Preconditioning GMRES for steady compressible inviscid flows

Philipp Birken*

May 2, 2002

Abstract

Most of the computing time in an implicit finite volume method is spent solving the linear equation systems. The performance of the Newton-Krylov solver depends on the preconditioner. Choosing the appropriate preconditioner is crucial for the overall performance of the flow solver. It is demonstrated through numerical experiments, that an incomplete LU decomposition with higher level of fill gives good efficiency for the considered aerodynamic problems.

1 Introduction

The simulation of flows around airfoils, as well as their aeroelastic interaction, has become an important topic in aircraft design and security control. The aircraft designers are not only required to provide extensive experiments with real aircrafts, but also to present numerical results, before a new plane is accepted by the office of aviation. Unfortunately, it is not possible to calculate realistic threedimensional turbulent flows fast enough. A thorough numerical testing of an aircraft requires several hundred if not thousands of runs of the flow solver. Tracking the turbulent wake of a starting plane for 80 wingspans to determine its impact on following planes is a goal of modern aerodynamics, but its industrial realization requires better algorithms as well as better computers.

The first task in developing a flow solver for threedimensional compressible turbulent flows is a code for twodimensional compressible inviscid flows. Including turbulence is comparatively easy and making a sequential threedimensional code is more hard work than actually introducing new concepts to the 2D-solver. This is why for this report, only two dimensional stationary problems were considered. We are not interested in issues of discretization, but only in the solution of nonlinear equation systems.

The famous CFL stability condition imposes a restriction on the time step of a solver. The speed of the fastest wave determines the time step for explicit methods. Thus it is guaranteed, that a shock does not move more than one cell in one time step. Implicit

*The author was supported by the DFG as a member of the SFB401

methods are not bound to this condition. The Navier Stokes and Euler equations, which model compressible flows, are very stiff and the shock does most often not travel at the speed of the fastest wave, but much slower. The CFL condition is therefore not necessary to resolve the flow adequately in time. This means that implicit methods are the method of choice even for unsteady flows.

In our context, there are essentially two different types of solver: Multigrid and Newton-Krylov. Multigrid may also be used as a preconditioner in a Newton-Krylov algorithm. In the multigrid field, there are two approaches: the FAS approach by Brandt [4], which applies a nonlinear multigrid algorithm to the steady state equations and the multigrid algorithm by Jameson [6], which is also applied to steady state problems, using an explicit Runge Kutta method as smoother. This is done using the so called dual time.

In an implicit method, a nonlinear equation system is obtained in every time step. This can be interpreted as a steady state equation, allowing us to apply the multigrid techniques.

In contrast, the Newton-Kylov approach uses a Newton method to solve the nonlinear equation system. The resulting linear system is treated with a preconditioned Krylov subspace method.

The multigrid approaches seem to be faster than the Newton Krylov solvers [7, 9, 12], promising even to obtain optimal convergence [8]. But Newton-Krylov solvers are much much easier to implement AND more robust, making them competitive [15, 23, 10]. Their success depends on the preconditioner used and we will show that an incomplete LU decomposition with higher level of fill [19] is a very good choice for the class of problems we are interested in.

2 The governing equations

We are considering the two dimensional time dependent Euler equations, which model the motion of a compressible gas. They form a system of conservation laws, namely the conservation of energy, mass and momentum. For a control volume Ω with boundary $\partial\Omega$ and outward normal $\mathbf{n}$, they can be written as:

$$\int_\Omega \frac{\partial \mathbf{u}}{\partial t} dV + \oint_{\partial\Omega} \mathbf{F}^c(\mathbf{u})\mathbf{n} dS = 0. \tag{1}$$

This means that the change in the conserved variables in the control volume is equal to the flux over the boundary. The state vector of conserved flow variables $\mathbf{u}$ and the physically conservative flux F^c are defined by

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

where ρ denotes the density, p the pressure, $\mathbf{v}$ the velocity vector, e_{tot} the total energy and h_{tot} the total enthalpy. The system is closed by the equation of state for a perfect gas $p = \rho(\gamma - 1)(e_{tot} - 1/2|\mathbf{v}|^2)$, where γ is the ratio of specific heats, taken as 1.4.

While we want to solve the time independent Euler equations, the task of solving this nonlinear equation directly is difficult. This is because they are - dependent on the Mach number - of elliptic, hyperbolic or mixed elliptic/hyperbolic type, while the time dependent equations are always hyperbolic. Therefore, we solve the time dependent equations forward in time, until we obtain steady state. For one timestep, the equations read as follows:

$$\int_{t_n}^{tn+1} \int_\Omega \frac{\partial}{\partial t}\mathbf{u}(\mathbf{x},t)dVdt + \int_{t_n}^{t_{n+1}} \oint_{\partial\Omega} \mathbf{F}^c(\mathbf{u}(\mathbf{x},t))\mathbf{n}dSdt = 0.$$

3 The Finite Volume Solver Quadflow

All the computations herein have been produced using the Quadflow solver [3], which is being developed at the IGPM and the Lehrstuhl für Mechanik. The core of the flow solver is a finite volume scheme which is valid for arbitrary meshes. Cells are considered as polygons, allowing for any form of cells and also for hanging nodes. The mesh is based on a multiblock decomposition of the computational domain. On each block, the grid is generated through a B-Spline technique (see [2]) , which is suitable for local adaptation and makes it possible to represent the grid with low memory costs. It should be noted that a proper multiblock decomposition is important for the overall performance of the Flow Solver. Without a good multiblock structure, no satisfactory solutions can be produced and the nonlinear solver may have severe performance degradations.

The adaptation criteria is based on a multiscale analysis [17]. By means of a sequence of nested grids, the cell averages $\mathbf{u}_j$ are decomposed into the averages on the coarsest level and details. Coarsening or refining is based on the size of the details.

In QUADFLOW, a variety of Riemann solvers and limiter functions for the discretization in space, as well as several timestepping schemes are incorporated. For the computations in this paper we used the HLLC scheme proposed by Batten and Leschziner [1] for the discretization of the convective fluxes. To obtain second order in space, a linear reconstruction of the variables is used. This was combined with the Venkatakrishnan-Limiter [22] to avoid oscillations near discontinuities. The time integration was done with the implicit Euler scheme, because for our purpose, no high order method is needed in time, as time efficiency is not important for steady state computations, but stability is the important criterion. This was combined with so called local timesteps for a fast convergence to steady state. The application of this technique can be interpreted as a nonlinear preconditioner for the time dependent PDE.

Local timesteps mean that in every timestep, a CFL-number is prescribed and the timesteps Δt are then computed locally in every cell via

$$\Delta t_i = \lambda_{i,max} \cdot CFL \cdot \Delta x_i,$$

where $\lambda_{i,max}$ is the largest Euler eigenvalue in the cell i. The CFL-number is computed by a simple heuristic: initially and after every adaptation step, it is set to a user defined

low (between 0.8 and 2) value. In every timestep, it is then multiplied by a factor of 1.2 until a given maximal CFL number is reached.

4 The Newton iteration

To solve the nonlinear equation arising in the implicit time integration, we use Newton's method. For our solution method of the stationary Euler equations, one Newton step per time step is fully sufficient to obtain convergence to steady state. The algorithm thus reads:

$$G'(U^n)\Delta U = -G(U^n),$$
$$U^{n+1} = U^n + \Delta U,$$

where in cell i, the function G is defined by:

$$G(u_i) = \frac{\mathbf{V}_i}{\Delta t_i}(u_i - u_i^n) + \sum_{edges\, e_i} |e_i| F^{HLLC}(U)\mathbf{n}_{e_i}$$

The linear system is solved using a preconditioned GMRES agorithm and the Jacobian matrix is obtained from the definition of G:

$$G'(U^n) = \mathbf{V}(\Delta t)^{-1} + \sum_{edges\, e} |e| \frac{\partial F^{HLLC}(U^n)}{\partial U}\mathbf{n}_e, \tag{3}$$

where here $\mathbf{V}$ and Δt are diagonal matrices, originating from the computation of the local timesteps.

The derivatives of the numerical Flux can be computed numerically via finite differences. Some discretizations schemes (e.g. HLLC or van Leer Flux Vector Splitting), can be differentiated analytically and the formulas are known. For this report, finite differences were used. It should be noted that the Jacobian corresponds to a first order discretization in space. The reconstructed values are used for the actual computation of the entries, but for simplicity and to save storage, in formula (3) the reconstruction procedure is not incorporated. Using a first order discretization for the Jacobian gives additional robustness. It can be observed that the corresponding domain of convergence of the Newton iteration is larger for a first order than for a second order discretization.

Thus we obtain a sparse unsymmetric linear system, which consists of a weighted identity matrix plus a second term whith derivatives of the flux function. For small time steps, the identity part is dominant, whereas for large timesteps, the other part dominates. This means that the matrix is the more ill conditioned the bigger the time step is and very well conditioned for small timesteps. Therefore we expect to need more computational effort to match a given tolerance in the linear equation system for higher CFL numbers. On the other hand, a high CFL number means less overall steps to reach steady state.

Over the whole integration procedure, we encounter a whole range of matrices, varying in condition number, structure, entries and eigenvalues. Therefore, a robust procedure

to solve the upcoming linear equation systems is needed. In the following chapter, the GMRES algorithm using ILU(p) preconditioning will be explained. Numerical results will show that this a very good and robust solver.

5 The linear solver

We cannot say much about the structure of the linear systems obtained during the solution procedure. They are indefinit, unsymmetric (although most often block symmetric) and rather ill conditioned. The eigenvalues of the matrices are not known. They are not M-matrices and already for low CFL numbers, they lose diagonal dominance. Currently, the most efficient and robust methods to solve sparse large linear systems are preconditioned Krylov subspace methods. For unsymmetric matrices, a variety of such methods exists, the most prominent of these being GMRES by Saad and Schultz [20] and Bi-CGSTAB by van der Vorst [21]. The speed of convergence depends on the condition number of the matrix. For ill conditioned linear systems, as those arising from the discretization of partial differential equations, a preconditioner is needed. In [14], different Krylov subspace methods for 2D compressible Euler and Navier-Stokes problems are compared. Bi-CGSTAB combined with ILU preconditioning is found to be the most robust and efficient method. However, this was only for an implementation where the matrix is stored explicitely. In view of future 3D-applications, we have chosen GMRES, as this allows for a matrix free implementation without loss of performance [5, 22, 13].

However, the choice of a specific Krylov subspace method is not as important as the choice of the preconditioner [16]. The preconditioner has to satisfy a compromise between savings in the number of iterations and the additional cost of one iteration through the preconditioning procedure. These are namely the time to setup or construct the preconditioner and the additional CPU time needed for every iteration. Furthermore, the computed iterates do depend on the preconditioner and thus the preconditioner influences the whole solution procedure. This means that for an evaluation, if a specific preconditioner is good in the context of a flow solver, it is not sufficient to count the savings in iterations, but the overall consumed CPU time for the whole flow solver from the initialization phase to convergence to steady state. For QUADFLOW, we have found that ILU(2) [19] preconditioning gives the best results in overall CPU time for a huge range of aerodynamic testcases. Furthermore, the number of GMRES iterations to satisfy the given tolerance is between 20 and 30. The linear systems are solved up to a user defined tolerance. If the 2-Norm of the linear residuum drops below this value, GMRES is terminated. As will be seen later, solving the linear systems very good is not beneficial for the performance of the flow solver.

We will now shortly explain the preconditioner. The incomplete LU Decomposition [19] is based on the Gaussian elimination procedure. The latter one is well known to produce a lot of fill in in the L and U matrices, resulting in a lot of storage needed to store the decomposition as well as arithmetic costs. Instead of storing and computing the exact decomposition, the ILU algorithm computes only the elements for a prescribed

sparsity pattern and sets all elements not contained in the pattern to zero.

Intuitively, the bigger the allowed sparsity pattern for the ILU decomposition, the smaller the approximation error. Although there do exist counterexamples, the introduction of "levels of fill" has proved to be a reasonable heuristic to control this error in some way, especially for matrices arising from the discretization of PDEs.

The levels of fill are defined recursively: The ILU(0) incomplete factorization is defined through the sparsity pattern of A, thus allowing no additional fill-in in L and U. Let the ILU(0) decomposition consist of the matrices L_0 and U_0. The matrix L_0U_0 has in general a bigger sparsity pattern than A. This is called first level fill and the incomplete decomposition corresponding to the augmented sparsity pattern is called $ILU(1)$. Applying this recursively, we obtain $ILU(p)$. Note that it is not necessary to compute lower level decompositions first. Instead, it is sufficient to first run a cheap algorithm to determine the sparsity pattern for the $ILU(p)$ decomposition and then compute the decomposition corresponding to the pattern.

5.1 The PETSc Interface

The implementation of the linear algebra routines in QUADFLOW uses the PETSc library of the Argonne National Laboratory [18]. The PETSc sparse matrix format is used and the GMRES and ILU routines of the library. PETSc vectors are needed in the subroutines, but the QUADFLOW vectors can be easily transferred to PETSc vectors using pointers.

6 Numerical Results

Numerical experiments were carried out on 2 different airfoils. First, on the well known NACA0012-profile and then on the so called SFB-401 cruise configuration. This is designed to resemble an AIRBUS A380 airfoil. During every run of Quadflow, a total number of six adaptations was carried out. An adaption step occurs after the weighted density residual measured in the 2-Norm has dropped four orders of magnitude. After the last adaptation the solver continues, until the residuum drops another seven orders of magnitude.

It should be noted, that an adaption step changes the discrete solution space and therefore the norm. The old solution is interpolated to the new space and the norm of the residual increases.

The computations were carried on on a Linux Intel PIII-600 MHz. All CPU times represent only the time spent by the flow solver and do not include the time needed for the adaptation or the grid generation.

A variety of test cases was computed over a broad range of Mach numbers (going from Mach 0.16 up to Mach 0.85), maximal CFL numbers and tolerances for the linear residuum as a stopping criterium for GMRES. We were interested in the performance of the preconditioner. There are a lot of algorithmic parameters to choose and a set of parameters is considered best, if it gives a good solution to the given problem (in the

engeneering sense) for the lowest cost in CPU time.

Besides the CPU time and the solution, the number of GMRES iterations needed to match the given tolerance was recorded in every timestep. While the overall CPU time is the prime measure for the performance of a preconditioner, the numbers of GMRES iterations give insight on whether a preconditioner can be significantly improved.

6.1 The transonic regime

This is the numerically most challenging of the considered problems. One of the Euler eigenvalues approaches zero, as the Mach number nears one and the problem becomes stiff. On top of the airfoil, a strong shock appears.

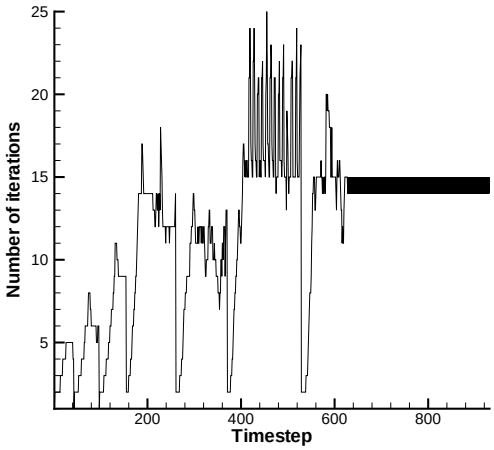

Figure 1: GMRES-iterations for a transonic NACA0012 flow ($M_\infty = 0.8$)

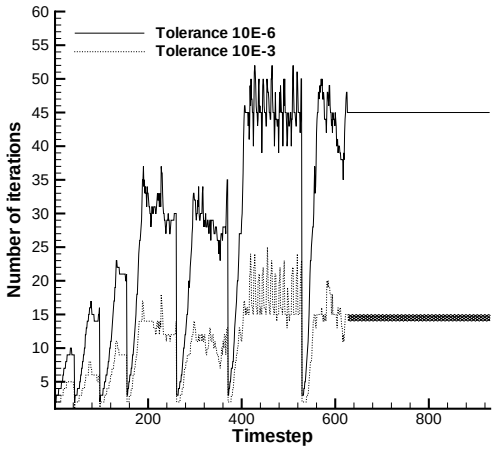

Figure 2: Effect of linear tolerance on the number of GMRES iterations

The first example is a flow over the NACA0012 airfoil at a freestream Mach number of 0.8 with an angle of attack of 0.1 degrees. In Figure 1, the number of iterations needed in GMRES to match the tolerance of the residuum in the linear system is plotted over the time steps. For this example, the tolerance was set to 10E-3. The minima correspond to an adaption step, as after every multiscale analysis, the CFL number ist reset to the low starting value (e.g. 0.8). This corresponds to a very good conditioned linear system, therefore, GMRES has no problems to solve these very quickly. After this, the CFL number is increased (and thus the condition worsens) and correspondingly, GMRES takes longer to fulfil the tolerance. After a while, the CFL number reaches a user defined maximum (here: $CFL = 500$). From there on, the iteration numbers go down, as the flow field becomes more stationary. On average, GMRES takes about 15 iterations and never more than 25 to terminate.

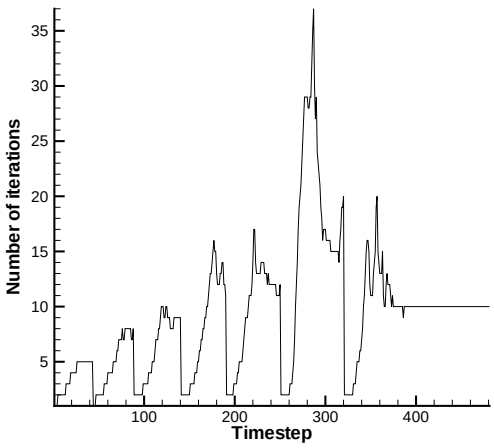

Figure 3: GMRES iterations for transonic flow ($M_\infty = 0.8$) over the SFB401 cruise configuration.

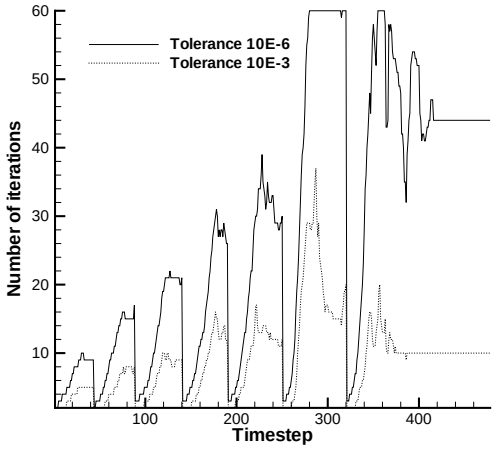

Figure 4: Comparison of different tolerances in GMRES for the SFB401 cruise configuration.

These results were compared with the same run of quadflow, only changing the tolerance in GMRES to $10E-6$. Not surprisingly, the iteration numbers increase (see Figure 2). But the adaptations, which are steered through the nonlinear residual, happen at exactly the same time. Furthermore, the total number of timesteps is the same, as is the convergence history. Thus, solving the linear systems more efficiently, brings no gain for the nonlinear solution algorithm. This is quantified in the following tabular:

GMRES-Tolerance	Timesteps	Max. Cells	CPU-Time
10E-3	931	13138	4700 s
10E-6	930	13138	7000 s

The same comparison was done, only using the SFB-cruise configuration instead of the NACA0012 airfoil. The result is qualitatively the same, as can be seen in figure 3 and 4 as well as in the next table.

GMRES-Tolerance	Timesteps	Max. Cells	CPU-Time
10E-3	478	13930	2900 s
10E-6	481	13930	2000 s

In figures 5 and 6, the norm of the volume weighted relative L_2-residual of the Density is plotted over the timestep for both cases (high and low tolerance in the linear solver). The curves match nearly exactly, a difference appears first for very low residuals. This is probably due to rounding errors, which become more significant if the residual becomes very small.

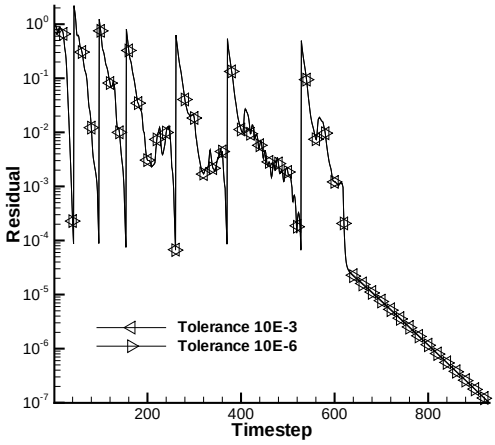

Figure 5: Convergence histories for transonic flow ($M_\infty = 0.8$) over the NACA0012 airfoil.

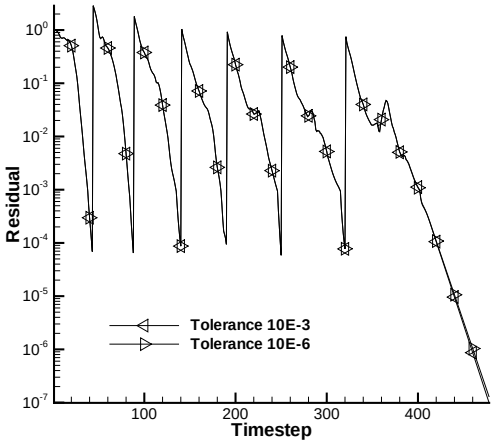

Figure 6: Convergence histories for transonic flow over the SFB401 cruise configuration.

6.2 The subsonic case

The subsonic regime is much easier to handle numerically. There are no strong shocks and the Euler eigenvalues stay away from zero. As an example, figures 7 shows the number of GMRES iterations per time step for a calculation with Mach number 0.5 and angle of attack 1, whereas figure 8 corresponds to Mach 0.16 with an angle of attack of 3 degrees.

GMRES-Tolerance	Timesteps	Max. Cells	CPU-Time
10E-2	330	6478	600 s
10E-2	404	2488	350 s

A tolerance of $10E-2$ ist sufficient for these much simpler cases to obtain convergence. The small CPU times show clearly, how much the faster convergence of GMRES due to the better condition number of the linear systems in the subsonic case, impacts the overall time to reach steady state. The number of time steps is roughly halfed, while the computing time is decreased by a factor of ten.

6.3 Different levels of fill

The next example is again a Naca0012 computation at Mach 0.8 with an angle of attack of 0.1. Exactly the same algorithmic parameters as in example one are used, except that the level of fill in the ILU decomposition is varied. Figure 9 shows the different plots of the number of iterations needed for GMRES to satisfy the tolerance. It can be clearly seen, that a higher level of fill results in a higher convergence speed of GMRES. Especially the

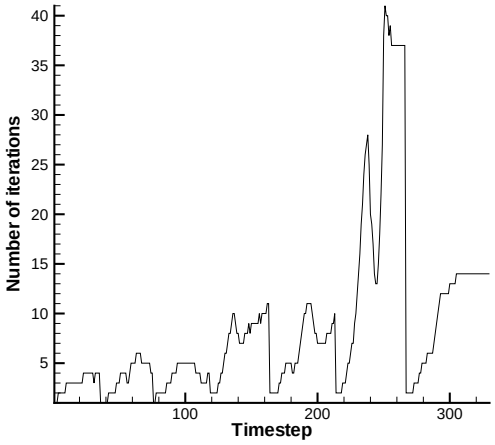

Figure 7: GMRES iterations per timestep for a subsonic flow ($M_\infty = 0.5$) over the NACA0012 airfoil.

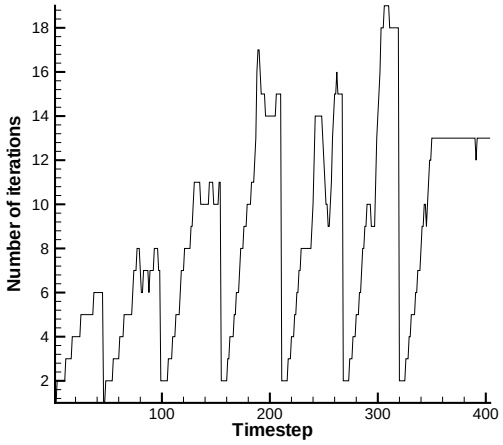

Figure 8: GMRES iterations per timestep for a subsonic flow ($M_\infty = 0.16$) over the NACA0012 airfoil.

first two levels of fill bring a substantial gain. It should be kept in mind, although, that a decomposition corresponding to a higher level of fill takes more time to compute than a low level fill decomposition. This is seen in the tabular. Althoug GMRES converges much faster with ILU(2) preconditioning than with ILU(1) preconditioning, the increase in overall speed is less substantial.

ILU-Level	Timesteps	Max. Cells	CPU-Time
0	922	13126	5900 s
1	925	13138	5000 s
2	931	13138	4800 s
3	929	13138	4700 s

Otherwise, the algorithmic behaviour is nearly identical. This was to be expected, as the tolerance in GMRES is the same for all preconditioners. It should be noted that, although ILU(3) performs best in this example, this was normally not the case in the numerical experiments, but ILU(2) was the better preconditioner.

6.3.1 Other Preconditioners

The only other notable preconditioner is the symmetric Gauss Seidel method using an appropriate renumbering of variables. Computations were made using Jacobi preconditioning, but convergence is at least 20 times slower than with ILU(2). SGS is also much slower than ILU, but it can be used to obtain a wholly matrix free method, were the matrix is not even stored for preconditioning purposes [10, 11].

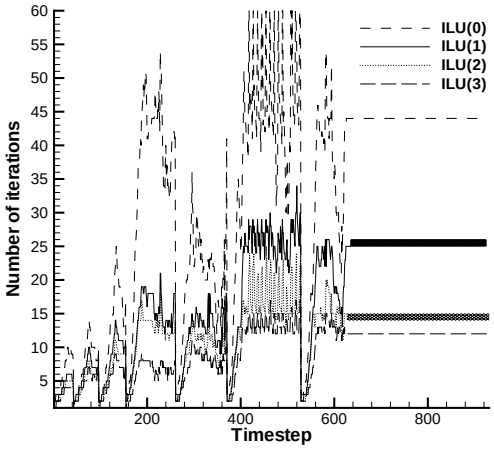

Figure 9: Comparison of GMRES performance for different ILU preconditioners (transonic NACA0012 flow).

6.4 The PETSc Interface

The PETSc routines are very efficient and self developed code seems to be less fast. Unfortunately, using PETSc is not easy and it takes some time to get used to the structure of the library. On the other hand, support is very good. The development team has several people working only on support and they answer every question you might have in one day.

7 Summary

An implicit finite volume solver using a Newton-Krylov method was outlined. The impact of preconditioning and the tolerance in the Krylov subspace method GMRES was examined with numerical experiments. It can be seen, that a high tolerance in GMRES is not beneficial for the implicit Euler method, but consumes just more computer power. This is probably due to the fact, that the Jacobian matrix is based on a first order discretization and therefore, no true Newton method is employed.

Furthermore, it was demonstrated, that ILU(2) is a very good preconditioner for the considered problem. Even for the numerically difficult transonic case, the tolerance is satisfied with about 20-30 GMRES iterations. Other authors obtain similar conclusions using ILU(0) [16, 23], but without considering ILU(2).

A higher level of fill in the incomplete decomposition results in faster convergence of GMRES. This is somewhat negated by the higher amount of time needed to construct the preconditioner. At the moment, the Jacobian and the ILU-decomposition is recomputed in every timestep. Freezing the Matrix or the preconditioner will probably save a lot of time, as was demonstrated by Meister [16], als well as by Venkatakrishnan [23]. It might

be that ILU(3) will then perform better than ILU(2).

On the other hand, 3D computations might make it necessary to use ILU(0) due to excessive storage requirements of higher level of fill decompositions. For parallel computations, ILU is not a good method. It has an inherent sequential natuer and does not scale well. Regarding the experiences with ILU, standard parallel preconditioners like SPAI will probably perform well.

Fortunately, this part of the parallelization process will be quite easy. PETSc is originally designed for parallel codes and for use with MPI. There do exist routines for parallel preconditioners like SPAI or additive or multiplicative Schwarz.

As a last topic, we want to consider the Navier Stokes equations. There, the large number of cells in the boundary layer increases storage problems. Otherwise, ILU is still a very good preconditioner [16].

References

[1] P. Batten, M. A. Leschziner and U.C. Goldberg: Average-State Jacobians and Implicit Methods for Compressible Viscours and Turbulent Flows, J. Comp. Phys. 137, 1997, pp 38-78

[2] K.H. Brakhage and S. Mueller: Algebraic-hyperbolic grid generation with precise control of intersection of angles, Int. J. Numer. Meth. Fluids. 33, 2000, pp 89-123

[3] F. Bramkamp, B. Gottschlich-Müller, M. Hesse, Ph. Lamby, S. Müller, J. Ballmann, K.-H. Brakhage, W. Dahmen: H-adaptive multiscale schemes for the compressible Navier-Stokes equations: polyhedral discretization, data compression and mesh generation, IGPM report 207, 2001

[4] A. Brandt: Multigrid techniques: 1984 Guide with Applications to Fluid Dynamics, Proceedings of the VKI Lectures Series 1984-04, 1984

[5] P.N. Brown, Y. Saad: Hybrid Krylov methods for nonlinear systems of equations, SIAM J. Sci. Stat. Comput., Vol. 11, 1990, pp. 450–481

[6] Jameson, A.: Solution of the Euler equations for two-dimensional transonic flow by a multigrid method, Appl. Math. Comp., Vol. 13, 1983, pp. 327–356

[7] A. Jameson, D.A. Caughey: How many steps are required to solve the Euler equations of steady, compressible flow: in search of a fast solution algorithm, AIAA paper 2001–2673

[8] B. van Leer and D. Darmofal: Steady Euler Solutions in O(N) Operations, Multigrid Methods VI, E. Dick, K. Riemslagh and J. Vierendeels, editors, 1999, pp 24-33

[9] I. Lepot, P. Geuzaine, F. Meers, J.-A. Essers and J.-M. Vaassen: Analysis of Several Multigrid Implicit Algorithms for the Solution of the Euler Equations on Unstructured Meshes, Multigrid Methods VI, E. Dick, K. Riemslagh and J. Vierendeels, editors, 1999, pp 157-163

[10] H. Luo, J. Baum and R. Loehner: A Fast, Matrix-free Implicit Method for Compressible Flows on Unstructured Grids, J. of Comp. Physics 146, 1998, pp 664-690

[11] R. Massjung: Numerical Schemes and well posedness in numerical aeroaelasticity, Dissertation, to appear

[12] D.J. Mavripilis and V. Venkatakrishnan: Implicit method for the computation of unsteady flows on unstructured grids, J. Comp. Phys. 127, 1996, pp 380-397

[13] P.R. McHugh and D.A. Knoll: Comparison of standard and matrix-free implementations of several Newton-Krylov solvers, AIAA J. 32, 1994, pp 394-2400

[14] 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, J. of Comp. Phys., Vol. 140, 1998, pp. 311–345

[15] Meister, A., Sonar, Th.: Finite-volume schemes for compressible fluid flow. Surv. Math. Ind., Vol. 8., 1998, pp. 1–36

[16] Meister, A., C. Vömel: Efficient preconditioning of linear systems arising from the discretization of hyperbolic conservation laws, Advances in Computational Mathematics 14, 2001, pp. 49-73

[17] S. Müller: Adaptive Multiscale Schemes for Conversation Laws, accepted for publication in Lecture Notes on computational Science and Engineering, Springer Verlag

[18] http://www-fp.mcs.anl.gov/petsc/

[19] Saad, Y.: Iterative methods for sparse linear systems, PWS Publishing Company, Boston (1996)

[20] Y. Saad and M.H. Schultz: GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Statist. Comput., Vol. 7, 1986, pp. 856–869

[21] H. van der Vorst: Bi-cgstab: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems, SIAM J. Sci. Stat. Comp. 13, 1992, pp 631

[22] Venkatakrishnan, V.: Convergence to Steady State Solutions of the Euler Equations on unstructured Grids with Limiters, J. Comp. Phys. 118, 1995

[23] Venkatakrishnan, V.: Implicit schemes and parallel computing in unstructured grid CFD, ICASE-report 95-28, 1995