49	254.96

Table 2. Test example with smooth solution – the number of GMRES iterations to reach convergence and the total execution time (in seconds) for different preconditioners, NIPG discretization and the degree of approximation $p = 2$

Preconditioner		Number of grid levels			
		2	3	4	5
SBGS-s	#iter	12	19	39	81
	CPU time	0.040	0.38	4.88	67.86
SBGS-l	#iter	7	11	19	40
	CPU time	0.043	0.47	5.37	73.91
SBILU	#iter	9	15	30	59
	CPU time	0.045	0.52	12.87	945.08
MBGS-s	#iter	11	13	13	13
	CPU time	0.074	0.49	4.52	37.95
MBGS-l	#iter	6	8	8	7
	CPU time	0.052	0.56	5.14	42.25
MBILU	#iter	7	8	9	9
	CPU time	0.054	0.55	7.18	190.83

Table 3. Test example with smooth solution – the number of GMRES iterations to reach convergence, the necessary storage (in MBytes) and the total execution time (in seconds) for different preconditioners, NDG discretization and the degree of approximation $p = 5$

Preconditioner		Number of grid levels	
		2	3
SBGS-s	#iter	58	NO
	storage	1.33	13.68
	CPU time	4.18	—
SBGS-l	#iter	9	15
	storage	4.40	47.45
	CPU time	3.80	36.74
SBILU	#iter	12	20
	storage	3.24	32.09
	CPU time	3.95	37.70
MBGS-s	#iter	24	94
	storage	2.26	21.96
	CPU time	4.74	68.59
MBGS-l	#iter	7	9
	storage	4.63	51.95
	CPU time	4.29	49.22
MBILU	#iter	8	12
	storage	3.46	35.48
	CPU time	4.50	43.02

same for the methods that converge in both cases). It can be observed that the storage grows almost linearly for all considered methods while the CPU time for execution scales linearly with the problem size only for multi-level methods with stationary iterations as preconditioners.

Several additional observations follow:

- large blocks improve significantly the convergence of block Gauss–Seidel algorithm,
- multilevel preconditioning reduces substantially the dependence of the convergence rates on the mesh size h (this effect would be even greater if more smoothing steps were used at each level),
- multilevel algorithms require only slightly more storage than single level algorithms; this fact results from using the hierarchy of refined elements as a basis for multilevel preconditioners. Since each divided element has eight "sons" the storage required for a next coarser grid is one eights the storage on a fine level. Asymptotically (for infinite number of mesh levels), adding a hierarchy of grids can increase the storage by no more than 15%,
- the increase in storage and CPU time per iteration is the price paid for better convergence properties of block Gauss–Seidel with larger blocks and ILU preconditioners,
- using higher order approximation increases the size of elementary blocks but reduces the number of mesh levels, elements and the degrees of freedom necessary to reach some assumed accuracy; as a result the scaling properties of the preconditioners are less important giving an advantage to single level and ILU preconditioners,
- for smooth problems, NIPG formulation improves the convergence of GMRES with multi-level preconditioners, especially for block Gauss–Seidel method with small blocks; for single level block Gauss–Seidel with large blocks and ILU preconditioners, it has no evident positive effect,

- two methods proving to be the best for this test example are MBGS-l and MBILU, the first with better scalability with respect to the number of degrees of freedom and larger required storage.

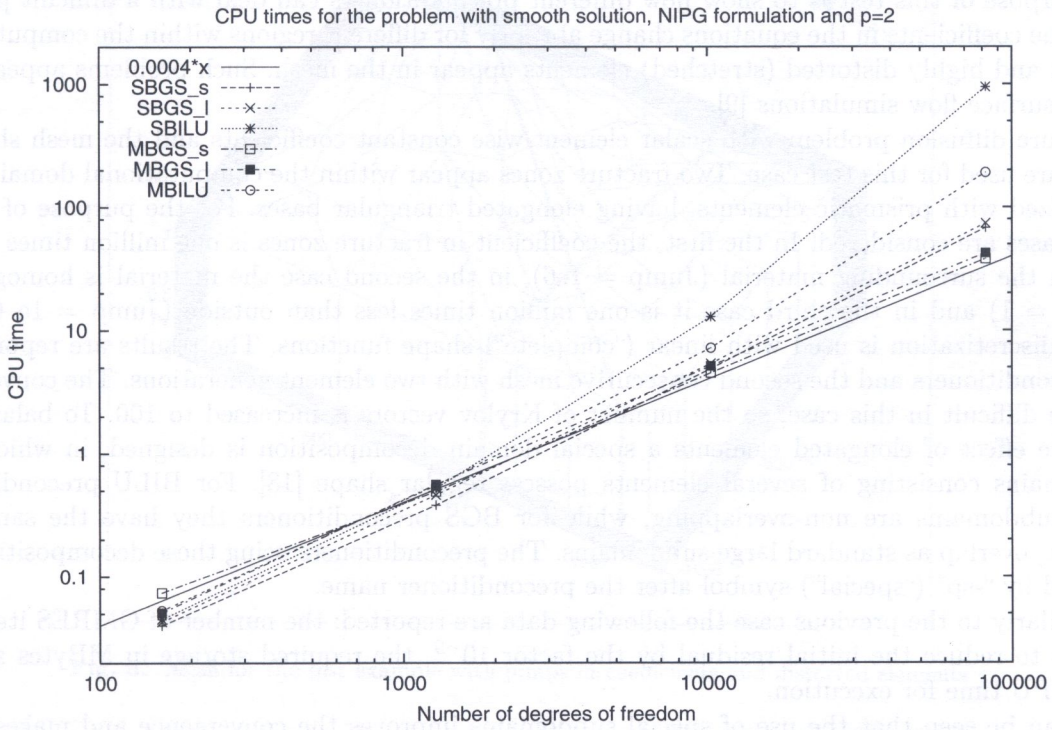


Fig. 6. Test example with smooth solution – execution times versus the number of degrees of freedom for different preconditioners, NIPG formulation and $p = 2$

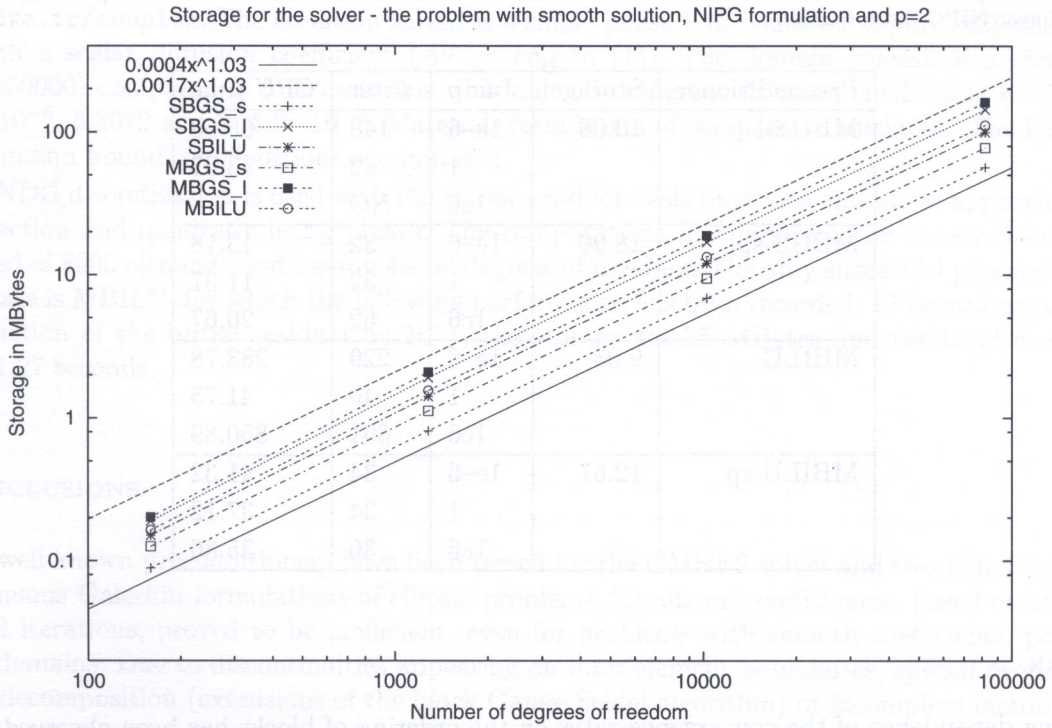


Fig. 7. Test example with smooth solution – necessary storage versus the number of degrees of freedom for different preconditioners, NIPG formulation and $p = 2$

4.2. An example with large jumps in coefficients and distorted elements with high aspect ratios

The purpose of this test is to show how different preconditioners can deal with a difficult problem when the coefficients in the equations change abruptly for different regions within the computational domain and highly distorted (stretched) elements appear in the mesh. Such problems appear often for subsurface flow simulations [9].

A pure diffusion problem with scalar element-wise constant coefficients and the mesh shown in Fig. 8 are used for this test case. Two fracture zones appear within the computational domain, both discretized with prismatic elements, having elongated triangular bases. For the purpose of testing three cases are considered. In the first, the coefficient in fracture zones is one million times greater than in the surrounding material (Jump = 1e6), in the second case the material is homogeneous (Jump = 1) and in the third case it is one million times less than outside (Jump = 1e–6). The NIPG discretization is used with linear ("complete") shape functions. The results are reported for all preconditioners and the second consecutive mesh with two element generations. The convergence is more difficult in this case, so the number of Krylov vectors is increased to 100. To balance the negative effect of elongated elements a special domain decomposition is designed, in which large subdomains consisting of several elements possess regular shape [18]. For BILU preconditioners these subdomains are non-overlapping, while for BGS preconditioners they have the same, one element, overlap as standard large subdomains. The preconditioners using these decompositions are denoted by "-sp" ("special") symbol after the preconditioner name.

Similarly to the previous case the following data are reported: the number of GMRES iterations (#iter) to reduce the initial residual by the factor 10^{-9}, the required storage in MBytes and the total CPU time for execution.

It can be seen that the use of special subdomains improves the convergence and makes it less sensitive to the variations of the diffusion coefficient.

Table 4. Test example with large jumps in coefficients and distorted elements: the required storage, the number of GMRES iterations to reach convergence and the total execution time for different preconditioners and linear NIPG discretization

Preconditioner	Storage	Jump	#iter	CPU time
MBGS-l	10.08	1e–6	148	66.36
		1	42	13.02
		1e6	NO	—
MBGS-sp	18.96	1e–6	32	13.18
		1	32	11.37
		1e6	62	20.67
MBILU	9.08	1e–6	229	288.78
		1	40	41.75
		1e6	691	850.89
MBILU-sp	12.67	1e–6	34	41.32
		1	34	37.12
		1e6	36	35.86

REMARK

- strong dependence of the convergence rates on the ordering of blocks has been observed for the case Jump = 1e6 and ILU preconditioners. The best results (reported in the table) have been obtained when blocks are ordered according to the increasing values of the diffusion coefficient [15].

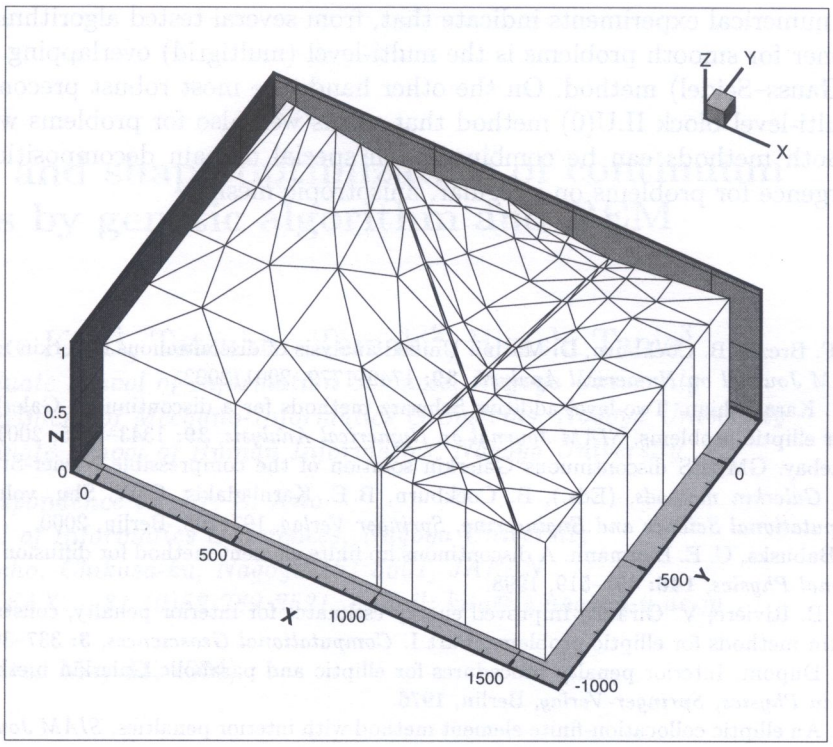


Fig. 8. Mesh for the test example with jumps in coefficients and distorted elements

4.3. An example with layered medium

The last example has been chosen to show the robustness of MBILU preconditioner. The test case comes from the suite of problems related to subsurface contaminant transport published at `www.andra.fr/couplex`. The equation solved is a single phase flow equation, a pure diffusion problem, with a scalar diffusion coefficient ($A^{ij} = k\delta_{ij}$ in (1)). The domain consist of a thin strip (692×250000) composed of four materials with radically different diffusion coefficients k: 25.2288, $3.1536 \cdot 10^{-6}$, 6.3072 and $3.1536 \cdot 10^{-5}$. Materials form layers of comparable thickness. The Dirichlet and Neumann boundary conditions are imposed.

The NDG discretization is used with the tensor product basis functions and linear approximation in z direction and quadratic in xy planes. The computations are performed on once refined mesh composed of 8000 elements and having 48000 degrees of freedom. The only successful preconditioner in this case is MBILU, for which the following performance has been recorded: 17 iterations to reach the reduction of the initial residual by 10^{-9}, the storage 193.35 MBytes and the total execution time 691.27 seconds.

5. Conclusions

Several well known preconditioners have been tested for the GMRES solver and two non-symmetric discontinuous Galerkin formulations of elliptic problems. Simple preconditioners, based on classical standard iterations, proved to be inefficient, even for problems with smooth coefficients, posed in regular domains. Due to discontinuities appearing on inter-element boundaries, special overlapping domain decomposition (extensions of the block Gauss–Seidel algorithm) or incomplete factorization preconditioners were necessary to guarantee convergence. For large scale problems, to overcome the effect of deteriorating convergence of single level preconditioners, multi-level preconditioners, based on the classical geometric multigrid method, have been implemented.

The reported numerical experiments indicate that, from several tested algorithms, the most efficient preconditioner for smooth problems is the multi-level (multigrid) overlapping domain decomposition (block Gauss–Seidel) method. On the other hand, the most robust preconditioner turned out to be the multi-level block ILU(0) method that works well also for problems with large jumps in coefficients. Both methods can be combined with special domain decomposition strategies to guarantee convergence for problems on irregular, anisotropic meshes.

References

[1] D. N. Arnold, F. Brezzi, B. Cockburn, D. Marini. Unified analysis of discontinuous Galerkin methods for elliptic problems. *SIAM Journal on Numerical Analysis*, **39:** 1749–1779, 2001/2002.

[2] X. Feng, O. A. Karakashian. Two-level additive Schwarz methods for a discontinuous Galerkin approximation of second order elliptic problems. *SIAM Journal on Numerical Analysis*, **39:** 1343–1365, 2001.

[3] F. Bassi, S. Rebay. GMRES discontinuous Galerkin solution of the compressible Navier-Stokes equations. In *Discontinuous Galerkin methods*, (Eds.), B. Cockburn, B. E. Karniadakis, C. W. Shu, volume 11 of *Lecture Notes in Computational Science and Engineering, Springer Verlag*, 197–208, Berlin, 2000.

[4] J. T. Oden, I. Babuska, C. E. Baumann. A discontinous hp finite element method for diffusion problems. *Journal of Computational Physics*, **146:** 491–519, 1998.

[5] M. F. Wheeler B. Rivière, V. Girault. Improved energy estimates for interior penalty, constrained and discontinuous Galerkin methods for elliptic problems, Part I. *Computational Geosciences*, **3:** 337–360, 1999.

[6] J. Douglas, T. Dupont. Interior penalty procedures for elliptic and parabolic Galerkin methods. volume 58 of *Lecture Notes in Physics, Springer-Verlag*, Berlin, 1976.

[7] M. F. Wheeler. An elliptic collocation-finite element method with interior penalties. *SIAM Journal on Numerical Analysis*, **15:** 152–161, 1978.

[8] Béatrice Rivière. Discontinuous Galerkin methods for solving the miscible displacement problem in porous media. *Ph.D. dissertation*, The University of Texas at Austin, 2000.

[9] B. Rivière, M. F. Wheeler, K. Banaś. Discontinuous Galerkin method applied to a single phase flow in porous media. *Computational Geosciences*, **4:** 337–349, 2000.

[10] Y. Saad, M. Schultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal of Scientific and Statistical Computing*, **7:** 856–869, 1986.

[11] G. Golub, J. M. Ortega. Scientific Computing with Introduction to Parallel Computing. *Academic Press*, San Diego, 1993.

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

[13] P. LeTallec. Domain decomposition method in computational mechanics. *Computational Mechanics Advances*, J. T. Oden (Ed.), North Holland, Amsterdam, 1994.

[14] B. Smith, P. Bjorstad, W. Gropp. Domain decomposition. Parallel multilevel methods for elliptic partial differnetial equation. *Cambridge University Press*, Cambridge, 1996.

[15] P. Bastian, V. Reichenberger. Multigrid for higher order discontinuous Galerkin finite elements applied to groundwater flow. Preprint 2000-37, Interdisziplinäres Zentrum für Wissenschaftliches Rechnen, University of Heidelberg, 2000.

[16] W. Hackbusch. Multigrid methods and applications. *Springer-Verlag*, Berlin, 1985.

[17] G. Wittum. On the robustness of ILU smoothing. *SIAM journal of scientific and statistical computing*, **10:** 699–717, 1989.

[18] W. Rachowicz. An overlapping domain decomposition preconditioner for an anisotropic h-adaptive finite element method. *Computer Methods in Applied Mechanics and Engineering*, **127:** 269–292, 1995.