INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS
Int. J. Numer. Meth. Fluids 2007; **55**:241–261
Published online 13 February 2007 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/fld.1450

Numbering techniques for preconditioners in iterative solvers for compressible flows

B. Pollul*, † and A. Reusken

Institut für Geometrie und Praktische Mathematik, RWTH Aachen University, D-52056 Aachen, Germany

SUMMARY

We consider Newton–Krylov methods for solving discretized compressible Euler equations. A good preconditioner in the Krylov subspace method is crucial for the efficiency of the solver. In this paper we consider a point-block Gauss–Seidel method as preconditioner. We describe and compare renumbering strategies that aim at improving the quality of this preconditioner. A variant of reordering methods known from multigrid for convection-dominated elliptic problems is introduced. This reordering algorithm is essentially black-box and significantly improves the robustness and efficiency of the point-block Gauss–Seidel preconditioner. Results of numerical experiments using the QUADFLOW solver and the PETSc library are given. Copyright © 2007 John Wiley & Sons, Ltd.

Received 2 March 2006; Revised 5 October 2006; Accepted 21 November 2006

KEY WORDS: Euler equations; Krylov subspace methods; preconditioning; ordering algorithms

1. INTRODUCTION

We are interested in efficient numerical techniques for the numerical simulation of two- and three-dimensional compressible flows. One important issue in this field is the solution of large sparse nonlinear systems of equations that arise after spatial discretization combined with an implicit time integration method. Two popular approaches for solving such nonlinear systems of equations are nonlinear multigrid solvers and Newton–Krylov methods. Well-known nonlinear multigrid techniques are the FAS method by Brandt [1], the nonlinear multigrid method by Hackbusch [2] and

*Correspondence to: B. Pollul, Institut für Geometrie und Praktische Mathematik, RWTH Aachen University, D-52056 Aachen, Germany.
†E-mail: pollul@igpm.rwth-aachen.de

Contract/grant sponsor: German Research Foundation

the algorithm introduced by Jameson [3]. It has been shown that a nonlinear multigrid approach can result in very efficient solvers which can even have optimal complexity for a certain class of problems [4–7]. Multigrid methods, however, require a coarse-to-fine grid hierarchy, whereas Newton–Krylov algorithms only need the matrix of the linearized system. Due to this and the fact that efficient implementations of many (preconditioned) Krylov subspace algorithms in sparse matrix libraries are available, the Newton–Krylov algorithms are in general much easier to implement than multigrid solvers. Furthermore, concerning efficiency, the Newton–Krylov approach with appropriate preconditioning can be competitive with multigrid. Thus, it is not surprising that Newton–Krylov techniques are often used in practice (cf. for instance, [8–13]).

In this paper we consider the Newton–Krylov approach. A method of this class has been implemented in the QUADFLOW package, which is an adaptive multiscale finite volume solver for stationary and instationary compressible flow computations. Descriptions of this solver are given in [14–18]. For the linearization we use a standard (approximate) Newton method. The resulting linear systems are solved by a preconditioned BiCGSTAB method using methods implemented in the PETSc-Library [19, 20].

Incomplete lower-upper (ILU)-factorization and Gauss–Seidel techniques are popular preconditioners that are used in solvers in the numerical simulation of compressible flows [12, 21–23]. The 'point-block'-variants of these preconditioners are obtained by applying the original point versions to the blocks of unknowns corresponding to each cell. Both preconditioners depend on the ordering of the cells (grid-points) [12, 22, 24–28]. In combination with PBILU the reverse Cuthill–McKee ordering algorithm [29, 30] is often used. This ordering yields a matrix with a 'small' bandwidth.

In this paper we focus on ordering algorithms for the point-block-Gauss–Seidel (PBGS) preconditioner. We do not know of any literature that deals with ordering techniques for Gauss–Seidel preconditioners applied to linearized Euler equations. The ordering algorithms consist of three steps. First a weighted directed graph, in which every vertex corresponds to a block unknown, is constructed. Then this graph is reduced by deleting edges with relatively small weights. This graph reduction is very similar to techniques used in algebraic multigrid methods [31]. Finally the renumbering of the vertices in this reduced graph is determined. For this we consider three different algorithms. Two of them are known (due to Bey and Wittum [24] and Hackbusch [25, 32]) from the field of robust multigrid solvers for convection-dominated elliptic problems. The third one is new. These methods are implemented in the QUADFLOW solver using the PETSc library. A systematic comparative study shows that for our problem class the new variant yields the best results. The reordering algorithm is essentially black-box. Using this reordering we can improve the robustness of the iterative solver: For large Courant–Friedrichs–Lewy (CFL) numbers we encounter linear systems for which the BiCGSTAB method with PBGS preconditioner converges only if we first apply the reordering. Using the reordering we can also improve the efficiency of the linear solver significantly. The execution time of the iterative solver part can be reduced by 10% (for complex transonic flows) up to 50% (for supersonic flows).

The remainder of this paper is organized as follows. In the following section we outline the discretization and linearization methods that are used in QUADFLOW for the numerical solution of the compressible Euler equations. In Section 3 we describe the PBGS preconditioner. Section 4 gives a detailed description of three renumbering algorithms. In Section 5 we apply these algorithms, implemented in the QUADFLOW solver, to some test problems. Finally we summarize some main results of the paper (Section 6).

DOI: 10.1002/fld

2. DISCRETE EULER EQUATIONS

We consider the conservative formulation of the Euler equations. For an arbitrary control volume $V \subset \Omega \subset \mathbb{R}^d$ $(d = 2, 3)$ one has equations of the form

$$\int_V \frac{\partial \mathbf{u}}{\partial t} \, \mathrm{d}V + \oint_{\partial V} \mathbf{F}^c(\mathbf{u}) \mathbf{n} \, \mathrm{d}S = 0 \tag{1}$$

Here, $\mathbf{n}$ is the outward unit normal on ∂V and $\mathbf{u} = (\rho, \rho\mathbf{v}, \rho e_{\mathrm{tot}})^{\mathrm{T}}$ the vector of unknown conserved quantities consisting of density ρ, vector of velocities $\mathbf{v}$ and total energy e_{tot}. The convective flux is given by

$$\mathbf{F}^c(\mathbf{u}) = \begin{pmatrix} \rho\mathbf{v} \\ \rho\mathbf{v} \circ \mathbf{v} + p\mathbf{I} \\ \rho h_{\mathrm{tot}} \mathbf{v} \end{pmatrix} \tag{2}$$

The symbol $\circ$ denotes the dyadic product and h_{tot} is the total enthalpy. The system is closed by the equation of state for a perfect gas and suitable initial and boundary conditions. For the numerical simulation of the compressible Euler equations we use the software package QUADFLOW, which is currently under development at Aachen University, cf. [14–18]. We briefly describe a few main features of this solver. QUADFLOW contains methods for the numerical simulation of two- and three-dimensional compressible Euler and Navier–Stokes equations. It is based on block-structured grids. The geometry of these blocks is described using tensor-product B-splines. For discretization finite volume techniques are applied. Several upwind methods, for instance flux-difference splitting (HLLC [33]), flux-vector splitting (van Leer [34], Hänel and Schwane [35]) and AUSDMV(p) [36, 37] have been implemented. A key ingredient in QUADFLOW is the use of local grid refinement in regions of high activity, for example in the neighbourhood of shocks. Both explicit and implicit time integration routines are available. The computation of an accurate approximation of a stationary solution is based on a nested iteration approach. One starts with an initial coarse grid and an initial CFL number γ_0, which determines the size of the time step. After each time step in the time integration the CFL number (and thus the time step) is increased by a constant factor until an *a-priori* fixed upper bound $\gamma_{\max}$ is reached. Time integration is continued until a tolerance criterion for the residual is satisfied. Then a (local) grid refinement is performed and the procedure starts again with an interpolated initial condition and a starting CFL number equal to γ_0. The indicator for the local grid refinement is based on a multiscale analysis using wavelets. The nonlinear systems that arise in each time step of an implicit method are solved using a Newton–Krylov approach. In every time step one approximate Newton iteration is performed. The resulting linear equations are solved using preconditioned Krylov-subspace methods that are available the PETSc library [19, 20]. For this an interface between QUADFLOW and PETSc has been developed. A first parallel version of QUADFLOW is available now. In this paper, however, we restrict ourselves to the sequential version. To give an impression of the multiblock and adaptivity features of QUADFLOW we show grids that are used in a simulation of an inviscous flow around a BAC 3-11/RES/30/21 airfoil (cf. Section 5.2) in Figure 1.

DOI: 10.1002/fld

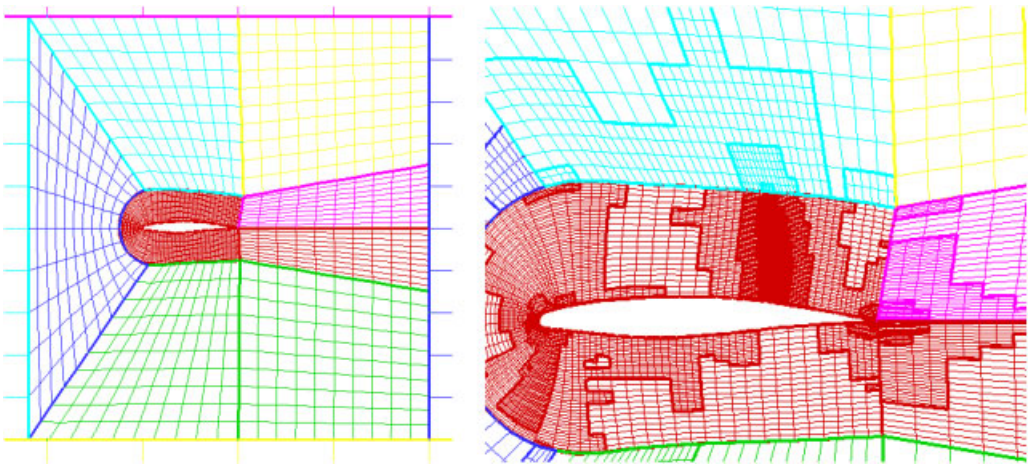

Figure 1. 12-block grid of BAC 3-11/RES/30/21 airfoil: part of the original grid and of the grid after 10 levels of local refinement. Test problem 2, cf. Section 5.2.

In most simulations an implicit time integration is used. Then the computational work for solving the large sparse systems in the Newton–Krylov method determines to a large extent the total computing time in a simulation run. Hence, the efficiency of the iterative solvers for these systems is an important issue. In general for stationary problems this issue plays a bigger role than for non-stationary problems. We therefore focus on stationary problems in this paper.

For the discretization we choose methods that are available in QUADFLOW. For spatial discretization the flux-vector splitting by Hänel and Schwane [35] is applied. A linear reconstruction technique is used to obtain second-order accuracy in regions where the solution is smooth. This is combined with the Venkatakrishnan limiter [38]. Although we are interested in stationary solutions the time derivative is not skipped. This time derivative is discretized by a numerical integration method which then results in a numerical method for approximating the stationary solution. To obtain fast convergence towards the stationary solution one wants to use large time steps and thus an implicit time discretization method is preferred. We use the b2-scheme by Batten *et al.* [39]. This approach then results in a nonlinear system of equations in each time step. Per time step one inexact Newton iteration is applied. In this inexact Newton method an *approximate* Jacobian is used in which the linear reconstruction technique is neglected and the Jacobian of the first-order Hänel–Schwane discretization is approximated by one-sided difference operators (as in [40]). These Jacobian matrices have the structure

$$\mathrm{DF}(\mathbf{U}) = \mathrm{diag}\left(\frac{|V_i|}{\Delta t}\right) + \frac{\partial R^{\mathrm{HS}}(\mathbf{U})}{\partial \mathbf{U}} \tag{3}$$

where $|V_i|$ is the volume of cell V_i, Δt the (local) time step and $R^{\mathrm{HS}}(\mathbf{U})$ the residual vector corresponding to the Hänel–Schwane fluxes. Details are given in [17]. Note that in general a smaller time step will improve the condition number of the approximate Jacobian in (3).

In this paper we introduce and compare several renumbering techniques that aim at improving the efficiency of preconditioned Krylov subspace methods for solving these linear systems in the approximate Newton linearization. We emphasize that for these ordering techniques the particular

DOI: 10.1002/fld

choice of discretization components is not essential. The renumbering methods show a similar behaviour, if instead of the Hänel–Schwane method, one uses another upwind method (see above), or if, instead of the Batten b2-scheme, one uses another implicit time integration method.

3. POINT-BLOCK-GAUSS–SEIDEL PRECONDITIONER

The approximate Newton method described above leads to large sparse linear systems of equations. For solving these systems we use a standard preconditioned Krylov subspace method, available in PETSc. We choose BiCGSTAB with a PBGS preconditioner. We briefly explain the latter.

On a computational grid as in Figure 1 there are $d+2$ unknowns per cell, cf. (1)–(2). We call the small $(d+2)\times(d+2)$ blocks occurring in the approximate Jacobian 'point-blocks'. If the cells are numbered $i=1,\ldots,N$, then the approximate Jacobian has a point-block structure $\mathrm{DF}(\mathbf{U})=\mathrm{blockmatrix}(A_{i,j})_{0\leqslant i,j\leqslant N}$ with $A_{i,j}\in\mathbb{R}^{(d+2)\times(d+2)}$ for all i, j and $A_{i,j}\neq 0$ only if $i=j$ or i and j correspond to neighbouring cells. Thus, we have linear systems of the form

$$\mathbf{A}\mathbf{x}=\mathbf{b},\quad \mathbf{A}=\mathrm{blockmatrix}(A_{i,j})_{1\leqslant i,j\leqslant N},\quad A_{i,j}\in\mathbb{R}^{(d+2)\times(d+2)} \tag{4}$$

For the right-hand side we use a block representation $\mathbf{b}=(b_1,\ldots,b_N)^{\mathrm{T}}$, $b_i\in\mathbb{R}^{d+2}$, that corresponds to the block structure of $\mathbf{A}$. The same is done for the iterands $\mathbf{x}^k$ that approximate the solution of the linear system in (4). The PBGS method is the standard block Gauss–Seidel method applied to (4). Let $\mathbf{x}^0$ be a given starting vector. For $k\geqslant 0$ the iterand $\mathbf{x}^{k+1}=(x_1^{k+1},\ldots,x_N^{k+1})^{\mathrm{T}}$ should satisfy

$$A_{i,i}x_i^{k+1}=b_i-\sum_{j=1}^{i-1}A_{i,j}x_j^{k+1}-\sum_{j=i+1}^{N}A_{i,j}x_j^{k},\quad i=1,\ldots,N \tag{5}$$

This method is well-defined if the $(d+2)\times(d+2)$ linear systems in (5) are uniquely solvable, i.e. if the diagonal blocks $A_{i,i}$ are non-singular. In our applications this was always the case. This elementary method is very easy to implement and needs no additional storage. The algorithm is available in the PETSc library [19, 20].

4. RENUMBERING TECHNIQUES

Incomplete LU-decomposition and Gauss–Seidel techniques are often used for preconditioning Krylov subspace methods applied to linear systems that arise in numerical simulations of compressible flows (cf. [12, 21–23]). Both preconditioners depend on the ordering of the cells (points) [12, 24–28]. This holds for the point-block variants point-block-ILU (PBILU) and PBGS, too. There are many studies available on numbering techniques for ILU preconditioners (cf. [22, 41] and references therein). For PBILU a reverse Cuthill–McKee ordering algorithm [29, 30] often leads to good results. This ordering yields a matrix with a 'small' bandwidth which is favourable for PBILU. Such PBILU methods combined with reordering techniques are often used in iterative solvers for compressible flow problems. A PBGS preconditioner is particularly useful in parallel and/or matrix-free iterative solvers. As for PBILU this preconditioner can be improved significantly by reordering techniques. For PBGS the ordering should be such that one approximately follows the directions in which information is propagated. In this section we introduce

DOI: 10.1002/fld

three renumbering methods that aim at realizing this. The first two of these algorithms are from the field of robust multigrid methods for convection-dominated problems and are due to Bey and Wittum [24] and Hackbusch [25]. The third one is a new variant, which for our applications turns out to be better.

All three algorithms are completely matrix-based, in the sense that one needs as input only the block-structured matrix from (4). In these algorithms we distinguish the following three steps:

1. *Construct a weighted directed matrix graph* in which every vertex corresponds to a block unknown and each edge to a non-zero off-diagonal block of the given matrix $\mathbf{A}$.
2. *Construct a reduced weighted directed matrix graph*. The reduction is obtained by deleting edges with relatively small weights.
3. *Determine a renumbering of the vertices*, based on the reduced weighted matrix graph. This provides a point-block-permutation of the given matrix $\mathbf{A}$.

While for all three algorithms presented below steps 1 and 2 are identical, they differ in the methods used in step 3. We explain these first two steps in Sections 4.1 and 4.2.

4.1. Construction of weighted directed matrix graph $\mathscr{G}(\mathbf{A})$

We introduce standard notation related to matrix graphs. Let $\mathscr{V} = \{1, \ldots, N\}$ be a vertex set (each vertex corresponds to a discretization cell). The set of edges $\mathscr{E}$ contains all directed edges

$$\mathscr{E} = \{(i, j) \in \mathscr{V} \times \mathscr{V} \mid A_{i,j} \neq 0, i \neq j\} \tag{6}$$

Note that $\mathscr{E}$ does not contain edges (i, i). The mapping

$$\omega : \mathscr{E} \to (0, \infty) \tag{7}$$

assigns to every directed edge $(i, j) \in \mathscr{E}$ a weight

$$\omega_{ij} := \omega(i, j) := \|A_{i,j}\|_F \tag{8}$$

We take the Frobenius-norm because it is easy to compute and all entries in a block $A_{i,j}$ are weighted equally. This yields a *weighted directed* matrix graph $\mathscr{G} = \mathscr{G}(\mathbf{A})$

$$\mathscr{G}(\mathbf{A}) := (\mathscr{V}, \mathscr{E}, \omega) \tag{9}$$

In opposition to the commonly used definition we call an edge $(i, j) \in \mathscr{E}$ an *inflow edge* of vertex $i \in \mathscr{V}$ and an *outflow edge* of vertex $j \in \mathscr{V}$. This is motivated by the following: an edge (i, j) in the graph corresponds to a flow *from* cell j *into* cell i in the underlying physical problem.

Consequently, for $(i, j) \in \mathscr{E}$ we call j a *predecessor* of i and i a *successor* of j. The set of predecessors of vertex $i \in \mathscr{V}$ is denoted by

$$\mathscr{I}_i := \{j \in \mathscr{V} \mid (i, j) \in \mathscr{E}\} \tag{10}$$

In the construction of $\mathscr{G}(\mathbf{A})$ one only has to compute the weights ω_{ij} in (8). For storage of this information we use a sparse matrix format. Note that the size of the sparse matrix corresponding to $\mathscr{G}(\mathbf{A})$ is $N \times N$ (and not $Nd \times Nd$, as for $\mathbf{A}$). Hence, the costs both for the computation and the storage of $\mathscr{G}(\mathbf{A})$ are low.

DOI: 10.1002/fld

4.2. Construction of reduced matrix graph $\hat{\mathscr{G}}$

Based on reduction techniques from algebraic multigrid methods in which *strong couplings* and *weak couplings* are distinguished [31, 42, 43], we separate *strong edges* from *weak edges*. For every vertex $i \in \mathscr{V}$ we neglect all inflow edges $(i, j) \in \mathscr{E}$ with a weight smaller than τ-times the average of the weights of all inflow edges of vertex i. Thus, we obtain a reduced set of *strong* edges $\hat{\mathscr{E}}$ and a corresponding reduced (weighted directed) graph $\hat{\mathscr{G}}(\mathbf{A})$

$$\sigma_i := \frac{1}{|\mathscr{I}_i|} \sum_{j \in \mathscr{I}_i} \omega_{ij} \tag{11}$$

$$\hat{\mathscr{E}} := \{(i, j) \in \mathscr{E} \mid \omega_{ij} \geqslant \tau \cdot \sigma_i\} \tag{12}$$

$$\hat{\mathscr{G}}(\mathbf{A}) := (\mathscr{V}, \hat{\mathscr{E}}, \omega_{|\hat{\mathscr{E}}}) \tag{13}$$

This simple construction of a reduced matrix graph $\hat{\mathscr{G}}(\mathbf{A})$ can be realized with low computational costs. In the rest of the reordering method we do not need $\mathscr{G}(\mathbf{A})$ anymore, and thus we do not need additional storage because we can overwrite $\mathscr{G}(\mathbf{A})$ with $\hat{\mathscr{G}}(\mathbf{A})$.

The use of graph reduction is essential for the performance of the reordering techniques discussed below. Note that the parameter τ controls the size of the reduced graph: for $\tau = 0$ there is no reduction of the original graph, whereas for $\tau \to \infty$ the reduced graph contains only vertices and no edges. The choice of an appropriate value for the parameter τ is discussed in Section 5.1. In particular it will be shown that the performance of the numbering techniques is not sensitive w.r.t. perturbations of the parameter value.

In the following sections we present three different methods that are used in step 3, resulting in three different ordering algorithms.

4.3. Downwind numbering based on $(\mathscr{V}, \hat{\mathscr{E}})$ (Bey and Wittum)

A numbering algorithm due to Bey and Wittum (Algorithm 4.3 in [24]) is presented in Figure 2 and denoted by 'BW'. It is used in multigrid methods for scalar convection–diffusion problems to construct so-called robust smoothers. To apply this algorithm for our class of problems we need the reduced directed graph $(\mathscr{V}, \hat{\mathscr{E}})$ as input. Note that the weights ω_{ij} are *not* used.

```
for all P ∈ V : Index(P) : = −1;
n_F := 1
for P ∈ V
   (if Index(P) < 0 ) SetF(P);
end P

procedure SetF(P)
   (if all predecessors B of P have Index(B) > 0 )
      Index(P) := n_F;
      n_F := n_F + 1;
      for Q successor of P
         if (Index(Q) < 0) SetF(Q);
      end Q
   end if
```

Figure 2. Downwind numbering algorithm BW.

DOI: 10.1002/fld

Remark 1
In the loop over $P \in \mathscr{V}$ in algorithm BW the ordering of the block-unknowns (cells) corresponding to the input matrix $\mathbf{A}$ is used. In the procedure SetF(P) a vertex is assigned the next number if all its predecessors have already been numbered. Hence, the first number is assigned to a vertex that has no inflow edges. Note that in the procedure SetF(P) there is freedom in the order in which the successors Q are processed. In our implementation we again use the ordering induced by the given matrix $\mathbf{A}$. The BW numbering is applied to the reduced matrix graph. If that graph is cycle-free the algorithm returns a renumbering that is optimal in the sense that this reordering applied to the matrix corresponding to $\hat{\mathscr{G}}(\mathbf{A})$ results in a lower triangular matrix. However, in our problem class the reduced graphs in general contain cycles. In that case, after algorithm BW has finished there still are vertices $P \in \mathscr{V}$ with Index(P) $= -1$, i.e. there are $N - n_F > 0$ vertices that have no (new) number. The numbers $n_F, \ldots, N$ are assigned to these remaining vertices in the order induced by the input matrix ordering. The two variants of BW that are treated below in general have fewer of such 'remaining' vertices.

Note that in this algorithm there are logical operations and assignments but no arithmetic operations.

4.4. Down- and upwind numbering based on $(\mathscr{V}, \hat{\mathscr{E}})$ (Hackbusch)

In Figure 3 we present an ordering algorithm, denoted by 'HB', that is due to Hackbusch [25]. As input for this algorithm one needs the reduced directed graph $(\mathscr{V}, \hat{\mathscr{E}})$ (*no* weights required). The presentation of this algorithm is as in Section 2.1 in [32]. The Routine 'SetF' is the same as in the BW algorithm in Figure 2.

Remark 2
In the BW algorithm the vertices are ordered in one direction, namely 'downwind' (in the 'flow direction'). The algorithm due to Hackbusch uses two directions: 'downwind' (SetF) and 'upwind' (SetL). In [25] and [32] techniques for handling cycles are presented. These techniques are rather complicated and often computationally expensive. In multigrid codes for convection-dominated problems one usually encounters the ordering algorithm HB as in Figure 3 which does not treat cycles. If the reduced matrix graph $(\mathscr{V}, \hat{\mathscr{E}})$ is not cycle-free there are remaining vertices. These

```
for all P ∈ V : Index(P) := -1;
n_F := 1; n_L := N;
for P ∈ V
  (if Index(P) < 0 ) SetF(P);
  (if Index(P) < 0 ) SetL(P);
end P

procedure SetL(P)
  (if all sucessors B of P have Index(B) > 0 )
    Index(P) := n_L;
    n_L := n_L -1;
    for Q predecessor of P
      if (Index(Q) < 0) SetL(Q);
    end Q
  end if
```

Figure 3. Down- and upwind numbering algorithm HB.

DOI: 10.1002/fld

```
for all P ∈ V : Index(P) := −1 ;
n_F := 1; n_L := N;

 /* (i) apply SetF and SetL to starting vertices */
do in an outflow-ordered list Ⓢ, S(Σ_i ω_ip, P): for P ∈ V                (14)
   (if Index(P) < 0 ) SetF(P, 1);
end P
do in an inflow-ordered list Ⓢ, S(Σ_j ω_pj, P): for P ∈ V                 (15)
   (if Index(P) < 0 ) SetL(P);
end P

 /* (ii) number remaining vertices */
do in an outflow-ordered list Ⓢ, S(Σ_i ω_ip, P): for P ∈ V                (16)
   (if Index(P) < 0 ) SetF(P, 0);
end P


procedure SetF(P, s)
   (if all predecessors B of P have Index(B) > 0 ) or (s = 0)
      Index(P) := n_F;
      n_F := n_F + 1;
      do in an outflow-ordered list Ⓢ, S(Σ_i ω_iq, Q): for Q successor of P   (17)
         if (Index(Q) < 0) SetF(Q, 1);
      end Q
   end if

procedure SetL(P)
   (if all sucessors B of P have Index(B) > 0 )
      Index(P) := n_L;
      n_L := n_L − 1;
      do in an inflow-ordered list Ⓢ, S(Σ_j ω_qj, Q): for Q predecessor of P   (18)
         if (Index(Q) < 0) SetL(Q);
      end Q
   end if

Ⓢ : p denotes the index of the vertex P of the input graph. S(Σ_i ω_ip, P) sorts the
vertices P descending in the corresponding values Σ_i ω_ip (similar for S(Σ_j ωpj, P)).
```

Figure 4. Weighted reduced graph numbering algorithm WRG.

are treated as described in Remark 1. The computational cost of algorithm HB is comparable to that of BW.

4.5. Weighted reduced graph numbering based on $(\mathscr{V}, \hat{\mathscr{E}}, \omega_{|\hat{\mathscr{E}}})$

In this section we present a modification of the methods of Bey, Wittum and Hackbusch. As input for our method we now need the *weighted* reduced graph $(\mathscr{V}, \hat{\mathscr{E}}, \omega_{|\hat{\mathscr{E}}})$.

The performance of the BW and HB numbering depend on the ordering of the input graph. We present an algorithm that uses the weights of the reduced graph to avoid the dependence on the initial ordering. The algorithm is denoted by 'WRG' and is given in Figure 4.

Remark 3

There are two important differences to the algorithms HB and BW. The first difference is related to the arbitrariness of the order in which the vertices are handled in the loops in HB and BW, cf. Remark 1. If there are different possibilities for which vertex is to be handled next we now use the

DOI: 10.1002/fld

weights ω_{ij} of the reduced graph to make a decision. This decision is guided by the principle that edges with larger weights are declared to be more important than those with relatively small weights. A weight based sorting occurs at several places, namely in (14)–(18). In (14) the vertices with no inflow edges are sorted ('starting' vertices) using the sum of the weights of the outflow edges at each vertex. Similarly, in (15) the vertices with no outflow edges are sorted. The 'remaining' vertices are finally sorted based on the sum of the outflow edges at each vertex in (16). In all three cases the number of vertices to be sorted is much smaller than N and thus the time for sorting is acceptable. Sorting is also used in (17) and (18) to determine the order in which successors and predecessors are handled. In SetF$(\cdot,\cdot)$ the successors Q of the current P are sorted using the sum over the weights of all outflow edges for each Q. This is done similarly in SetL$(\cdot)$ for all predecessors of the current P.

The second difference is that the loop over the numbering routine SetF is called *two* times. The first call SetF$(P,1)$ in part `(i)` of algorithm WRG is similar to the call of SetF(P) in the algorithms BW and HB but now with an ordering procedure used in SetF. The second call SetF$(P,0)$ (in part `(ii)` in WRG) is introduced to handle the remaining vertices that still have index value -1. In this call we do not consider the status of inflow edges and continue numbering in downwind direction (SetF$(\cdot,0)$). The inner call SetF$(Q,1)$ to number the successors still requires that all predecessors have been numbered. After part `(ii)` of the algorithm is finished the only possibly not yet numbered vertices are trivial ones, in the sense that these are vertices that have no edges to other vertices. Note that part `(i)` can be also obtained by applying HB to a *a priori* sorted graph. Step `(ii)` in WRG does not have a counterpart in HB.

Due to the additional sorting routines in (14)–(18) the computational costs of the renumbering algorithm WRG are higher than of those BW and HB. However, if we use algorithm WRG in step 3 the total time needed for the execution of the steps 1–3 is still acceptable, cf. Remark 4.

Remark 4

As indicated in our comments above, in all three algorithms the computational time that is needed and the storage requirements are modest compared to other components of the iterative solver. Of course this will not be true for general matrices but it does hold for the class of large sparse point-block-matrices that forms our problem class. In our pseudo-time integration we have a sequence of time steps on every level of adaptation. The time needed for solving the linear systems is typically increasing during the discrete time integration. This is due to the increase of the CFL number, cf. Section 2. Since the Jacobian matrices of consecutive time steps are in some sense similar we apply reordering not in each iteration but only 'now and then' and keep it for the subsequent time steps, cf. Section 5. Thus, the total execution time for the reordering routines is very small compared to the total time needed for the linear solves with the preconditioned Krylov-subspace method. In our test problems the reordering routines consume at most a few percent of the total execution time of the iterative solver.

In the experiments in Section 5 the time for reducing the graph and determining the reordering was smaller than the average time for solving the system of linear equations in one time step. From numerical examples, cf. Section 5.1, it turns out that reasonable values are $\tau \in [0.75, 2.00]$, cf. (12). For such values WRG numbering needs between 10 and 50% more time than the other numbering techniques, depending on the value for the grid reduction parameter τ.

Both the computational costs and the quality of the reordering algorithm depend on the value for τ. For large τ-values the reduced set of edges $\hat{\mathscr{E}}$ contains only few elements and thus the reduced

DOI: 10.1002/fld

graph $\hat{\mathscr{G}}(\mathbf{A})$ is close to a trivial one. The computational costs for constructing the corresponding renumbering (step 3) are relatively low but the resulting renumbering will in general hardly improve the quality of the PBGS preconditioner. The choice of the value for the graph reduction parameter τ is discussed in Section 5.

5. NUMERICAL EXPERIMENTS

In this section we present results of numerical experiments. We will illustrate the behaviour of the different numberings presented above for a few test problems.

In all experiments below we use a left preconditioned BiCGSTAB method. The approximate Jacobian matrices as in (3) are computed in QUADFLOW. For the preconditioned BiCGSTAB method and the PBGS preconditioner we use routines from the PETSc library [19, 20]. As described in Section 2, in the time integration on a given discretization level the CFL number is increased, aiming at fast convergence towards the stationary solution. In QUADFLOW the default strategy for determining this CFL number γ_k in the kth time step is as follows: $\gamma_k = \min\{\gamma_0\gamma^k, \gamma_{\max}\}$. In all experiments we set $\gamma_0 = 1$, $\gamma = 1.1$ (default values in QUADFLOW). We continue time integration on every discretization level until the residual of the density has been reduced by a factor 10^2. On the finest discretization level we require a reduction by a factor 10^4. The number of discretization levels used depends on the problem and on certain parameters used in the adaptive refinement strategy.

In a typical computation most time is spent on solving the linear equation systems on the grid that corresponds to the finest level of adaptation. Therefore, we present the number of iterations of the preconditioned BiCGSTAB method that is needed to reduce the starting residual of the linear (Jacobian) system by a factor 10^4 on the finest grid in order to measure the quality of the renumberings. We compare four different numberings. The BW, HB and WRG methods have been explained above. The fourth numbering is the one induced by the discretization routines in QUADFLOW and is denoted by QN. One central feature of the QUADFLOW solver is the multiscale analysis that is used for error estimation and induces local refinement. This results in a hierarchy of locally refined grids, cf. Section 2. In this process the cells are numbered levelwise from the coarsest to the finest level. This leads to a sort of hierarchical block-structure of the matrix. A typical pattern of the Jacobian is shown in Figure 5. In Figure 8(e) a typical computational grid is given. Every cell has a colour representing its number induced by the multiscale analysis (QN numbering). The figure shows that the numbers are given blockwise and levelwise: Cells that a generated at higher levels in the local refinement procedure obtain higher numbers.

After a prolongation to the next finer level in the nested iteration method we perform a renumbering after the first time step. In each of the following time steps we have a new Jacobian system to which a renumbering algorithm can be applied. For efficiency reasons we do *not* apply the renumbering method (steps 1–3) to every new Jacobian but use the known renumbering as computed in the first time step. We determine a new renumbering only after every k_r time steps. Typical values for k_r are $k_r = 10, \infty$. All three numbering techniques are sensitive with respect to the choice of the value for the parameter τ. In our sub- and supersonic problems $\tau = 1.25$ turned out to be agood default value. In transonic problems the performance can often be improved by taking a somewhat large τ-value (e.g. $\tau = 2.00$). At the end of the next subsection we confirm the choice of the value for τ by an experiment.

DOI: 10.1002/fld

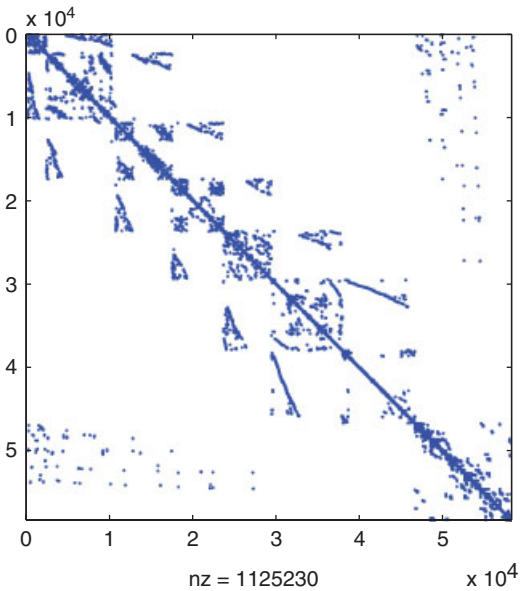

Figure 5. Test problem 2: non-zero pattern of the matrix on the finest level (BAC 3-11/RES/30/21).

Table I. Test cases 1A, 1B, 1C: Mach number M_∞ and angle of attack α for NACA0012 airfoil.

	M_∞	α (deg.)
Test case 1A	0.80	1.25
Test case 1B	0.95	0.00
Test case 1C	1.20	0.00

5.1. Test problem 1: stationary flow around NACA0012 airfoil

The first problem is a standard test case for inviscid compressible flow solvers. We consider the inviscid, transonic stationary flow around the NACA0012 airfoil (cf. [44], see also Table I). In this section we present some results for three test cases as shown in Table I.

Results of a numerical simulation for case 1B are shown in Figure 6. Renumbering is applied only once after every prolongation to the next finer discretization level ($k_r = \infty$). The maximum CFL number was set to $\gamma_{\max} = 1000$. Computations are done as in [17]: We allow eight maximum levels of refinement. In the cases 1A and 1C 10 cycles of adaptations are performed, 13 levels are used in case 1B.

Tables II–IV show the average iteration count on the finest level for the different orderings. The average is taken over all time steps used on the finest discretization level. The savings compared to the original QUADFLOW numbering QN (cf. above in Section 5) are displayed in the last row. In all three cases the savings were not improved significantly when using smaller k_r values.

DOI: 10.1002/fld

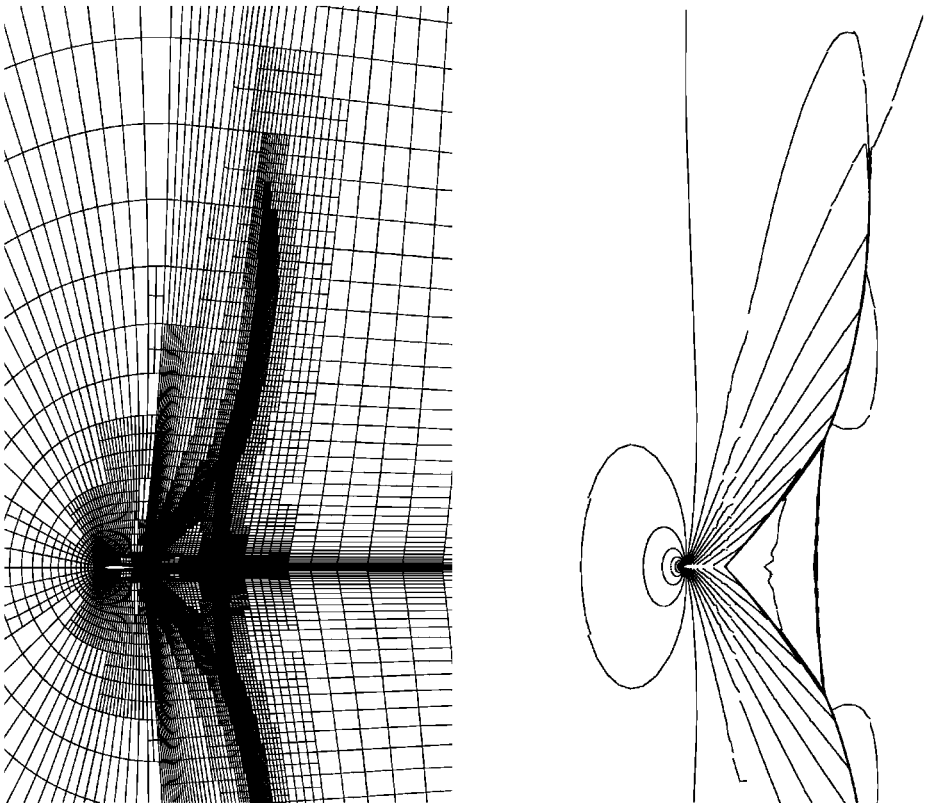

Figure 6. Case B: computational grid (left) and Mach distribution (right), $M_{\min} = 0.0$, $M_{\max} = 1.45$.

Table II. Case 1A: average iteration count on finest level (10th discretization level).

Numbering	QN	BW	HB	WRG
Average iteration count	32.0	30.6	28.6	23.0
Saving (%)	0	4.4	10.6	28.1

Table III. Case 1B: average iteration count on finest level (13th discretization level).

Numbering	QN	BW	HB	WRG
Average iteration count	20.2	20.1	18.2	18.4
Saving (%)	0	0.5	9.9	8.9

Table IV. Case 1C: average iteration count on finest level (10th discretization level).

Numbering	QN	BW	HB	WRG
Average iteration count	24.2	12.5	12.6	10.9
Saving (%)	0	48.3	47.9	55.0

DOI: 10.1002/fld

Table V. Case 1C: finest computational grid: experiment with different values for the grid reduction parameter τ. Number of cells that were numbered in steps (i) and (ii) in WRG algorithm, cf. Figure 4.

Step of WRG	(i)	(ii)	Step of WRG	(i)	(ii)
$\tau \leqslant 0.75$	0	40 213	$\tau = 1.75$	40 207	6
$\tau = 1.00$	11 153	29 060	$\tau = 2.00$	40 211	2
$\tau = 1.25$	39 869	344	$\tau = 2.25$	40 211	2
$\tau = 1.50$	40 205	8	$\tau \geqslant 2.50$	40 213	0

In all cases the reduced matrix graph was constructed with $\tau = 1.25$. With the WRG renumbering method we save between 9 and 55% of PBGS-preconditioned BiCGSTAB iterations on the finest level compared to the original numbering QN. Since the renumbering has to be computed only once ($k_r = \infty$) the additional computational costs for WRG are negligible. The improvement is strongest for case 1C, which is due to the fact that in this case the flow is almost supersonic and thus there is a main stream in which information is transported. A further increase of the Mach number would lead to a decrease of the number of iterations and would improve the profit of the renumbering techniques as well.

In cases with higher CFL numbers $\gamma_{\max}$ the linear systems are in general harder to solve and the importance of an improvement due to a better numbering increases.

For test case 1C we illustrate the dependence of the iteration count on the graph reduction parameter τ. In Figure 9 the results for $\tau = 0.25 \cdot k, k = 0, 1, \ldots, 12$ are given. The dashed line (right y-axis) shows that the number of edges in the corresponding reduced graph of the Jacobian is decreasing monotonically if the value of τ is increased. Table V shows the number of vertices renumbered in each of the steps (i) and (ii) in the WRG algorithm, cf. Figure 4. For values $\tau \leqslant 0.75$ the reduced graph is too complex so that in the first step, cf. (i) in Figure 4, none of the 40 213 vertices is given a new number. On the other hand with $\tau = 1.50$ 98% of the vertices are given a new number in the first step (i), so that a further increase of the value for τ would be counterproductive. These results illustrate a general phenomenon: Increasing the value of τ more and more cycles split up and most of the vertices can be renumbered in the first step (i) of the WRG algorithm. However, too large values of τ lead to too many isolated vertices. An appropriate choice of the value for τ has to find a compromise between these two conflicting effects. Fortunately, the solid line in Figure 9 representing the performance of the preconditioned BiCGSTAB method indicates that the choice of the value for τ is not very sensitive. For this test case values $0.75 \leqslant \tau \leqslant 2.00$ all give quite good results. We obtain the best results for $\tau = 1.25$; in this case the reduced graph $\hat{\mathscr{G}}(\mathbf{A})$ can be reordered so that it is nearly a lower-diagonal graph as shown in Figure 7.

This ordering induces a renumbering of all vertices in the original graph $\mathscr{G}(\mathbf{A})$. Note that in the original graph there are no edges deleted. The reordered Jacobian contains precisely as many non-zero entries as the original Jacobian. In general both have a symmetric pattern. The effect of the reordering is that the dominant entries of the reordered Jacobian lie mostly in the lower diagonal part. It should be noted that such reordering techniques can only be effective for point-block matrices in which there is a significant difference between $\|A_{i,j}\|_F$ and $\|A_{j,i}\|_F$ for most i, j with $\|A_{i,j}\|_F \neq 0$.

The dependence of the results on the value for τ holds for all tested cases and for the other numbering algorithms. Only in problems with $M_\infty \approx 1$ the vertices of the graphs are stronger

DOI: 10.1002/fld

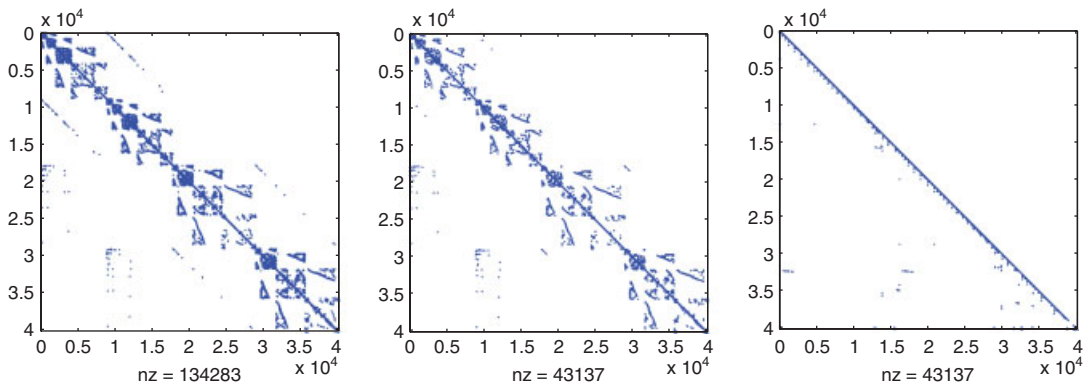

Figure 7. Test case 1C: graph $\mathcal{G}(\mathbf{A})$, reduced graph $\hat{\mathcal{G}}(\mathbf{A})$ and renumbered reduced graph of Jacobian matrix on finest grid, $\tau = 1.25$.

coupled so that a higher value for τ may be needed to split up an adequate amount of the cycles. That is why in case 1B the results for WRG numbering can be improved by a stronger reduction of the graph. With $\tau = 2.00$ the saving with WRG is about 21% (instead of 9%). In this transonic case the pattern of directions in which information is propagated has a more complex structure than in the other cases. Therefore, the savings are less than in the other examples. We want to point out that the ordering QN induced by the QUADFLOW discretization routines is already quite good. If namely a (point-block) *random* numbering is used, then the PBGS preconditioned BiCGSTAB method turns out to diverge in all tested cases, even when computing supersonic flow.

To further illustrate the effect of the renumbering algorithms we take a closer look at which number is given to which cell of the computational domain. In Figure 8 the complete computational domain with the finest grid is given for test case 1C. Figure 8(e) shows the QN numbering, that is the standard QUADFLOW numbering induced by the multiscale analysis. Every cell has a colouring indicating its number. Multiblock structure and effects of the adaptation can be recognized. The fact that the cells generated by the adaptation have higher cell numbers turns out to be a reasonably good choice. Figure 8(a)–(d) show the numbering generated by WRG for different values of τ. As already seen from Figure 9 the effect of the numbering is best for $\tau = 1.25$. In Figure 8(f) and (g) two results using HB numbering are given. The plots for BW numbering are similar and therefore not shown here.

5.2. *Test problem 2: stationary flow around BAC 3-11/RES/30/21 airfoil*

This test case is a standard cruise configuration [45] of the Collaborative Research Center SFB 401 [18] with $M_\infty = 0.77$ and $\alpha = 0.00°$, see also [46]. In Figure 1 we give a typical grid that is used in the simulation. We take parameter values $\tau = 1.25$, $\gamma_{\max} = 200$ and $k_r = 10$. For a typical Jacobian $\mathbf{A}$ we show graph $\mathcal{G}(\mathbf{A})$, reduced graph $\hat{\mathcal{G}}(\mathbf{A})$ and the effect of the WRG renumbering in Figure 10.

The behaviour of the preconditioned BiCGSTAB method is illustrated in Figure 11. In this Figure we give the number of iterations that the PBGS-preconditioned BiCGSTAB method needs to satisfy the stopping criterion for the linear solver in every time step. We only give results for the time steps after the last (10th) adaptation.

DOI: 10.1002/fld

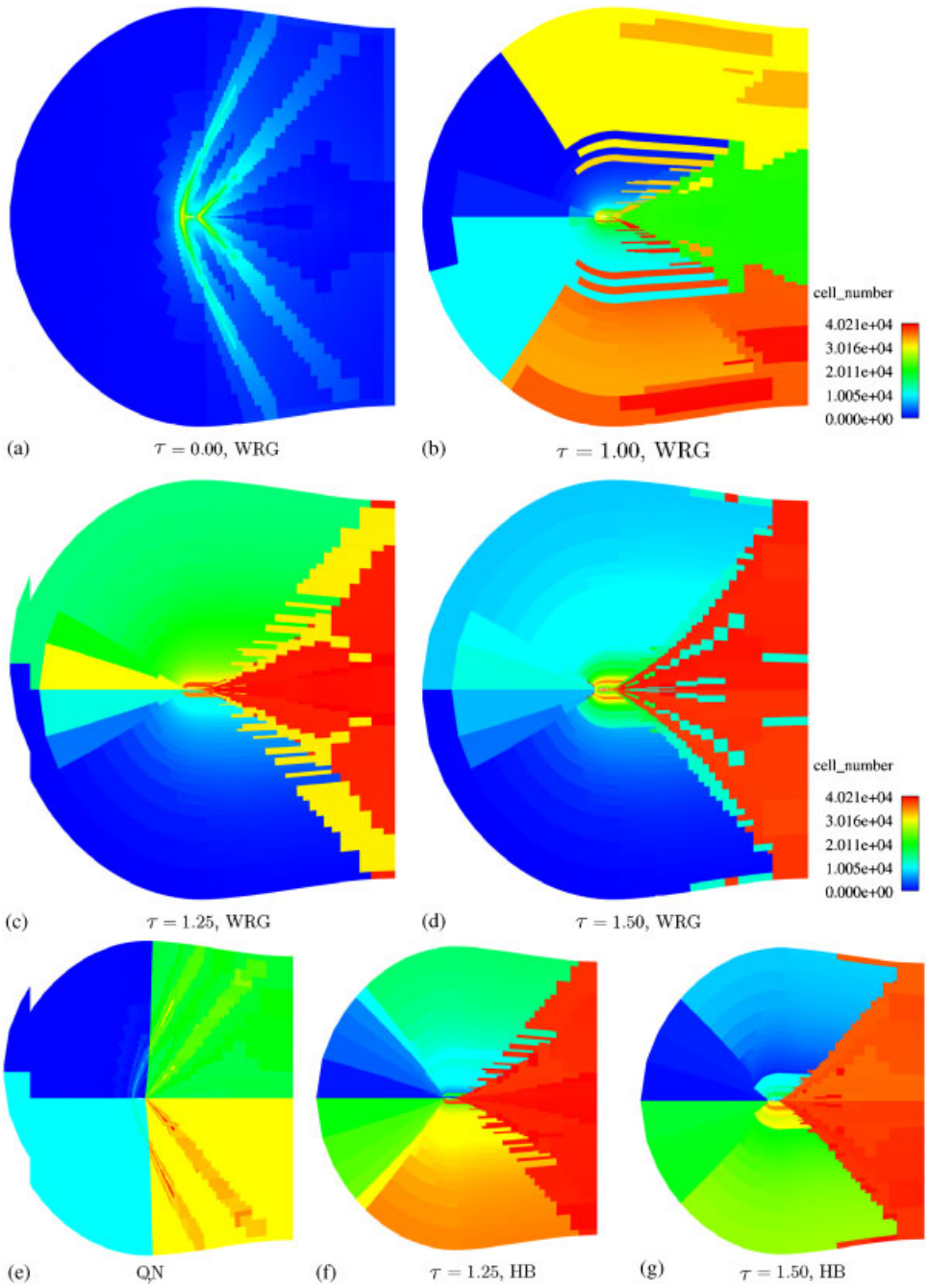

Figure 8. Test problem 1C, finest computational grid. The colouring represents the numbers the cells were given by the algorithms QN, HB and WRG for different values of τ: (a) $\tau = 0.00$, WRG; (b) $\tau = 1.00$, WRG; (c) $\tau = 1.25$, WRG; (d) $\tau = 1.50$, WRG; (e) QN; (f) $\tau = 1.25$, HB; and (g) $\tau = 1.50$, HB.

DOI: 10.1002/fld

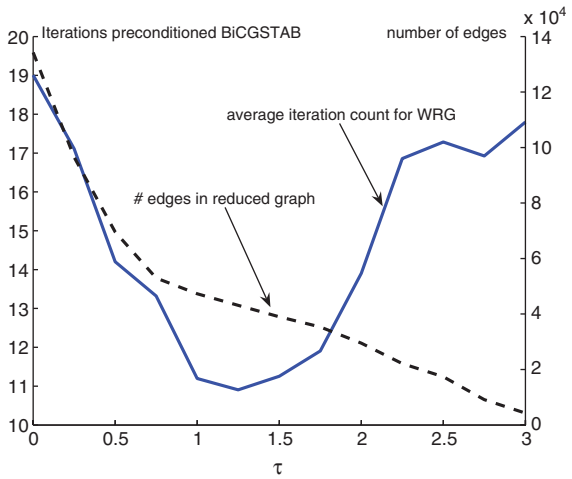

Figure 9. Test case 1C, finest computational grid: experiment with different values for the graph reduction parameter τ. Average iteration count for preconditioned BiCGSTAB using WRG numbering (solid, left axis) and number of edges of the corresponding reduced graph (dashed, right axis).

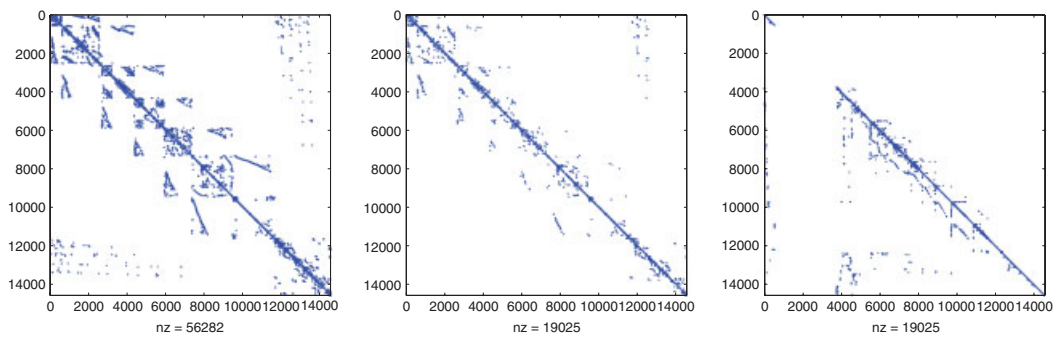

Figure 10. Test problem 2: graph $\mathcal{G}(\mathbf{A})$, reduced graph $\hat{\mathcal{G}}(\mathbf{A})$ and renumbered reduced graph of Jacobian matrix $\mathbf{A}$ from Figure 5.

There is a clear systematic improvement when using the WRG renumbering. The savings are about 38%. A comparison to the BW and HB renumbering methods is shown in Table VI.

5.3. Test problem 3: stationary flow in oblique three-dimensional channel

In this problem we consider a flow through a three-dimensional channel with a bump at the bottom. Cross-sections of this channel with the x–y and x–z-plane are given in Figure 12. The non-rectangular form is used to obtain a truly three-dimensional flow. Inflow and outflow conditions are prescribed at both ends of the channel. At inflow we take $M_\infty = 1.3$ and $\alpha = 0.00°$.

The parameters in this test case are $\gamma_{\max} = 200$, $\tau = 1.25$ and $k_r = \infty$. Some results are presented in Table VII. If instead of $\gamma_{\max} = 200$ we take $\gamma_{\max} = 1000$ then with the orderings resulting from

DOI: 10.1002/fld

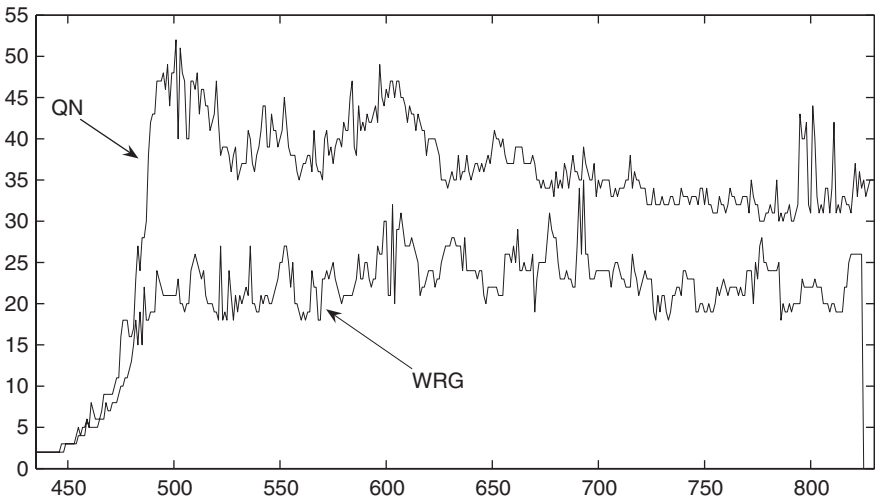

Figure 11. Test problem 2: number of PBGS-preconditioned BiCGSTAB-iterations in every time step, time steps on finest level.

Table VI. Test problem 2: average iteration count on finest level (10th discretization level).

Numbering	QN	BW	HB	WRG
Average iteration count	33.5	22.2	22.2	20.6
Saving (%)	0	33.7	33.6	38.4

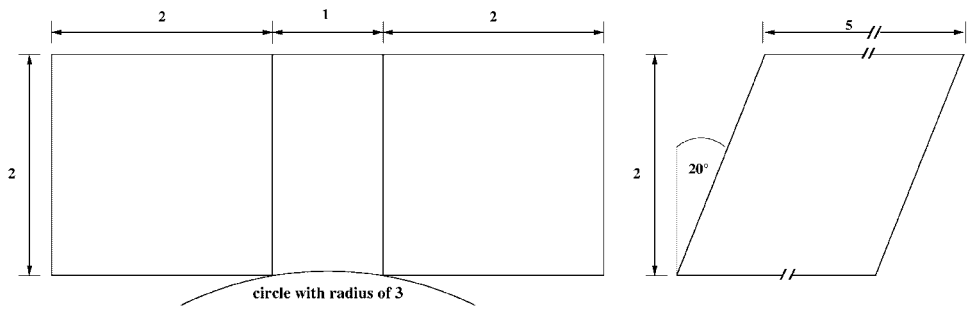

Figure 12. Test problem 3: oblique channel with a bump. Left: x–y plane. Right: x–z plane.

Table VII. Test problem 3, average iteration count on finest level (after 4th adaptation).

Numbering	QN	BW	HB	WRG
Average iteration count	15.0	14.3	14.1	12.9
Saving (%)	0	4.2	6.0	13.9

DOI: 10.1002/fld

QN, BW and HB the PBGS-preconditioned BiCGSTAB solver diverged in at least one time step during the time integration on the finest discretization level. With WRG renumbering, however, this was not the case. Thanks to the higher value $\gamma_{\max} = 1000$ we need about 16.1% of time steps less than with $\gamma_{\max} = 200$. The average iteration count then is 13.9 for WRG. When summing up all Krylov-iterations on the finest level, the total amount of iterations is 22.3% less than with QN numbering and $\gamma_{\max} = 200$.

Hence, this illustrates a further important advantage of the WRG renumbering, namely that it improves the *robustness* of the linear solver.

6. SUMMARY

Both the PBILU and PBGS methods are useful preconditioners in Newton–Krylov methods for compressible flow simulations. The behaviour of these preconditioners depends on the ordering of the block-unknowns (cells). In this paper we present *ordering techniques for the PBGS method* that use ideas from algebraic multigrid methods. First, a reduced weighted directed graph is constructed and then a renumbering of the vertices in this graph is determined. For this renumbering we use methods from the field of multigrid solvers for convection-dominated problems (BW and HB) and a modification of these (WRG). All three methods are implemented in QUADFLOW using the PETSc library. The reordering algorithm is *black-box*, except for the (critical) graph reduction parameter τ in (12). In most test cases a good choice for this grid-reduction parameter turns out to be $\tau = 1.25$. A systematic comparative study shows that for our problem class the WRG reordering yields the best results. Using this reordering we improve the *robustness* of the iterative solver. Even with large CFL numbers (e.g. 200, 1000, 5000) the linear solver always converges if we use PBGS with WRG reordering, whereas with other orderings the solver sometimes diverges. This implies that with WRG reordering it is possible to use larger CFL numbers in order to reduce the total number of time steps. Using the reordering one can improve the *efficiency* of the linear solver significantly. The execution time of the iterative solver part can be reduced by 10% (for complex transonic flows) up to more than 50% (for supersonic flows). For efficiency reasons the reordering is not computed for each new Jacobian but kept fixed in a number of time steps.

The reordering algorithm can also be applied in the setting of (linear or nonlinear) multigrid solvers with block-Gauss–Seidel type smoothers for compressible flow problems.

ACKNOWLEDGEMENTS

The experiments in Section 5 are done using the QUADFLOW solver developed in the Collaborative Research Center SFB 401. The authors acknowledge the fruitful collaboration with several members of the QUADFLOW research group.

We would also like to thank to the referee for his constructive comments.

REFERENCES

1. Brandt A. Multi-level adaptive solutions to boundary value problems. *Mathematics of Computation* 1997; **31**: 333–390.
2. Hackbusch W. *Multi-grid Methods and Applications*. Springer: Berlin, 1985.
3. Jameson A. Solution of the Euler equations for two-dimensional transonic flow by a multigrid method. *Applied Mathematics and Computation* 1983; **13**:327–356.

DOI: 10.1002/fld

4. Jameson A, Caughey DA. How many steps are required to solve the Euler equations of steady, compressible flow: in search of a fast solution algorithm. *AIAA Paper*, vol. 2673, 2001.
5. van Leer B, Darmofal D. Steady Euler solutions in $O(N)$ operations. In *Multigrid Methods*, Dick E, Riemslagh K, Vierendeels J (eds), vol. VI. 1999; 24–33.
6. Lepot I, Geuzaine P, Meers F, Essers J-A, Vaassen J-M. Analysis of several multigrid implicit algorithms for the solution of the Euler equations on unstructured meshes. In *Multigrid Methods*, Dick E, Riemslagh K, Vierendeels J (eds), vol. VI. 1999; 157–163.
7. Mavripilis DJ, Venkatakrishnan V. Implicit method for the computation of unsteady flows on unstructured grids. *Journal of Computational Physics* 1996; **127**:380–397.
8. Luo H, Baum J, Löhner R. A fast, matrix-free implicit method for compressible flows on unstructured grids. *Journal of Computational Physics* 1998; **146**:664–690.
9. McHugh PR, Knoll DA. Comparison of standard and matrix-free implementations of several Newton–Krylov solvers. *AIAA Journal* 1994; **32**:394–400.
10. Meister A. Comparison of different Krylov subspace methods embedded in an implicit finite volume scheme for the computation of viscous and inviscid flow fields on unstructured grids. *Journal of Computational Physics* 1998; **140**:311–345.
11. Meister A, Sonar Th. Finite-volume schemes for compressible fluid flow. *Surveys on Mathematics for Industry* 1998; **8**:1–36.
12. Meister A, Vömel C. Efficient preconditioning of linear systems arising from the discretization of hyperbolic conservation laws. *Advances in Computational Mathematics* 2001; **14**:49–73.
13. Venkatakrishnan V. Implicit schemes and parallel computing in unstructured grid CFD. *ICASE-Report*, vol. 28, 1995.
14. Ballmann J (ed.). Flow modulation and fluid–structure-interaction at airplane wings. *Numerical Notes on Fluid Mechanics*, vol. 84. Springer: Berlin, 2003.
15. Brakhage KH, Müller S. Algebraic-hyperbolic grid generation with precise control of intersection of angles. *International Journal for Numerical Methods in Fluids* 2000; **33**:89–123.
16. Bramkamp F, Gottschlich-Müller B, Hesse M, Lamby Ph, Müller S, Ballmann J, Brakhage K-H, Dahmen W. H-adaptive multiscale schemes for the compressible Navier–Stokes equations: polyhedral discretization. In *Data Compression and Mesh Generation*, Ballmann J (ed.), 2001. Flow modulation and fluid–structure-interaction at airplane wings. *Numerical Notes on Fluid Mechanics*, vol. 84. Springer: Berlin, 2003; 125–204.
17. Bramkamp F, Lamby Ph, Müller S. An adaptive multiscale finite volume solver for unsteady and steady state flow computations. *Journal of Computational Physics* 2004; **197/2**:460–490.
18. SFB 401. Collaborative Research Center, Modulation of flow and fluid–structure interaction at airplane wings, RWTH Aachen University of Technology, http://www.lufmech.rwth-aachen.de/sfb401/kufa-e.html
19. Balay S, Buschelman K, Gropp WD, Kaushik D, Knepley M, McInnes LC, Smith BF, Zhang H. PETSc, http://www-fp.mcs.anl.gov/petsc/, 1992.
20. Balay S, Buschelman K, Eijkhout V, Gropp WD, Kaushik D, Knepley MG, McInnes LC, Smith BF, Zhang H. *PETSc Users Manual*, *ANL-95/11—Revision 2.1.5*, Argonne National Laboratory, 2004.
21. Ajmani K, Ng W-F, Liou M. Preconditioned conjugate gradient methods for the Navier–Stokes equations. *Journal of Computational Physics* 1994; **110**:68–81.
22. D'Azevedo EF, Forsyth PA, Tang W-P. Ordering methods for preconditioned conjugate gradient methods applied to unstructured grid problems. *SIAM Journal on Matrix Analysis and Applications* 1992; **13**(3):944–961.
23. Saad Y. Preconditioned Krylov subspace methods for CFD applications. In *Solution Techniques for Large-Scale CFD-Problems*, Habashi WG (ed.). Wiley: New York, 1995; 139–158.
24. Bey J, Wittum G. Downwind numbering: robust multigrid for convection–diffusion problems. *Applied Numerical Mathematics* 1997; **23**:177–192.
25. Hackbusch W. On the feedback vertex set for a planar graph. *Computing* 1997; **58**:129–155.
26. Hackbusch W, Probst T. Downwind Gauß–Seidel smoothing for convection dominated problems. *Numerical Linear Algebra with Applications* 1997; **4**(2):85–102.
27. Rentz-Reichert H, Wittum G. A comparison of smoothers and numbering strategies for laminar flow around cylinder. In *Flow Simulation with High-Performance Computers II*, Hrischel E (ed.). Notes on Numerical Fluid Mechanics, vol. 52. Vieweg: Bräunschweig, 1996; 134–149.
28. Turek S. On ordering strategies in a multigrid algorithm. *Proceedings 8th GAMM—Seminar*, Notes on Numerical Fluid Mechanics, vol. 41. Vieweg: Bräunschweig, 1992.
29. Cuthill E. Several strategies for reducing the band width of matrices. In *Sparse Matrices and Their Applications*, Rose DJ, Willoughby RA (eds). New York, 1997; 157–166.

DOI: 10.1002/fld

30. Cuthill E, McKee J. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of ACM National Conference*, New York, 1969; 157–172.
31. Stüben K. An introduction in algebraic multigrid, Appendix A. In *Multigrid*, Trottenberg U, Oosterlee CW, Schüller A (eds). Academic Press: GMD Birlinghoven, St. Augustin, 2001.
32. Gutsch S, Probst T. Cyclic and feedback vertex set ordering for the 2D convection–diffusion equation. *Technical Report*, Universit Kiel, vol. 97-22, 1997.
33. Toro EF, Spruce M, Speares W. Restoration of the contact surface in the HLL Riemann solver. *Shock Waves* 1994; **4**:25–34.
34. van Leer B. Flux vector splitting for the Euler equations. In *Proceedings of the 8th International Conference on Numerical Methods in Fluid Dynamics*, Krause E (ed.). Lecture Notes in Physics, vol. 170. Springer: Berlin, 1982; 507–512.
35. Hänel D, Schwane F. An implicit flux-vector splitting scheme for the computation of viscous hypersonic flow. *AIAA Paper*, vol. 0274, 1989.
36. Edwards J, Liou MS. Low-diffusion flux-splitting methods for flows at all speeds. *AIAA Journal* 1993; **36**(9): 457–497.
37. Wada Y, Liou MS. A flux splitting scheme with high-resolution and robustness for discontinuities. *AIAA Paper*, vol. 94-0083, 1994.
38. Venkatakrishnan V. Convergence to steady state solutions of the Euler equations on unstructured grids with limiters. *Journal of Computational Physics* 1995; **118**:120–130.
39. Batten P, Leschziner MA, Goldberg UC. Average-state Jacobians and implicit methods for compressible viscous and turbulent flows. *Journal of Computational Physics* 1997; **137**:38–78.
40. Vanden KJ, Orkwis PD. Comparison of numerical and analytical Jacobians. *AIAA Journal* 1996; **34**(6):1125–1129.
41. Saad Y. *Iterative Methods for Sparse Linear Systems*. PWS Publishing Company: Boston, 1996.
42. Reusken A. On the approximate cyclic reduction preconditioner. *SIAM Journal on Scientific Computing* 2000; **21**:565–590.
43. Kickinger F. Algebraic multigrid for discrete elliptic second-order problems, multigrid methods V. In *Proceedings of the 5th European Multigrid Conference*, Hackbusch W (ed.). Lecture Notes in Computational Science and Engineering, vol. 3. Springer: Berlin, 1998; 157–172.
44. Jones DJ. Reference test cases and contributors, test cases for inviscid flow field methods. *AGARD Advisory Report*, vol. 211(5), 1986.
45. Ballmann J. Flow modulation and fluid–structure-interaction at airplane wings—survey and results of the collaborative Research Center SFB 401. *DGLR 2002-009*, 2002.
46. Moir IRM. Measurements on a two-dimensional aerofoil with high-lift-devices, volume 1 and 2. *AGARD Advisory Report*, A Selection of Experimental Test Cases for the Validation of CFD Codes, vol. 303, 1994.

DOI: 10.1002/fld