Journal of Engineering Mathematics **35:** 437–453, 1999.
© 1999 *Kluwer Academic Publishers. Printed in the Netherlands.*

Fast computation of stationary inviscid flow around an airfoil

PATRICK STRATING and RENÉ VAN BUUREN
Faculty of Applied Mathematics, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands
e-mail: p.strating@math.utwente.nl e-mail: r.vanbuuren@math.utwente.nl

Received 13 May 1998; accepted in revised form 25 November 1998

Abstract. The parallel performance of an implicit solver for the Euler equations on a structured grid is discussed. The flow studied is a two-dimensional transonic flow around an airfoil. The spatial discretization involves the MUSCL scheme, a higher-order Total Variation Diminishing scheme. The solver described in this paper is an implicit solver that is based on quasi Newton iteration and approximate factorization to solve the linear system of equations resulting from the Euler Backward scheme. It is shown that the implicit time-stepping method can be used as a smoother to obtain an efficient and stable multigrid process. Also, the solver has good properties for parallelization comparable with explicit time-stepping schemes. To preserve data locality domain decomposition is applied to obtain a parallelizable code. Although the domain decomposition slightly affects the efficiency of the approximate factorization method with respect to the number of time steps required to attain the stationary solution, the results show that this hardly affects the performance for practical purposes. The accuracy with which the linear system of equations is solved is found to be an important parameter. Because the method is equally applicable for the Navier-Stokes equations and in three-dimensions, the presented combination of efficient parallel execution and implicit time-integration provides an interesting perspective for time-dependent problems in computational fluid dynamics.

Key words: inviscid compressible flow, parallel implicit time-integration, TVD schemes, multigrid.

1. Introduction

Typical computational fluid dynamics problems require large numbers of floating-point operations, and the wall-clock simulation time of a flow problem is an important aspect. Different computer architectures, for instance based on vector or scalar processors, favor different algorithms, and the choice of algorithm depends on the available hardware.

In many flow solvers time-stepping methods are used in which the solution progresses with time steps towards the steady state. For these solvers, an important choice is whether to use implicit or explicit time-integration. Explicit time-integration methods feature the ease of implementation, and the good performance on state-of-the-art vector and parallel computers, but the time steps are bounded by conditions of numerical stability. These restrictions on the time steps for explicit methods may necessitate a time step much smaller than the time step required for a (time-)accurate solution. In these cases, implicit methods with a larger or even infinite stability region may be more efficient.

In a study of the inviscid two-dimensional Euler equations Van Buuren *et al.* [1] showed that for the third order accurate MUSCL scheme [2] for the spatial discretization and with an Euler Backward implicit time-integration scheme, the residuals could be reduced towards the final residual level corresponding to the steady state within fewer time steps as compared to an explicit four-stage Runge-Kutta method (Figure 1), because time-stepping could proceed with larger steps. In the implicit method at each time step a quasi-Newton iteration produces a linear system of equations that requires the inversion of a large banded matrix. This matrix is further simplified by approximate factorization.

Surprisingly in this study of Van Buuren, the time required for an implicit (local) time step is comparable to the time required for an explicit four-stage Runge-Kutta (local) time step. This is due to the cost of the fourfold calculation of the expensive MUSCL flux for the Runge-Kutta scheme, which has to be calculated only once for the implicit scheme, thus compensating for the additional costs of the calculation and (approximate) inversion of the Jacobian matrix. The convergence rates of the two methods are compared in Figure 1, which clearly demonstrates the advantage of the implicit over the explicit time-stepping method with respect to computation time.

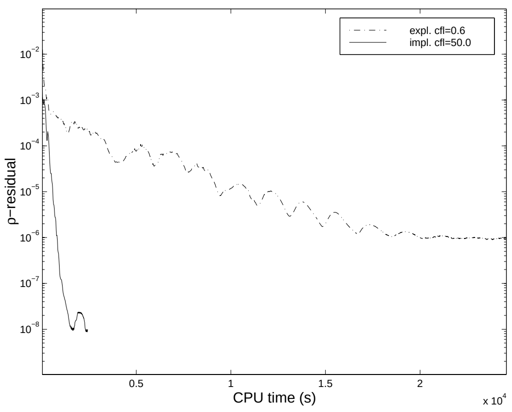


Figure 1. Comparison of explicit four-stage, second-order Runge-Kutta method (dash-dotted line) and implicit Euler backward method. The implicit method (left, solid line) requires much less computation time.

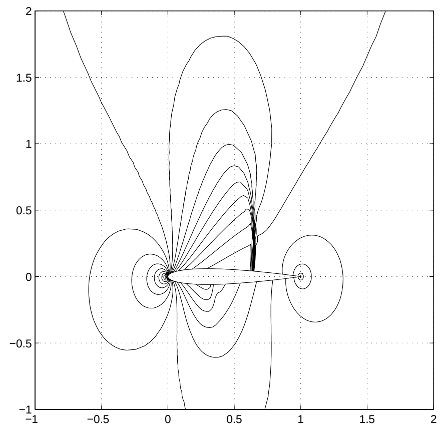


Figure 2. Pressure contours for the two-dimensional, inviscid flow around the NACA0012 airfoil at transonic conditions. In the far field the Mach number of the flow is 0·8, the angle of the airfoil with the incoming flow is 1·25.

Another important point in [1] is the stagnation in convergence visible in Figure 1. This stall in convergence has been understood now as follows. In [1] it was corroborated that the solution of the steady-state Euler equations ($\mathbf{F}(\mathbf{q}) = 0$) at the given free-stream conditions is not a stable solution. This was found after grid refinement, and it was concluded that the instability is physical. If accurate time-stepping is used as a mechanism for the evolution of the solution, the solution will evolve towards a nonstationary (time-dependent) state with a final residual level around 10^{-6}. If the time steps are chosen small enough (CFL$<$ 1·0), this holds for both the implicit and explicit time-marching schemes. This does not mean that a machine-accurate solution of the discretized equations does not exist. With Euler Backward time-stepping a machine-accurate solution is found if the CFL number is chosen between approximately 1·0 and 7·0. However, it is very difficult to understand or even address the interplay of this instability and the solution process, especially if multigrid is involved, because the effect of the instability on the convergence behavior may be very sensitively depending on the grid level.

In this paper the properties of the implicit approximate factorization method are investigated further. The paper is organized as follows. After explaining the basic numerical method in Section 2, we briefly describe the results of multigrid acceleration in Section 3.

In Section 4 we discuss the parallelizability of the implicit inviscid flow solver. Because the prospects of a solver for the Navier-Stokes or Euler equations are determined by its actual performance on new or future computers, the competition between implicit and explicit methods is being influenced by the increased availability of parallel computers. So far, explicit methods have benefited most from the developments in computer hardware, because the repetition of many, simple instructions is well suited for vector processing, and the sole dependence on data from the previous time step is easily handled on parallel computers by supplying copies where needed. The parallelization experiments described in Section 4 indicate that a good parallel performance can be expected for the implicit solver described in this paper, because the solver is not too sensitive to approximations made to make parallel execution possible. In this research we will focus on spatial domain-decomposition techniques. The reason for this is that implementation on both shared memory systems and distributed memory systems is fairly easy, and it should be possible to achieve comparable parallel performance on both types of parallel architectures. Because the domain-decomposition approach described here affects the convergence rate towards the steady state, in contrast to most explicit multiblock algorithms, (theoretically) perfect scaling with the number of processors is not achieved, but the parallel efficiencies seem sufficient for practical purposes.

The simulations described here are for the well-known test case of inviscid flow around a two-dimensional NACA0012 airfoil at a free-stream Mach number of $M_\infty = 0{\cdot}8$ and an angle of attack of the incoming flow of $\alpha = 1{\cdot}25°$ [3]. This problem is also well-documented in [4]. At the given conditions the flow is transonic; the solution has a strong shock on the upper part of the airfoil, a weak shock on the lower part of the airfoil, and a contact discontinuity in the wake. Contour lines of the pressure field of the steady state are plotted in Figure 2. The grid is a C-grid with 289×65 points. Several test computations for different free-stream conditions were performed that behaved analogously to the results for the test case selected in this paper. The case $M = 0{\cdot}85$, $\alpha = 1{\cdot}0$ with stronger shocks, the subsonic case $M = 0{\cdot}63$, $\alpha = 2{\cdot}0$ without shocks and the supersonic case $M = 1{\cdot}2$, $\alpha = 7{\cdot}0$ all produced convergence histories with the same characteristics as described for the case $M = 0{\cdot}80$, $\alpha = 1{\cdot}25$. Therefore, in the main part of the paper we will restrict the results to this case.

In all simulations described in this paper, the coefficients of drag c_d and lift c_l are found to be equal up to at least six digits, so we will only consider the residuals and the convergence histories in this investigation. We will occasionally refer to the time-stepping towards the steady state as ‘convergence of the outer iteration’, which should be distinguished from any (‘inner’) iterative method used to solve the linear system of equations at each time step. In general, the convergence towards the steady state can be rated according to the *number* of time steps required, or according to the overall *computing time*.

2. Governing equations and numerical method

2.1. GOVERNING EQUATIONS

The equations governing inviscid compressible flow are the Euler equations. In Cartesian coordinates in two-dimensions they read

$$\frac{\partial \mathbf{q}}{\partial t} + \frac{\partial \mathbf{f}}{\partial x} + \frac{\partial \mathbf{g}}{\partial y} = 0 \tag{1}$$

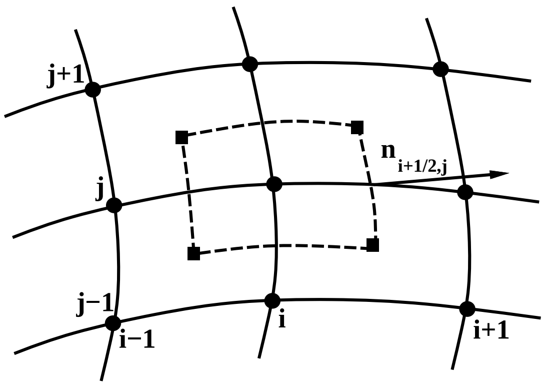


Figure 3. Spatial discretization.

with

$$\mathbf{q} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ E \end{bmatrix}, \qquad \mathbf{f} = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho u v \\ u(E+p) \end{bmatrix} \quad \text{and} \quad \mathbf{g} = \begin{bmatrix} \rho v \\ \rho u v \\ \rho v^2 + p \\ v(E+p) \end{bmatrix}, \tag{2}$$

where ρ is the density, E is the total energy density and u and v are the velocity components in x and y directions, respectively. The constitutive equation for the pressure, p, is given by

$$p = (\gamma - 1)(E - \tfrac{1}{2}\rho(u^2 + v^2)), \tag{3}$$

where γ is the adiabatic gas constant, which we take as $\gamma = 1{\cdot}4$.

2.2. Spatial discretization

The Euler equations in (1) are in conservation form where the state vector **q** contains the densities of the conserved quantities and **f** and **g** are the corresponding flux vectors. Integration of (1) over an arbitrary volume in space shows that the components of **q** change only due to a flux through the boundaries of this volume. To solve (1) we use a finite-volume method on a structured grid that computes the flux over the control-volume edges in Figure 3. The equation reads

$$\Omega_{ij}\frac{\mathrm{d}\mathbf{q}_{ij}}{\mathrm{d}t} + \mathbf{h}_{i+(1/2),j} - \mathbf{h}_{i-(1/2),j} + \mathbf{h}_{i,j+(1/2)} - \mathbf{h}_{i,j-(1/2)} = 0, \tag{4}$$

where Ω_{ij} is the volume of the control volume and **h** is the numerical flux vector on the four boundaries segments $(i + \frac{1}{2}, j)$, etc.

In transonic-flow applications shocks may occur. These shocks demand certain features of the numerical scheme. Firstly, the shock must be properly located. Secondly, the scheme has to suppress numerical oscillations near the shock. By construction, the scheme of (4) is automatically in conservation form. This is a necessary condition for a proper capturing of the shock [5]. To suppress numerical oscillations near the shock, artificial dissipation can be added explicitly or implicitly by the use of upwind schemes. First-order upwind schemes suppress these oscillations, but smear the shock. This can be avoided if a higher-order TVD (Total Variation Diminishing) scheme, *e.g.* as developed by Van Leer [2], is used. This type of higher-order scheme uses a nonlinear *limiter* function $\psi(r)$ which limits gradient differences of the solution between adjacent cells. The limiters can roughly be divided into two groups [5]: limiters which are differentiable for $r > 0$, *e.g.*, Van Leer's limiter, or limiters which are nondifferentiable for $r > 0$, like the minmod-type limiter which we use in this paper.

The numerical flux on the control-volume edges is approximated by the flux-difference splitting method of Roe [6], *e.g.*

$$\mathbf{h}_{i+(1/2),j} = \tfrac{1}{2} l_{i+(1/2),j} \begin{bmatrix} \mathbf{f}(\mathbf{q}_l) + \mathbf{f}(\mathbf{q}_r) \\ \mathbf{g}(\mathbf{q}_l) + \mathbf{g}(\mathbf{q}_r) \end{bmatrix} \cdot \mathbf{n}_{i+(1/2),j} - \tfrac{1}{2} l_{i+(1/2),j} \, |\mathbf{A}_{i+(1/2),j}| \, (\mathbf{q}_r - \mathbf{q}_l), \quad (5)$$

where $l_{i+(1/2),j}$ is the length of the control-volume edge, $\mathbf{n}_{i+(1/2),j}$ is the normal on the edge, $\mathbf{q}_l$ and $\mathbf{q}_r$ are appropriate left and right state vectors, the dot denotes the inner product and $\mathbf{A}_{i+(1/2),j}$ is the combined flux Jacobian obtained by taking the Jacobian of the inner product of the normal with the flux vector on the control-volume edge. The absolute value of the flux Jacobian matrix is defined as $|\mathbf{A}| = R|\Lambda|L$, where R and L are right and left eigenvector matrices of $\mathbf{A}$ and $|\Lambda|$ a diagonal matrix containing the absolute values of the eigenvalues of $\mathbf{A}$. As a result of this projection the flux Jacobian becomes a function of $\mathbf{q}_l$, $\mathbf{q}_r$ and the normal $\mathbf{n}_{i+(1/2),j}$

$$\mathbf{A}_{i+(1/2),j} = \mathbf{A}(\mathbf{q}_l, \mathbf{q}_r, \mathbf{n}_{i+(1/2),j}) = \mathbf{A}(\mathbf{q}_{lr}, \mathbf{n}_{i+(1/2),j}), \quad (6)$$

where $\mathbf{q}_{lr} = \mathbf{q}_{i+(1/2),j}$ denotes Roe's average state vector which we determine by calculating the primitive variables according to

$$u_{lr} = \frac{u_l + \mathrm{d}u_r}{1+d}, \qquad v_{lr} = \frac{v_l + \mathrm{d}v_r}{1+d}, \qquad H_{lr} = \frac{H_l + \mathrm{d}H_r}{1+d}, \quad (7)$$

with $d = \sqrt{\rho_l/\rho_r}$ and where H is the specific enthalpy. For the first order approximation of the flux the left state vector $\mathbf{q}_l$ equals $\mathbf{q}_{ij}$ and the right state vector $\mathbf{q}_r$ equals $\mathbf{q}_{i+1,j}$. To achieve a higher-order monotonic upwind method we interpolate the state vector with the MUSCL scheme [2] using the minmod-limiter

$$\begin{aligned} \mathbf{q}_l &= \mathbf{q}_{ij} + \tfrac{1}{4}[(1-\kappa)\mathrm{Lim}(\Delta\mathbf{q}_{i-(1/2)}, \omega\Delta\mathbf{q}_{i+(1/2)}) \\ &\qquad + (1+\kappa)\mathrm{Lim}(\Delta\mathbf{q}_{i+(1/2)}, \omega\Delta\mathbf{q}_{i-(1/2)})], \end{aligned} \quad (8)$$

$$\begin{aligned} \mathbf{q}_r &= \mathbf{q}_{i+1,j} - \tfrac{1}{4}[(1-\kappa)\mathrm{Lim}(\Delta\mathbf{q}_{i+(3/2)}, \omega\Delta\mathbf{q}_{i+(1/2)}) \\ &\qquad + (1+\kappa)\mathrm{Lim}(\Delta\mathbf{q}_{i+(1/2)}, \omega\Delta\mathbf{q}_{i+(3/2)})], \end{aligned} \quad (9)$$

with $\Delta\mathbf{q}_{i+(1/2),j} = (\mathbf{q}_{i+1,j} - \mathbf{q}_{ij})$, and the limiting function Lim( ) is defined as

$$\mathrm{Lim}(a, b) = \tfrac{1}{2}\,(\mathrm{sign}(a) + \mathrm{sign}(b)) \min(|a|, |b|). \quad (10)$$

The parameters κ and ω must obey

$$-1 \leqslant \kappa \leqslant 1, \quad (11)$$

$$1 \leqslant \omega \leqslant \frac{3-\kappa}{1-\kappa}, \tag{12}$$

to ensure monotonicity [7]. The scheme defined by (8) and (9) is called 'limited κ-scheme'. For the special choice of $\kappa = \frac{1}{3}$ the scheme is third order accurate in smooth regions [8]. In this paper we will use $\kappa = \frac{1}{3}$ and $\omega = \frac{3}{2}$. For a discussion of the $\kappa = \frac{1}{3}$ scheme see [9].

2.3. TIME-INTEGRATION METHOD

2.3.1. *Explicit time-integration method*

For the numerical calculations with explicit time-integration we use the compact-storage, four-stage Runge-Kutta method

$$\begin{aligned} \mathbf{q}_{ij}^0 &= \mathbf{q}_{ij}^{(n)}, \\ \mathbf{q}_{ij}^k &= \mathbf{q}_{ij}^0 - \Delta t \beta_k \mathbf{F}_{ij}(\mathbf{q}^{k-1}), \quad k = 1 \ldots 4, \\ \mathbf{q}_{ij}^{(n+1)} &= \mathbf{q}_{ij}^4, \end{aligned} \tag{13}$$

with $\beta_1 = \frac{1}{4}$, $\beta_2 = \frac{1}{3}$, $\beta_3 = \frac{1}{2}$ and $\beta_4 = 1$. $\mathbf{F}_{ij}$ is the total numerical flux in the grid point (i, j). Note that, thus defined, ij is in fact one single index. We have chosen this scheme because of its stability region and because it is widely used for different applications. We will not discuss this familiar method further.

2.3.2. *Implicit time-integration method*

In this section we formulate the implicit Euler Backward time-integration method. The discrete version of (1) for the Euler Backward scheme can be written as

$$\mathbf{q}_{ij}^{(n+1)} = \mathbf{q}_{ij}^{(n)} - \Delta t \mathbf{F}_{ij}(\mathbf{q}^{(n+1)}), \tag{14}$$

where the superscript (n) labels the time level and $\mathbf{F}_{ij}$ is again the numerical flux. The Euler backward scheme is only first order accurate in time, but this suffices since we are only interested in the steady state solution. A first order Taylor expansion of $\mathbf{F}$ around $\mathbf{q}^{(n)}$ yields

$$\left(\frac{\mathcal{I}}{\Delta t} + \frac{\partial \mathbf{F}}{\partial \mathbf{q}}(\mathbf{q}^{(n)}) \right) \cdot \Delta \mathbf{q} = -\mathbf{F}(\mathbf{q}^{(n)}), \tag{15}$$

where $\partial \mathbf{F}/\partial \mathbf{q}$ is the symbolic representation of the Jacobian matrix of $\mathbf{F}$ and $\Delta \mathbf{q}_{ij} = \mathbf{q}_{ij}^{(n+1)} - \mathbf{q}_{ij}^{(n)}$ the difference vector in grid point (i, j). For infinite Δt and exact Jacobian matrix this scheme is equal to Newton iteration for the nonlinear equation $\mathbf{F}(\mathbf{q}) = 0$. However, it is not possible to obtain the exact Jacobian matrix at a reasonable cost. Therefore, we approximate the Jacobian with the 'first order approximation', as in [10]. With this approximation, due to the five-point stencil of Roe's scheme in two-dimensional, we get a Jacobian matrix with only five bands of 4×4-matrices. The five blocks for a grid point (i, j) are given by

$$\begin{aligned} \mathbf{D}_{ij} &= \mathbf{A}^+_{i-(1/2),j} - \mathbf{A}^-_{i+(1/2),j} + \mathbf{A}^+_{i,j-(1/2)} - \mathbf{A}^-_{i,j+(1/2)}, \\ \mathbf{N}_{ij} &= \mathbf{A}^-_{i,j+(1/2)}, \\ \mathbf{S}_{ij} &= -\mathbf{A}^+_{i,j-(1/2)}, \\ \mathbf{E}_{ij} &= \mathbf{A}^-_{i+(1/2),j}, \\ \mathbf{W}_{ij} &= -\mathbf{A}^+_{i-(1/2),j}. \end{aligned} \tag{16}$$

The matrices $\mathcal{D}$, $\mathcal{N}$, $\mathcal{S}$, $\mathcal{E}$ and $\mathcal{W}$ stand for the matrices yielding the diagonal, north, south, east and west contributions for the grid point (i, j), if we use the notation $\mathcal{D}_{ij,kl} = \mathbf{D}_{ij}\delta_{ij,kl}$, $\mathcal{N}_{ij,kl} = \mathbf{N}_{ij}\delta_{i(j+1),kl}$ etc. The matrices $\mathcal{A}^+$ and $\mathcal{A}^-$ are determined by the positive and negative eigenvalues of the flux Jacobian matrix

$$\mathcal{A} = R\Lambda L = R(\Lambda^+ + \Lambda^-)L = R\Lambda^+L + R\Lambda^-L = \mathcal{A}^+ + \mathcal{A}^-. \tag{17}$$

The delta formulation in (15) allows the use of additional simplifications of the Jacobian matrix without changing the steady-state solution: if the iteration process converges it follows from (15) that the flux equals zero in all grid points and hence that the solution satisfies the stationary discrete equations. If we define the matrix on the left-hand side in (15) as $\mathcal{M}$, we can approximate $\mathcal{M}$ with ‘approximate factorization’

$$\mathcal{M} \;=\; \frac{\mathcal{I}}{\Delta t} + \mathcal{A} = \left(\frac{\mathcal{I}}{\Delta t} + \mathcal{D} + \mathcal{N} + \mathcal{S} + \mathcal{E} + \mathcal{W}\right) \tag{18}$$

$$\approx \left(\frac{\mathcal{I}}{\Delta t} + \mathcal{D} + \mathcal{S} + \mathcal{W}\right)\left(\frac{\mathcal{I}}{\Delta t} + \mathcal{D}\right)^{-1}\left(\frac{\mathcal{I}}{\Delta t} + \mathcal{D} + \mathcal{N} + \mathcal{E}\right). \tag{19}$$

in which $\mathcal{D}$, $\mathcal{N}$, $\mathcal{S}$, $\mathcal{E}$ and $\mathcal{W}$ are the contributions to $\mathcal{A}$ from the corresponding parts in (16). For small Δt, the error in the last line is $\mathcal{O}((\Delta t)^2)$ compared to the first term.

With this approximation the system of (15) consists of a lower, upper and diagonal matrix and can be solved in two steps according to

$$\begin{aligned}
\left(\frac{\mathcal{I}}{\Delta t} + \mathcal{D} + \mathcal{S} + \mathcal{W}\right)\Delta\mathbf{q}^* &= -\mathbf{F}(\mathbf{q}^{(n)}) \\
\left(\frac{\mathcal{I}}{\Delta t} + \mathcal{D} + \mathcal{N} + \mathcal{E}\right)\Delta\mathbf{q} &= \left(\frac{\mathcal{I}}{\Delta t} + \mathcal{D}\right)\Delta\mathbf{q}^* \\
&= -\mathbf{F}(\mathbf{q}^{(n)}) - \mathcal{S}\Delta\mathbf{q}^* - \mathcal{W}\Delta\mathbf{q}^*.
\end{aligned} \tag{20}$$

The above equations yield an approximate solution of (15), which has the general form $\mathcal{M}\cdot\Delta\mathbf{q} = -\mathbf{F}$. The last line in (20), which follows from the first line, shows that this factorization method can be regarded as two sweeps of a point Gauss-Seidel relaxation with initial guess $\Delta\mathbf{q} = 0$, and where the direction in which the variables are solved changes in the second sweep, but no intermediate update of the flux or the numerical Jacobian matrix is computed. This observation suggests several other approximate solution methods, because Gauss-Seidel sweeps can be varied in both direction and number of sweeps. This will be used in Section 4.2.

2.4. Boundary conditions

For the flow around an airfoil there are two types of boundaries: the far-field boundary, due to the finite extent of the computational domain, and the solid wall. In the far field we permit subsonic inflow or outflow. We use a method which takes the incoming and outgoing characteristics into account. Depending on whether the boundary is an inflow or an outflow boundary, we extrapolate one or three Riemann invariants from the inner field and set the remaining Riemann invariants to their values at infinity [6]. The far-field boundary conditions are treated implicitly in the numerical scheme.

The only physical condition for inviscid flow over a solid wall is the impermeability of the solid wall which is equivalent to the normal velocity on the solid wall being equal to zero. As numerical boundary conditions we extrapolate the density, the tangential velocity and the pressure.

To initialize the flow field, we set all dependent variables equal to their values at infinity determined by the Mach number, M_∞, and the angle of attack, α.

3. Multigrid acceleration

In the introduction we showed in Figure 1 the faster convergence towards the steady state of the implicit method as compared to the explicit time-integration method, the latter being limited to small CFL numbers for numerical stability reasons. A popular approach to accelerate the convergence towards the steady state for explicit Runge-Kutta time-stepping is the application of a multigrid technique. Because it is such a palpable question how the convergence of the present implicit solver compares to these accelerated explicit Runge-Kutta methods, we briefly address the application of implicit and explicit time-stepping in combination with multigrid.

We applied a multigrid technique described in [11, 12]. In this method, nonlinear multigrid is applied to solve $\mathbf{q}$ from the stationary Euler equations

$$\mathbf{F}(\mathbf{q}) = 0. \tag{21}$$

As a relaxation mechanism on the successive grids we use (pseudo) time-stepping for the nonstationary Euler equations, as this will push the solution towards the solution of (21). Both explicit Runge-Kutta schemes and the Euler Backward time-stepping method described in this paper can be used as time-stepping techniques.

This multigrid method is different from a multigrid technique where the multigrid is used to accelerate the solution of the linear system (15) in each time step, as described in [13]. In our implicit Euler Backward scheme, application of a multigrid method to solve the linear system is not attractive, because we only have an approximate Jacobian matrix, and accurate solution of the linear system will not pay off. The multigrid method used in this paper is closely related to the work reported in [14]. However, in [14] only first order spatial discretization is applied, and the relaxation method is not based on (pseudo) time-stepping.

In our multigrid method the solution is restricted to coarser grids by injection, and the defect vector by full weighting. The correction to the solution is prolonged to the finer grid by bilinear interpolation. With this choice of the intergrid operators the approximation property is satisfied [15]. The initial solution is improved with Full Multigrid, and each multigrid W-cycle follows the so called Full Approximation Scheme.

It is known from the literature (see Van der Burg [4]) that the multigrid method does not converge very well for the Euler equations with the MUSCL scheme and several explicit compact storage Runge-Kutta time-stepping schemes. This is confirmed by our experiments. In Figure 4 a result is shown for the four-stage explicit Runge-Kutta scheme (13) and CFL number of 0·6. The speedup in convergence rate compared to the single-grid computation is considerable, but the final residual level is not as low as in the single-grid explicit computation, and convergence itself is only attained for carefully chosen numbers of coarse grid, pre- and post-relaxations. On the other hand, the implicit method (*e.g.* at a CFL number of 50) is also accelerated by the multigrid process, but for a wide range of multigrid parameters and CFL

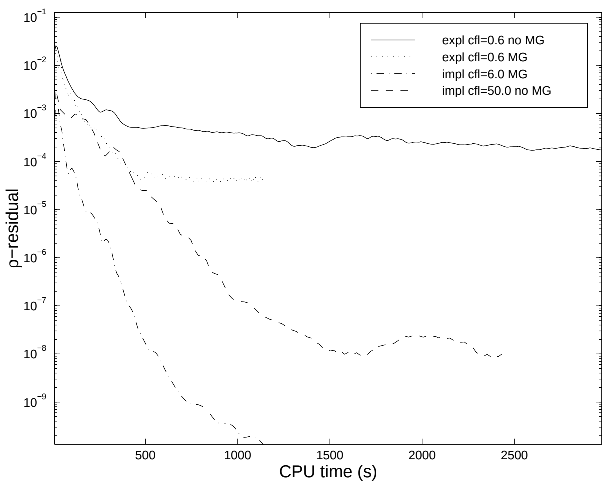


Figure 4. Convergence of the multigrid technique applied to attain the steady state of the Euler equations. For explicit Runge-Kutta time-stepping, the final residual level with multigrid ($\sim 10^{-4}$) is higher than in the single-grid case ($\sim 10^{-6}$, truncated in the picture) and convergence depends sensitively on the multigrid parameters, while for the implicit time-stepping method convergence acceleration is achieved for various CFL numbers and multigrid parameters. The results without multigrid of Figure 1 are also shown.

numbers. Again, the final level of the residual depends on the CFL number, but the final level found here is well below the final level of the explicit Runge-Kutta scheme of Figure 1, which is sufficient for engineering accuracy. By choosing different grid levels for the finest grid, we also found the convergence rate to be grid-independent. These results were confirmed with test computations for different free-stream conditions. The case $M = 0{\cdot}85$, $\alpha = 1{\cdot}0$ with stronger shocks, the subsonic case $M = 0{\cdot}63$, $\alpha = 2{\cdot}0$ without shocks and the supersonic case $M = 1{\cdot}2$, $\alpha = 7{\cdot}0$ all gave convergent multigrid processes when implicit time-stepping was used. The final residual levels and the convergence rates were found to be dependent on the CFL number as in the single-grid computations.

These results show that the implicit scheme has better smoothing properties than the explicit Runge-Kutta scheme [16]. The dissipation in the implicit scheme is sufficient to ensure a convergent multigrid process. For higher-order TVD schemes and explicit time-stepping, the special treatment of defect correction may be required to reduce the residuals at all, as found by Van der Burg [4]. (For a discussion of the application of defect correction, see, *e.g.*, [17].) It is noted that the problems with multigrid are typical for the use of a higher-order TVD scheme as the MUSCL scheme employed in this paper. For more dissipative spatial discretizations, such as the popular second-order Jameson scheme, convergent multigrid methods are more readily obtained, also with explicit time-stepping schemes.

For a proper comparison of the implicit and explicit time-stepping methods as smoothers for multigrid, explicit Runge-Kutta schemes should be tested different from the compact storage scheme of (13). Tests with the five-stage scheme proposed by Jameson [18] gave no significant improvement of the convergence behavior, which can be understood because the damping of high frequency components is similar for this scheme. This paper aims at demonstrating the power and simplicity of the implicit scheme. The property that ‘everything works’ without multigrid parameter study or defect correction, is a major advertisement for the application of the implicit scheme. More extensive research into the multigrid approach for the Euler equations is planned, especially in combination with parallel execution of the code.

4. Parallelization

According to the parallel version of Amdahl's law, the performance is limited by the fraction of the program that is necessarily sequential. In order to establish the theoretical parallel performance of an implicit solver as compared to the parallel performance of an explicit solver, it is sufficient to consider only that part of the implicit program that is essentially different from its explicit counterpart. The implicit program has three stages: the calculation of the Jacobian matrix and the numerical flux from the old solution, the matrix inversion described in Section 2.3.2, and the calculation of the new solution. Because the calculation of the Jacobian matrix is very similar to the calculation of the numerical flux, the first and third stages in the implicit algorithm do not differ in nature from any flux calculations in explicit programs. These stages can be done in parallel as in every explicit method with spatial domain decomposition. Therefore, for our present goal it is sufficient to consider only the solving phase (20) of the implicit factorization method to establish the theoretical parallelizability. In the present implementation this stage takes about thirty percent of the total computation time for each time step. If this portion is not parallelizable, this would lead, according to Amdahl's law, to a maximum speedup of 3 on any parallel computer, which would be a severe limitation on the performance of the solver.

4.1. Domain decomposition

Domain-decomposition methods have gained wide acceptance in solving the compressible Navier-Stokes or Euler equations. The suitability for parallelization is clear because of the presence of coarse grained loops over the subdomains in the program. These loops over the subdomains are usually limited in number and contain sufficient work to make parallel overhead relatively unimportant. Also, the possibility of local data storage is attractive, especially on distributed systems. Locality of data in this context means that the required data reside in the local memory of a processing element for the duration of the computation, with only exchange of data at the 'boundaries' of the data.

The calculation of the Jacobian matrix and the fluxes can be done with domain decomposition in a familiar way, by introducing dummy points. It seems worthwhile if the matrix inversion part can be treated similarly, because then all the important subroutines of the program can compute with local data, with only boundary data exchange at appropriate stages.

In the sequential version of the matrix inversion, we cover the grid typically by starting at a far-field point, *e.g.* a far-field corner point or the far-field point in the wake, and traversing the field systematically, taking at each point into account the values of the neighboring points. At each point, two out of four neighboring points have been updated already. In the second stage of (20), the field is crossed in the opposite direction. This has no advantage for parallel execution, because, if normal domain decomposition is applied, each processor has to wait until the processor of the neighboring domain has completed its computations. Nevertheless, domain decomposition is attractive, especially when other parts of the program, for instance the flux calculation, already use domain decomposition for parallel execution. This is in contrast to a parallel computation in which the grid points are renumbered, such as the 'wavefront method', which is designed to solve tridiagonal systems similarly to the method used to solve

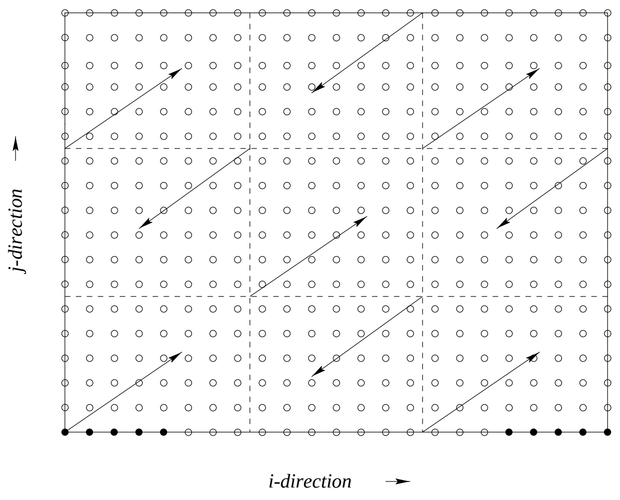


Figure 5. First stage in the traversal of the C grid in the factorization method with domain decomposition. In the second stage the arrows are reversed. The grid is shown in the computational domain. The dark grid points on the domain boundary are joint points in the wake behind the airfoil.

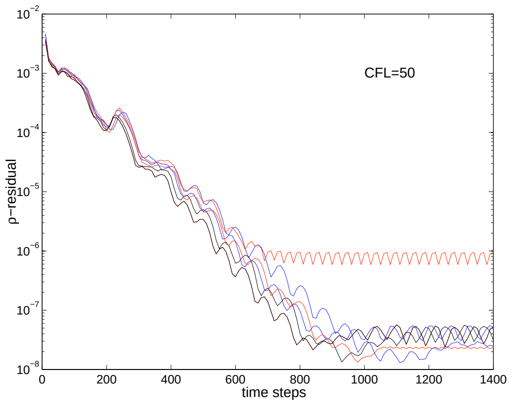
(20) in this paper. The wavefront method does give the correct results, but the method is not scalable, because at the beginning and at the end of the sweeps some processors are idle. Other disadvantages are that the data must be divided over the processors in a different way, taking less advantage of the structured grid, and that synchronization takes place after each wavefront line calculation.

Instead we choose the simplest form of domain decomposition, by ignoring the (small) errors made at the subdomain boundaries, and start the iteration in each subdomain independently of the other subdomains. To obtain the closest resemblance with the performance of the sequential code, we seek the best way of crossing the grid when solving (20). We found that the best results with respect to convergence rate and final residual level are obtained when the domain-decomposed field is crossed by starting out from opposite corners in each pair of adjacent subdomains in the first stage, exchanging boundary data with the neighboring subdomains after completing the first sweep, and then proceeding in the opposite direction, returning to the starting point. This is depicted in Figure 5. Thus the field is traversed in a 'continuous' manner, in the sense that after each sweep each block starts with new values, either because of a flux and Jacobian update after a whole time step, or because new values have been 'supplied' by the neighboring block.

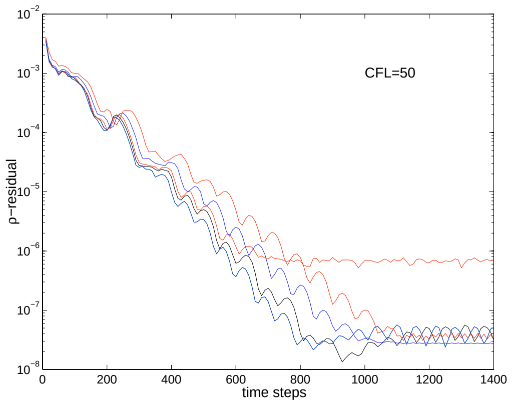
However, if the number of subdomains is larger than three, the two sweeps of the iterative process do not cover the entire grid, as the information from the boundaries travels only across two adjacent blocks after two stages. This is likely to influence the overall convergence of the method. Because the gain of parallel computation may outweigh the loss of convergence, it is necessary to study the convergence rate for several domain divisions.

4.2. NUMERICAL RESULTS

In this section the convergence behavior for several domain divisions is investigated. We studied the convergence for several divisions up to a total of sixteen subdomains, where the number of subdomains is varied in both the i- and the j-direction. Possible vertical block lines in the wake region were chosen to be continuous across the wake line. The block divisions are



(a)



(b)

Figure 6. Convergence behavior of the approximate factorization method at a CFL number of 50 (a) for domain divisions 1×1, 2×1, 4×1, 5×1, 8×1 and 16×1 in the i-direction and (b) for domain divisions 1×1, 2×1, 2×2, 2×4, and 2×8 in the j-direction. The fastest convergence is obtained with the 1×1 division (most left curve in both Figures (a) and (b)). Note the two simulations 5×1 and 2×2 with different final residual level.

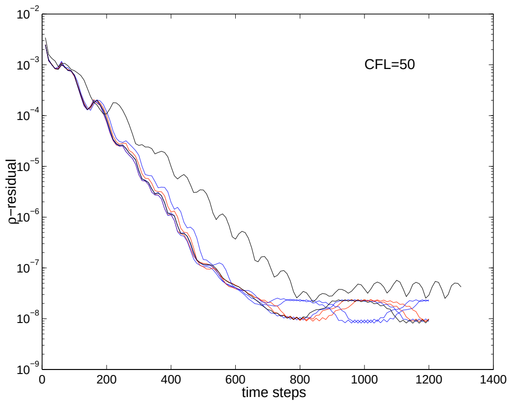
indicated as $n_i \times n_j$, n_i being the number of divisions in the i-direction and n_j the number in the j-direction. The division in Figure 5 is 3×3.

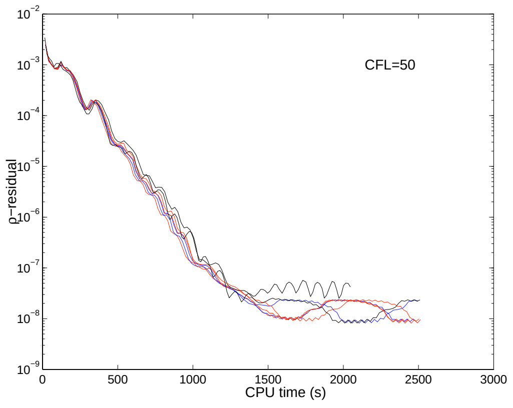
In Figure 6 convergence rates are plotted for various subdomain divisions. The fastest convergence is achieved with the 1×1 single-block division as expected. The convergence deterioration for different subdomain divisions is, however, rather small.

The domain decomposition is seen to have two effects. Firstly, the convergence rate is smaller if more subdomains are used. Moreover, better results are obtained with more divisions in the i direction than in the j direction. This suggests that the convergence-rate deterioration is related to the number of interface points created by the subdomain division: because of the 289×65 grid, a single bisection in the j-direction creates a larger boundary portion than a bisection in the i-direction.

Secondly, the final convergence level may occasionally deviate considerably from the expected value. Because it is undesirable that the final results are influenced by the block division, this problem should be eliminated. The solution to this problem is obtained from the fact that the approximate factorization method gives a solution of the linear system only approximately. Our computations (we will come to this point shortly) suggest that the solution of the linear system with approximate factorization is merely accurate enough to retain convergence. The difference between the single-block grid and domain-decomposed grid is that for the latter internal block boundary points are treated with Jacobi relaxation instead of Gauss-Seidel relaxation in the first sweep, such that the accuracy required for fast convergence behavior is no longer attained. This suggests that we should solve the linear system more accurately when more blocks are used. To obtain a more accurate solution we first experimented with four Gauss-Seidel sweeps rather than two as prescribed by the approximate factorization method. The results have been plotted in Figure 7a, where the residual of the density has been plotted as a function of the number of outer iterations for several domain divisions.

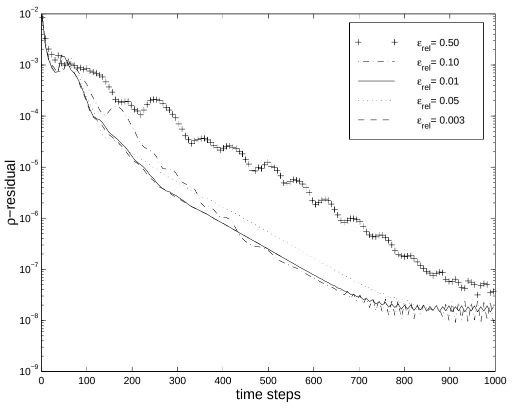
At the cost of more CPU time per outer iteration step, the simulations in Figure 7a display a more favorable convergence behavior than the simulations in Figure 6a with the same

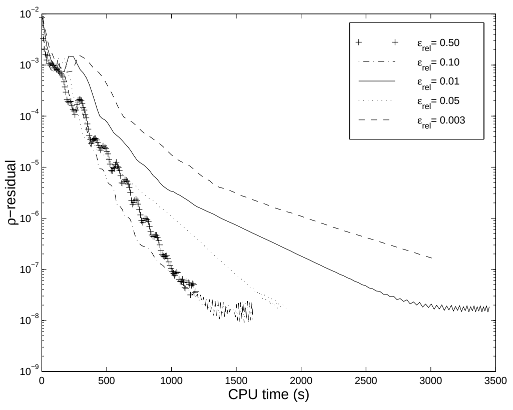




(a) (b)

Figure 7. Density residual (a) as a function of the number of outer iterations (time steps) and (b) as a function of CPU time, for various domain divisions, but calculated with *four* Gauss-Seidel sweeps per outer iteration. The solid line (most right in Figure (a)) is the original single-block line of the approximate factorization method with two Gauss-Seidel sweeps. The different subdomain divisions can hardly be discerned.





(a) (b)

Figure 8. Density residual (a) as a function of the number of outer iterations (time steps) and (b) as a function of CPU time, for increasing accuracy of the solution of the linear system. The convergence rate hardly improves for $\varepsilon_{\mathrm{rel}} < 0{\cdot}05$ as a function of the number of time steps, while the computation time increases with increasing accuracy.

block divisions. Firstly, the convergence stall is eliminated, which was the principal goal. And secondly, the convergence towards the steady state requires fewer time steps, and the convergence rate (in terms of time steps) is also less influenced by the domain decomposition. If the residual level is plotted as function of the CPU time instead of time steps, we even see that the improved convergence behavior leads to no extra computational costs (Figure 7b).

The convergence rate of the residual is dependent on the accuracy of the solution of the linear system. To quantify this, we introduce the relative error $\varepsilon_{\mathrm{rel}}$ in the solution $\Delta\mathbf{q}$ as

$$\varepsilon_{\mathrm{rel}} = \frac{||\mathcal{M}\cdot\Delta\mathbf{q} + \mathbf{F}||}{||\mathbf{F}||}, \tag{22}$$

where $||\cdot||$ is the discrete L_2 norm of the state vector over the entire grid. We have repeated the computations with increasing numbers of Gauss-Seidel sweeps until a prescribed relative error $\varepsilon_{\mathrm{rel}}$ is reached at each time step. In Figure 8a the convergence history is plotted as a function of outer iteration steps and in Figure 8b as a function of the CPU time. Figure 8a shows that we cannot improve the convergence rate indefinitely by solving the linear system of equations more accurately, because the linear system contains only an approximation of the exact Jacobian (see, *e.g.*, Venkatakrishnan [19] or Tan [20], and references quoted therein). An optimal accuracy regarding computational performance is found for $\varepsilon_{\mathrm{rel}}$ between 0·1 and 0·5. Between these numbers the actual CPU time is of the same order of magnitude (Figure 8b). This relative error range corresponds to roughly one to six Gauss-Seidel sweeps. Because, apparently, a large relative error in the solution is allowed, the use of sophisticated linear solvers like GMRES or Conjugent Gradient methods [21] will probably not be very rewarding in view of CPU costs.

To demonstrate the effectiveness of the Gauss-Seidel relaxation we plotted the relative error $\varepsilon_{\mathrm{rel}}$ as function of sweeps or ‘iterations’ in Figure 9 for Red-Black relaxation and for point Gauss-Seidel methods with different sweep directions. From this figure it is clear that the alternating or symmetric Gauss-Seidel method is most efficient in reducing the error in the solution of the linear system, although the asymptotic convergence rates of the displayed methods are not very different. This shows that, because of the modest accuracy that is required for the implicit method described in this paper, the efficiency of linear solvers is determined by their performance in the initial sweeps, and not by the asymptotic convergence rate.

We verified that, as long as the relative error as defined by (22) is prescribed, the outer convergence rate, *i.e.* the required number of time steps towards the steady state, is retained, even if different approximate linear solvers, like for instance Red-Black relaxation, are used. Moreover, it was checked that, if domain decomposition is applied and $\varepsilon_{\mathrm{rel}}$ is prescribed rather than the number of Gauss-Seidel relaxations, both the outer convergence rate and final convergence level are preserved for all subdomain divisions, although possibly now at the price of more Gauss-Seidel relaxations.

We remark that the results of this section indicate that the use of square subdomains is favorable for the total computation time. We have seen that the loss of convergence rate is mainly due to a less accurately solved linear system. Because domain decomposition introduces internal boundary points which are treated with Jacobi rather than Gauss-Seidel relaxation, on average more relaxation sweeps are required to obtain the same solution-vector accuracy. Therefore, subdomain divisions with the least number of internal boundary points, *i.e.* square subdomains, are favorable. This was already mentioned to explain the preference for bisections in the i-direction rather than in the j-directions. Although it is not inconceivable that the flow geometry introduces some preference direction for subdomain divisions, the present results suggest that the numerical error caused by the simplified treatment of internal boundary points is proportional to the *number* of internal boundary points, and outweighs any physical preference direction.

To demonstrate the parallel performance, parallel computations were done on a Cray T3E distributed memory computer. As an example we show the performance for domain decomposition with subdomain divisions only in the i-direction, the number of subdomains matching the number of processors exactly. In the case of 32 CPUs this is hardly optimal, because the subdomains are far from being square, but the performance is still quite good for this relatively small problem, as shown in Figure 10. We have verified that the parallel performance is even better if the grid is refined once in both the i and j directions. This improvement can be predicted roughly with a model based on the ratio of interior and boundary points and the problem size.

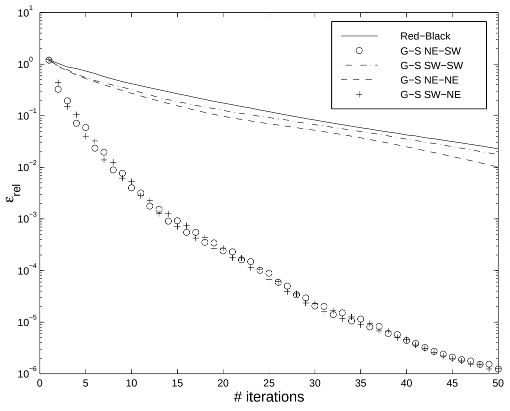


Figure 9. Relative error ε_{rel} as a function of the number of iterations for Red-Black relaxation and for point Gauss-Seidel relaxations with different sweep directions. Note that one Red-Black relaxation is defined as the treatment of both the 'red' and the 'black' points. In this plot the alternating (or symmetric) Gauss-Seidel relaxation is most efficient. Note that two iterations of Symmetric Gauss-Seidel corresponds with the original 'approximate factorization' approach.

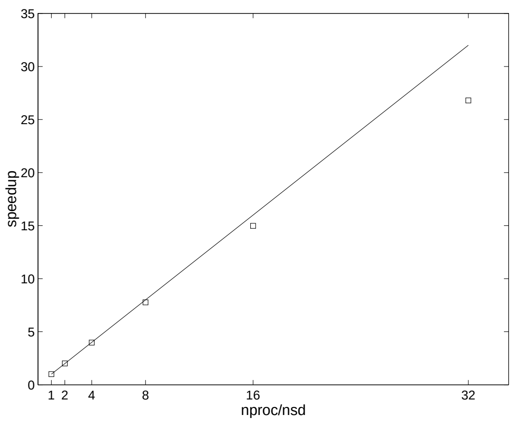


Figure 10. Parallel speedup as a function of the number of CPUs on the the Cray T3E. The domain decomposition was only done in the i-direction, and the number of subdomains was chosen equal to the number of processing elements.

5. Conclusions

In this paper we discussed the parallel performance of an implicit solver based on quasi-Newton iteration and an approximate factorization method. It is found that the convergence of the outer iteration is determined by the accuracy of the solution of the linear system that results from the quasi-Newton iteration, and that 'approximate factorization' is merely a cheap, two-stroke point Gauss-Seidel method to solve the linear system of equations with sufficient accuracy. The relative error ε_{rel} defined by (22) is a suitable measure for outer iteration control when different linear solvers are used. Because the accuracy required to achieve convergence is low, the high efficiency of the initial sweeps of the cheap alternating point Gauss-Seidel relaxation pays off. It was found that Red-Black relaxation for this problem is between two or three times less efficient than Gauss-Seidel. The use of more expensive linear solvers based on Krylov projection [21] will probably not be very profitable. The theoretical asymptotic convergence rates of these iteration schemes are not very important if the required accuracy is as low as described in this paper.

The implicit time-stepping method described here is also very suitable when it is used as a multigrid smoother. Convergent multigrid results are obtained without a multigrid parameter study, in contrast to multigrid with explicit Runge-Kutta time-stepping, for which convergence depends sensitively on the multigrid parameters .

With straightforward domain decomposition a parallelizable multiblock code is obtained with roughly the same convergence rates as the single-domain problem. As discussed in the introduction, the convergence rate and final level of residuals depends on the CFL number. The domain-decomposition method affects the accuracy of solving the linear system of equations resulting from the quasi-Newton method. By prescribing a fixed accuracy of the solution of the linear system, we retain the convergence rate and the final residual level of the Euler Backward time-stepping for the single block code. Divisions up to 50 subdomains and more were checked to have no serious effect on the overall computation time. This does not only boost the theoretical performance of the solver, but the apparent insensitiveness to the domain decomposition is also promising for cases where the geometry is less simple.

In the introduction it was mentioned that perfect scaling with the number of processors is not achieved. Should this cause any problems, then it is possible turn to different iterative solution methods of the linear system, such as the Red-Black method. With this method, if communication costs are neglected, perfect scaling is possible, because for each subdomain division the same numerical problem is solved. Experiments with different relaxation methods show that, as long as the linear system is solved with the same accuracy, similar convergence behavior is found with respect to the number of time steps required and the final residual level.

Although in this paper we studied inviscid flow with the Euler equations, we expect similar results for the compressible Navier-Stokes equations. Preliminary tests with Navier-Stokes confirm our expectations: the method described in this paper has been applied successfully to DNS of a two-dimensional shock boundary-layer interaction flow over a flat plate [22]. Also, extensions to three-dimensional problems will not pose problems regarding the algorithm. In [23] a similar numerical method was used for three-dimensional Navier-Stokes: implicit time-stepping, Newton's method and factorization.

Acknowledgements

Part of this work was supported by the HPCN program NICE (Netherlands Initiative for CFD in Engineering). This work was sponsored by the Stichting Nationale Computerfaciliteiten (National Computing Facilities Foundation, NCF) for the use of supercomputer facilities, with financial support from the Nederlandse Organisatie voor Wetenschappelijk Onderzoek (Netherlands Organization for Scientific Research, NWO).

References

1. R. van Buuren, J. G. M. Kuerten, and B. J. Geurts, Instabilities of stationary inviscid compressible flow around an airfoil. *J. Comp. Phys.* 138 (1997) 520–539.
2. B. van Leer, Towards the ultimate conservative difference scheme ii. Monotonicity and conservation combined in a second-order scheme. *J. Comp. Phys.* 29 (1974) 11–31.
3. H. Yoshihara and P. Sacher, *Test Cases for Inviscid Flow Field Methods.* Technical report, AGARD-AR-211 (1985).
4. J. W. van der Burg, *Numerical Methods for Transonic Flow Calculations*. PhD Thesis, University of Twente, Enschede, The Netherlands (1992) 171pp.

5. R. Leveque, *Numerical Methods for Conservation Laws*. Lectures in Mathematics, ETH Zürich, Basel, Birkhäuser Verlag (1992) 214pp.
6. P. L. Roe, Characteristic-based schemes for the Euler equations. *Ann. Rev. Fluid Mech.* 18 (1991) 337–365.
7. H. C. Yee, *Computational Fluid Dynamics*. Lecture Series 1989-04, Von Karman Institute for Fluid Dynamics (1989).
8. B. van Leer, Upwind-difference methods for aerodynamic problems governed by the Euler equations. In: B. E. Engquist, S. Osher, and R. C. J. Somerville (eds.), *Lectures in Applied Mathematics*, 22(2), American Mathematical Society, Rhode Island (1985) 327–336.
9. W. Hundsdorfer, B. Koren, M. van Loon, and J. G. Verwer, A positive finite-difference advection scheme. *J. Comp. Phys.* 117 (1995) 35–46.
10. H. C. Yee, Construction of explicit and implicit shock capturing methods. *J. Comp. Phys.* 68 (1987) 151–179.
11. J. W. van der Burg, J. G. M. Kuerten, and P. J. Zandbergen, Multigrid and Runge-Kutta time stepping applied to the uniformly non-oscillatory scheme for conservation laws. *J. Eng. Math.* 25 (1991) 243–263.
12. H. Kuerten and B. Geurts, Multigrid acceleration of a block structured compressible flow solver. *J. Eng. Math.* 14 (1995) 361–370.
13. W. A. Mulder, Multigrid relaxation for the Euler equations. *J. Comp. Phys.* 60 (1985) 235–252.
14. P. W. Hemker and S. P. Spekreijse, Multiple grid and osher's scheme for the efficient solution of the steady Euler equations. *App. Num. Math.* 2 (1986) 475–493.
15. W. Hackbusch, *Multi-grid Methods and Applications*. Springer Series in Computational Mathematics. Springer-Verlag, Heidelberg (1985) 377pp.
16. P. Wesseling, *An Introduction to Multigrid Methods*, Chapter 7. Chichester, John Wiley & Sons (1992) 284pp.
17. M.-H. Lallemand and B. Koren, Iterative defect correction and multigrid accelerated explicit timestepping schemes for the steady state Euler equations. *SIAM J. Sci. Comp.* 14 (1993) 953–970.
18. A. Jameson and T. J. Baker, Multigrid solution of the Euler equations for aircraft configurations. Technical report, AIAA-paper 84-0093 (1984).
19. V. Venkatakrishnan, Preconditioned conjugate gradient methods for the compressible Navier-Stokes equations. *AIAA J.* 29(7) (1991) 1092–1100.
20. K. H. Tan, *Local Coupling in Domain Decomposition*. PhD Thesis, University of Utrecht, Utrecht, The Netherlands (1995) 167pp.
21. R. Barrett, M. Berry, T. F. Chan, J. Donato J. Demmel, J. Dongarra, R. Pozo V. Eijkhout, C. Romine, and H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, Philadelphia, PA (1994) 124pp.
22. R. van Buuren, J. G. M. Kuerten, and B. J. Geurts, Implicit time accurate simulation of unsteady flow. Submitted.
23. M. M. Rai and P. Moi, Direct numerical simulation of transition and turbulence in a spatially evolving boundary layer. *J. Comp. Phys.* 109 (1993) 169–192.