Numerical Experiments with the Multiresolution Scheme for the Compressible Euler Equations

BJÖRN SJÖGREEN

Department of Scientific Computing, Uppsala University, Box 120, S-751 04, Uppsala, Sweden

Received June 2, 1994

We investigate the performance of the multiresolution method. The method was recently proposed as a way to speed up computations of compressible flows. The method is based on truncating wavelet coefficients. The method showed good performance on one-dimensional test problems. We implement the method in two space dimensions for a more complex compressible flow computation, intended to simulate conditions under which many production CFD codes are running. Our conclusion is that we can in many cases reduce the CPU time, but that the gain in efficiency is not as large as for the one-dimensional problems. We furthermore observe that it is essential to use the adaptive multiresolution method, which mixes centered differences with TVD methods. © 1995 Academic Press, Inc.

1. INTRODUCTION

We implement and test the multiresolution scheme for the two-dimensional Euler equations on a curvilinear grid. The multiresolution scheme is a method which is used to speed up computations with expensive ENO or TVD schemes. It is based on a wavelet decomposition of the solution to flag points where the solution is nonsmooth. In these points the numerical flux is computed as usual, using an ENO or TVD method; in the other points, where the solution is smooth enough, the numerical flux can be computed by an inexpensive interpolation from surrounding grid points. The method has recently been proposed and shown to have good preformance when used for hyperbolic model problems in one space dimension [1, 2].

The purpose of this work is to evaluate how this method works in a practical CFD computation. We give details about implementation at boundaries and in two space dimensions. In [2], an efficiency of around 5 was reported for a one-dimensional Riemann problem for the compressible Euler equations. Can we obtain a similar performance for a more complex flow in two space dimensions?

2. MULTIRESOLUTION REPRESENTATION OF DATA

In this section, we describe the method by giving a brief summary of [1].

Consider the approximation of a PDE in one space dimension, on the uniform grid $x_j = j\,\Delta x, j = 0, 1, ..., N$. The numerical solution is given by $(u_0, u_1, ..., u_N)$, with u_j an approximation to the solution at x_j. Introduce L levels of successively coarser grids,

$$G_k = (x_0, x_{2^k}, ..., x_N), \quad k = 0, ..., L.$$

Let x_j^k denote grid point j on grid G_k. Then $x_j^0 = j\,\Delta x$ and $x_j^k = x_{j2^k}^0 = j2^k\,\Delta x$. Let N_k denote the number of points in G_k. Then $N_k = 2^{L-k}N_L$.

We let u_j^k denote the numerical solution of a PDE at the point x_j^k.

Assume that the solution is given on grid G_k and that we want to find it on the finer grid G_{k-1} (Fig. 2.1). For the even numbered grid points we have

$$u_{2j}^{k-1} = u_j^k, \quad j = 0, 1, ..., N_k;$$

to find the solution at the odd grid points, we let $I(x, u^k)$ interpolate u_j^k on G_k, such that $I(x_j^k, u^k) = u_j^k$. We then have the approximation $\hat{u}_{2j-1}^{k-1} = I(x_{2j-1}^{k-1}, u^k)$ to u_{2j-1}^{k-1}. The interpolation error is $d_j^k = u_{2j-1}^{k-1} - \hat{u}_{2j-1}^{k-1}$. Thus with knowledge of u^k and d^k we can reconstruct the solution on G^{k-1}. We call (u^k, d^k) the multiresolution representation of u^{k-1}. Note that the vectors u^k, d^k together contain the same number of elements as does u^{k-1}. In summary, we switch between the representation u^{k-1} and (d^k, u^k) by the forward transformation

$$\begin{aligned} u_j^k &:= u_{2j}^{k-1} \\ d_j^k &:= u_{2j-1}^{k-1} - I(x_{2j-1}^{k-1}, u^k) \end{aligned} \tag{2.1a}$$

and the backward transformation

$$\begin{aligned} u_{2j}^{k-1} &:= u_j^k \\ u_{2j-1}^{k-1} &:= d_j^k + I(x_{2j-1}^{k-1}, u^k). \end{aligned} \tag{2.1b}$$

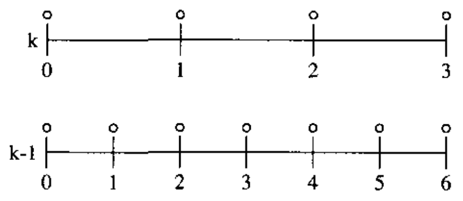

FIG. 2.1. Alignment of grids for finite differences.

This is inexpensive if $I(x_{2j-1}^{k-1}, u^k)$ is a straightforward linear interpolation operator.

We transform consequtively on all grids

$$u^0 \to (d^1, u^1) \to (d^1, d^2, u^2) \to \dots (d^1, d^2, \dots, d^L, u^L).$$

The vectors $(d^1, d^2, \dots, d^L, u^L)$ are the multiresolution representation of u^0. The interpolation errors d^k contain information about the smoothness of the solution. In the multiresolution scheme given in [1], the multiresolution representation is used to flag the nonsmooth parts of the solution, which can be used in adaptive numerical techniques. If d_j^k is large at some grid points, this indicates that the solution is nonsmooth there. There will be no direct computations of the solution involving the d^k coefficients; they are only used as smoothness monitors.

In the context of cell averages, similar formulas can be derived.

Let u_j denote an approximation to the cell average

$$\frac{1}{\Delta x}\int_{x_j}^{x_{j+1}} u(x)\,dx \tag{2.2}$$

over the cell $[x_j, x_{j+1}]$. Use the grids G_k as defined above. From the cell average property (2.2), we obtain

$$u_j^k = (u_{2j}^{k-1} + u_{2j+1}^{k-1})/2.$$

Figure 2.2 clarifies the correspondance between G_k and G_{k-1}. To find the value u_{2j+1}^{k-1} by interpolation from G_k, we proceed through reconstruction by the primitive function; i.e., we evaluate the primitive function

$$P_j^k = \sum_{m=-\infty}^{j} u_m^k \,\Delta x$$

on G_k. Then P^k is interpolated and the interpolant differentiated to obtain the approximation

$$\hat{u}_{2j+1}^{k-1} = \frac{P_{j+1}^k - I(x_{2j+1}^{k-1}, P^k)}{\Delta x}.$$

Note that P_j^k are point centered and thus enumerated according to Fig. 2.1. Given this approximation, we define as in the point-centered case the transformation

$$d_j^k := u_{2j+1}^{k-1} - \hat{u}_{2j+1}^{k-1} \tag{2.3a}$$

$$u_j^k := \tfrac{1}{2}(u_{2j}^{k-1} + u_{2j+1}^{k-1}). \tag{2.3b}$$

Thus we can define the multiresolution coefficients d_j^k for finite volume as well as finite difference approximations.

We conclude this section by giving the exact formulas used in our computations.

For finite differences we use the grid alignment in Fig. 2.1 and the formula (2.1) with

$$\begin{aligned}
I(x_{2j-1}^{k-1}, u^k) &= \tfrac{1}{16}(-u_{j-2}^k + 9u_{j-1}^k + 9u_j^k - u_{j+1}^k),\\
& \quad j = 2, \dots, N_k - 2,\\
I(x_{2j-1}^{k-1}, u^k) &= \tfrac{1}{16}(5u_{j-1}^k + 15u_j^k - 5u_{j+1}^k + u_{j+2}^k),\\
& \quad j = 1,\\
I(x_{2j-1}^{k-1}, u^k) &= \tfrac{1}{16}(35u_j^k - 35u_{j+1}^k + 21u_{j+2}^k - 5u_{j+3}^k),\\
& \quad j = 0,
\end{aligned} \tag{2.4}$$

which are fourth-order accurate formulas, including the boundary points. The formulas for the boundary points $j = N_k - 1$, N_k are symmetric with $j = 0, 1$ and are not given.

For finite volumes we use the grid alignment in Fig. 2.2 and the formula (2.3), with

$$\begin{aligned}
\hat{u}_{2j+1}^{k-1} &= \tfrac{1}{8}(-u_{j-1}^k + 8u_j^k + u_{j+1}^k), \quad && j = 1, \dots, N_k - 2,\\
\hat{u}_{2j+1}^{k-1} &= \tfrac{1}{8}(5u_j^k + 4u_{j+1}^k - u_{j+2}^k), && j = 0,\\
\hat{u}_{2j+1}^{k-1} &= \tfrac{1}{8}(11u_j^k - 4u_{j-1}^k + u_{j-2}^k), && j = N_k - 1,
\end{aligned} \tag{2.5}$$

which are third-order accurate formulas, including the boundary cells. Note that here u_j^k is a cell average.

3. MULTIRESOLUTION NUMERICAL SCHEME

Assume that we want to solve the hyperbolic partial differential equation

$$u_t + f(u)_x = 0, \quad -\infty < x < \infty,\ t > 0,$$

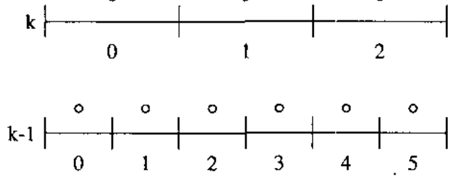

FIG. 2.2. Alignment of grids for finite volumes.

where initial data $u(0, x) = u_0(x)$ are given. We approximate by a difference method in conservative form,

$$u_j^{n+1} = u_j^n - \lambda(h_{j+1/2}^n - h_{j-1/2}^n)$$

on the grid G_0; u_j^n is the approximation of the cell average of the solution in the cell $[x_{j-1/2}, x_{j+1/2}]$ at the time $t_n = n\,\Delta t$, where Δt is the time step. The numerical flux function, $h_{j-1/2}^n = h(u_{j+p}^n, u_{j+p-1}^n, ..., u_{j-q}^n)$ is consistent with the physical flux, i.e., $h(u, u, ..., u) = f(u)$.

In [2] it is described how the multiresolution decomposition in Section 2 can be used to evaluate the numerical fluxes very efficiently. We next give a summary of this method.

The idea of the multiresolution method is as follows. All numerical fluxes are evaluated in the points of the coarsest grid, and for the finer levels, the numerical fluxes are evaluated only when the smoothness monitor, d^k is larger than some given tolerance, otherwise the fluxes are obtained by interpolation from the coarser level. The assumption in the multiresolution method is that the numerical flux $h_{j+1/2}^n$ is considerably more expensive to evaluate than to compute it by interpolation from neighboring points. This is the case in, e.g., the ENO method. The multiresolution method contains three ingredients:

1. A multiresolution representation to flag the smooth parts of the solution. We write this as

$$u_0 \to (d^1, d^2, ..., d^L, u^L),$$

where d^k contains the interpolation errors. This was described in Section 2.

2. A truncation algorithm to decide where to evaluate the numerical flux function, and where to interpolate it

$$(d^1, d^2, ..., d^L, u^L) \to (m^1, m^2, ..., m^L),$$

where m_j^k contains 0 or 1 to flag whether a flux evaluation is necessary or not. This array is described below.

3. A reordering of the flux evaluation. We start by evaluating the numerical fluxes on the coarsest grid, G_L,

$$h_{j2^L-1/2} = h(u_{j2^L+p}^n, u_{j2^L+p-1}^n, ..., u_{j2^L-q}^n), \quad j = 0, ..., N_L.$$

Note that the fluxes are evaluated using the points on the fine grid. Once the fluxes are given on G_k, we compute new fluxes at G_{k-1} as

$$h_{2j+1/2}^{k-1} = \begin{Bmatrix} h(u_{i+p}^n, ..., u_{i-q}^n)\ i = (2j+1)2^{k-1} & \text{if } m_j^k = 1 \\ I(x_{2j+1/2}^{k-1}, h^k) & \text{if } m_j^k = 0 \end{Bmatrix}$$
$$j = 0, ..., N_k - 1.$$

The coefficients d_j^k are used to decide how to evaluate the numerical flux function. If d_j^k is small, then the solution is smooth and the flux $h_{j+1/2}$ can be obtained by interpolating from fluxes given in neighboring points. If d_j^k is large, the solution is irregular and $h_{j+1/2}$ is evaluated using a shock capturing method. The assumption here is that the interpolation of $h_{j+1/2}$ is less costly than the direct evaluation of $h_{j+1/2}$ and, thus, that a substantial gain in efficiency can be obtained. This is the case if an ENO method is used.

The truncation algorithm given in [1], consists of the steps shown in Algorithm 3.1. We assume a finite volume approximation. The algorithm computes the flags m_j^k from the given multiresolution coefficients d_j^k.

ALGORITHM 3.1.

```
$m_j^k = 0, \quad 0 \le j \le N_k - 1 \quad 1 \le k \le L$
**for** $j := 0$ **to** $N_k - 1$ **do**
    **if** $|d_j^k| > \varepsilon_k$ **then**
        $m_{j-l}^k = 1, l = -1, 0, 1$
        **if** $|d_j^k| > 2^p\varepsilon_{k-1}$ **then**
            $m_{2j+1}^{k-1} = 1$
            $m_{2j}^{k-1} = 1$
        **endif**
    **endif**
**endfor**
```

The algorithm is made such that we apply the flux interpolation to compute the smooth values of the function; i.e., we need smoothness at time t_{n+1}. Thus the first test, $|d_j^k| > \varepsilon_k$, flags that the solution is nonsmooth at x_j^k, and to take into account that the nonsmoothness can spread to neighboring points (assuming a CFl number less than one), the neighboring points $j - 1$, $j + 1$, are flagged too. The test $|d_j^k| > 2^p\varepsilon_{k-1}$, is a test for formation of shocks, i.e., the gradients are becoming sharper, p is the expected convergence rate of the solution; for details see [1]. It is shown in [2] that the proper choice is $\varepsilon_k = \varepsilon/2^k$. We thus have two parameters to tune, ε and p.

4. IMPLEMENTATION IN TWO SPACE DIMENSIONS

The two-dimensional compressible Euler equations is a system of PDE on the form

$$\mathbf{w}_t + \mathbf{F}(\mathbf{w})_x + \mathbf{g}(\mathbf{w})_y = \mathbf{0},$$

where $\mathbf{w} = (\rho, \rho u, \rho v, e)$ with density ρ, x-velocity u, y-velocity v, and total energy e.

We approximate these equations on a curvilinear grid of $n_i + 1 \times n_j + 1$ grid points,

$$\{(x(\xi, \eta), y(\xi, \eta)) | (\xi, \eta) \in [0 \dots n_i] \times [0 \dots n_j]\}.$$

With this grid, the Euler equations are transformed into

$$\mathbf{w}_t + \frac{1}{A}((y_\eta \mathbf{f}(\mathbf{w}) - x_\eta \mathbf{g}(\mathbf{w}))_\xi + (-y_\xi \mathbf{f}(\mathbf{w}) + x_\xi \mathbf{g}(\mathbf{w}))_\eta) = \mathbf{0}$$

in the uniform (ξ, η) space. Here $A = x_\xi y_\eta - x_\eta y_\xi$. We consider numerical methods in conservative form

$$\frac{d\mathbf{w}_{i,j}}{dt} = -\frac{1}{A_{i,j}}(\Delta_{+i}\mathbf{h}_{i+1/2,j} + \Delta_{+j}\mathbf{g}_{i,j+1/2}).$$

We use a cell centered method; i.e., the dependent variables $\mathbf{w}_{i,j}$ are stored at the midpoint of the cell formed by the corners (i, j), $(i + 1, j)$, $(i, j + 1)$, $(i + 1, j + 1)$ of the grid. The numerical fluxes $\mathbf{h}_{i+1/2,j}$, $\mathbf{g}_{i,j+1/2}$ are approximating the fluxes $y_\eta \mathbf{f}(\mathbf{w}) - x_\eta \mathbf{g}(\mathbf{w})$ and $-y_\xi \mathbf{f}(\mathbf{w}) + x_\xi \mathbf{g}(\mathbf{w})$ on the cell interfaces.

We use a first-order, or a second-order, accurate Roe's method [3] for the numerical fluxes. The first-order method has a three point stencil, and the numerical flux function depends on two arguments

$$\mathbf{h}^{(1)}_{i+1/2,j} = \mathbf{h}(\mathbf{u}_{i+1,j}, \mathbf{u}_{i,j}).$$

The second-order extension is obtained from a piecewise linear reconstruction with a slope $\mathbf{s}_{i,j} = \mathbf{s}(\mathbf{u}_{i+1,j} - \mathbf{u}_{i,j}, \mathbf{u}_{i,j} - \mathbf{u}_{i-1,j})$ in each grid cell. The second-order numerical flux then becomes

$$\mathbf{h}^{(2)}_{i+1/2,j} = \mathbf{h}(\mathbf{u}_{i+1,j} - \mathbf{s}_{i+1,j}/2, \mathbf{u}_{i,j} + \mathbf{s}_{i,j}/2).$$

We give boundary fluxes

$$\mathbf{h}_{-1/2} = \mathbf{h}(\mathbf{w}^*, \mathbf{w}_0 - \mathbf{s}_0/2)$$

at the boundary x_0 and, similarly, at other boundaries. The slope of the last cell $\mathbf{s}_0$ is computed through extrapolation of interior slopes. $\mathbf{w}^* = \mathbf{w}_\infty$ is the free stream state at farfield boundaries. For solid walls we use $\mathbf{w}^* = R(\mathbf{w}_0 - \mathbf{s}_0/2)$ with R the reflexion operator which takes the normal velocity, v to $-v$, and leaves the rest of the variables unchanged. With the boundary fluxes we can update the solution at all points and thus obtain the solution at the new time $\mathbf{w}^{n+1}$ at all cells.

We can directly generalize the one-dimensional multiresolution method by applying it linewise for the i and j fluxes separately. For the i fluxes we apply the one-dimensional algorithm to each grid line $j = j_0$. Similarly for the j fluxes, we apply the one-dimensional algorithm to each grid line $i = i_0$. We thus define the grid levels as

$$G_k = \{(x(\xi, \eta), y(\xi, \eta)) \,|\, (\xi, \eta) \in [0 \dots n_i/2^k] \times [0 \dots n_j]\},$$
$$k = 0, 1, \dots, L,$$

for the i-flux computation, and

$$G_k = \{(x(\xi, \eta), y(\xi, \eta)) \,|\, (\xi, \eta) \in [0 \dots n_i] \times [0 \dots n_j/2^k]\},$$
$$k = 0, 1, \dots, L,$$

for the j-flux computation. The algorithm becomes identical to the one-dimensional algorithm, except for an additional outer loop over the j grid lines (or over i lines for the j direction sweep). To compute the multiresolution representation, we take the cell area into account by using (4.1) instead of (2.3b),

$$A^k_{i,j} u^k_{i,j} = A^{k-1}_{2i+1,j} u^{k-1}_{2i+1,j} + A^{k-1}_{2i,j} u^{k-1}_{2i,j}, \tag{4.1}$$

where $A^k_{i,j}$ is the area of cell (i, j) in G_k. The function $A^k_{i,j} u^k_{i,j}$ is interpolated one-dimensionally on G_k to obtain the interpolant $I(x, A^k_{i,j} u^k_{i,j})$.

Implemented in this way, it is necessary to redefine the grid hierarchy in each time step; i.e., we switch between i and j each step. It would be possible to set up the grid hierarchies once and keep them in memory, but the largest computational cost in the multiresolution algorithm is the encoding procedure, which has to be done even if the grids are precomputed. Thus the recomputing of the grids does not give a significant contribution to the CPU time. The multiresolution representation $(d^1, d^2, \dots, d^L, u^L)$ has to be computed separately in the i and j directions. We thus have to compute two multiresolution representations for each time step.

Alternatively, it is possible to do a pure two-dimensional multiresolution representation of the solution, which can be used for both the i and j directions. We base this algorithm on the grid levels

$$G_k = \{(x(\xi, \eta), y(\xi, \eta)) \,|\, (\xi, \eta) \in [0 \dots n_i/2^k] \times [0 \dots n_j/2^k]\},$$
$$k = 0, 1, \dots, L.$$

The flux calculations are still done one-dimensionally; i.e., $h_{i+1/2,j}$ is computed for all j, but with a multiresolution in the i-direction. The numerical scheme based on this purely two-dimensional grid hierarchy turned out, in our implementation, to be less efficient than doing two one-dimensional sweeps. The reason is that the computation of the primitive function is considerably more complicated in two space dimensions.

We evaluate the i direction fluxes, $h_{i+1/2,j}$, as described in Algorithm 4.1, which is very similar to the one-dimensional algorithm. The j direction fluxes are computed similarly. As usual we denote the grid levels, G_0, ... G_L, with G_0 the finest grid.

We are solving a system of partial differential equations, but this does not lead to any additional difficulties. The interpolation and multiresolution decomposition are straightforward to do componentwise. The truncation algorithm is done scalarly, where in statements such as $|d^k_j| > \varepsilon_k$, $|d^k_j|$ is interpreted as the L^1 norm of the four components of $|d^k_j|$. The flag vector element m^k_j is thus a scalar.

Note that the flag m^k_j is only used on the levels $k = 1, \dots, L$. On the finest level G_0, it is possible to define an additional flag vector m_j^0, which can be used to further improve the execution time [2]. In [2] this is done by defining

$$m_{2j-1}^0 = 1, \quad m_{2j}^0 = 1, \quad m_{2j+1}^0 = 1,$$

if $|d_j^1| > \varepsilon_{\text{osc}}$, with ε_{osc} a tolerance level for the smallest oscillations allowed. Otherwise m_j^0 is set equal to zero. The numerical flux function is then defined as

$$h_{j+1/2} = \begin{cases} h_{j+1/2}^{\text{TVD}} & \text{if } m_j^0 = 1 \\ h_{j+1/2}^{c} & \text{if } m_j^0 = 0, \end{cases} \tag{4.2}$$

where $h_{j+1/2}^c$ is an inexpensive centered difference flux, and $h_{j+1/2}^{\text{TVD}}$ is the computationally expensive flux of a TVD method. Thus, this means that we use centered differences in the smooth part of the solution.

ALGORITHM 4.1.

```
1. Compute all boundary fluxes.
2. for $j := 0$ to $n_j - 1$ do
       for $i := 1$ to $n_i/2^L$-1 do
           Compute the coarse grid fluxes $h_{i2^L-1/2,j}$
       endfor
   endfor
3. for $k := L - 1$ to 1 do
       for $j := 0$ to $n_j - 1$ do
           for $i := 1$ to $n_i/2^{k+1} - 1$
               if $m_{i,j}^{k+1}$=0 then
                   Interpolate to obtain the flux
$h_{2j+1/2}^k = h_{2^k(2i+1)-1/2,j}$.
               else
                   Compute the flux $h_{2j+1/2}^k$.
               endif
           endfor
       endfor
   endfor
4. Update the solution.
```

5. NUMBER OF ARITHMETIC OPERATIONS

Let us compare the cost in number of arithmetic operations. Let c_s be the cost per cell to set up the multiresolution scheme, i.e., computation of d^k and truncation m^k. Let c_f be the cost of computing one numerical flux function, and let c_i be the cost of interpolating one numerical flux function. Let us assume that p is the fraction of interpolated fluxes. The cost of using the multiresolution algorithm is

$$C_{mr} = c_s + pc_i + (1 - p)c_f.$$

This should be compared with the cost of the usual algorithm

$$C = c_f.$$

Thus it is necessary to have

$$C_{mr} \leq C \Rightarrow p \geq \frac{c_s}{c_f - c_i}$$

in order to gain from using multiresolution. In our implementation we have $c_s \approx 53$, $c_i \approx 20$, $c_f \approx 140$ for Roe's method, in two dimensions on a curvilinear grid. We obtain the condition

$$p \geq 0.44.$$

Thus in order to gain execution time more than 44% of the fluxes must be interpolated. In the more favorable case of the ENO scheme, the fourth-order ENO/Roe method [4] can be implemented using around 520 operations to compute one numerical flux for two-dimensional Euler equations on a curvilinear grid. This would give us

$$p \geq 0.1;$$

i.e., more than 10% of the fluxes must be interpolated. We conclude that the multiresolution method will probably increase the efficiency for the ENO method.

In reality, we have observed that even a larger precentage than mentioned above is required. The explanation of this is that by reordering the flux computation as is done in the hierarchical computation of fluxes in the multiresolution method, we lose efficiency on many computer architectures. This is certainly true on vector machines, but the reordering seems to have a negative effect also for RISC-type architectures. Furthermore, the multiresolution order of computing fluxes leads to more paging to the disk on machines using virtual memory. Introducing the factor of loss from reordering, α, i.e., if $\alpha = 2$ the algorithm is twice as slow when reordered. Taking this factor into account we get the cost of the MR algorithm

$$C_{mr} = c_s + pc_i + (1 - p)\alpha c_f,$$

which leads to the criterium for p,

$$p \geq \frac{c_s + (\alpha - 1)c_f}{\alpha c_f - c_i}.$$

In the most favorable case when $c_f \gg c_i, c_s$, we obtain the criterion

$$p \geq 1 - 1/\alpha$$

for having a gain in execution time using the multiresolution method.

From the previous discussion, it is clear that c_s and α are the

TABLE I

Performance in CPU Seconds for Roe's Method Computed in Various Ways

	65×33	129×65	257×129	513×257
Straight flux computation	0.072	0.28	1.1	5.0
MR order flux computation	0.076	0.28	1.2	5.0
MR order fluxes + MR overhead	0.13	0.50	2.0	9.3
MR order fluxes + ρ-MR overhead	0.10	0.40	1.8	8.6

critical parameters. Table I presents results from an experiment which was set up in order to study the relative importance of these two quantities. Table I shows the CPU time in seconds for taking one time step with Roe's method for the two-dimensional Euler equations on a Sun SPARC station 10/40 workstation, using the FORTRAN programming language, compiled with the `-fast` compiler option.

The first row shows the standard flux computation, the second row shows exactly the same computation, but with the flux computed in the hierarchical way of the multiresolution scheme. The number of levels were increased from two on the coarsest grid to five levels on the finest. In the third row we show the time, when the actual multiresolution decomposition is done, but with all fluxes computed by Roe's method. We see clearly that on this computer, the overhead from reordering the flux computation is not large, but that the cost of doing the multiresolution decomposition is significant. This overhead does not depend on the method, so that with a more expensive high order ENO flux computation it would be less significant.

The multiresolution decomposition is only used to flag the nonsmooth part of the solution. It is therefore reasonable to compute the coefficients $d^k_{i,j}$ for only one component of the equations. In the last row we present performance figures for the method where multiresolution is made only in the density. A small gain is obtained from this ρ-multiresolution method.

From Table I we can estimate the practical value of c_s/c_f by taking the quotient between the costs for the overhead and the straight flux computation. We obtain the value 78%, which means that, based on the value $c_f = 140$, we have $c_s \approx 110$. This gives the considerably more negative estimate $p \geq 0.92$ for the first-order Roe's method; i.e., 92% of the fluxes must be interpolated in order to have any gain in CPU time.

We present in Table II a decomposition of the multiresolution overhead for the grid with 257×129 points. Encoding $\mathtt{I}$ is the cost to compute the coefficients d^k_{ij} for the I-direction flux computation. Truncation is the cost to compute the flag vector $m^k_{i,j}$ from the coefficients $d^k_{i,j}$. The table shows the total cost for all multiresolution levels. It is clearly the encoding which dominates the cost. Note that the cost for computing the truncation is significant, although there are hardly any arithmetic operations in this algorithm.

In another test, where we used Roe's method in an actual multiresolution computation on an IBM RS6000 workstation, we obtained by numerical experiments that $p \geq 62\%$ in order to have a gain in CPU time. In the next section we will go more into detail about the actual performance of the method.

6. NUMERICAL RESULTS

In order to monitor the performance of the multiresolution method, we here present some computations that are standard in computational fluid dynamics (CFD). We used a Sun SPARCstation 10/40 workstation and we implemented the algorithms in Fortran 77, using the compiler option `-fast`. We use a second-order TVD difference method in finite volume formulation and, thus, we use cell average interpolation formulas. One of the purposes of this work was to evaluate the multiresolution method under standard CFD conditions. This is the reason why we chose the finite volume method, since it is a commonly used technique in CFD, and this is also the reason why we chose a steady state computation.

The first test is the supersonic flow past a disk. The freestream Mach number is 3. The iso-Mach contours of a typical steady state solution are shown in Fig. 6.1.

The problem was solved on grids of various sizes. The multiresolution scheme becomes more efficient for grids with a large number of grid points, since a relatively larger number of fluxes can be interpolated. In Table III we show results using a grid of 257×129 points. This turned out to be the smallest number of points required to obtain any gain in CPU time. We give the percentage of flux evaluations in the computation for different numbers of grid points and different choices of the truncation parameter ε in Algorithm 3.1. We use $p = 1$, where p is the other parameter in Algorithm 3.1. The number of multiresolution levels were five. Multiresolution was done in the i-direction for the i-direction fluxes and in the j-direction for the j-direction fluxes, as described in Section 4. The fourth-order formulas (2.4) were used in the flux interpolation.

We show performance results in Table III. There is no

TABLE II

Closer Investigation of the MR Overhead in CPU Seconds

	MR	ρMR
Total overhead	0.8	0.6
Encode $\mathtt{I}$	0.28	0.18
Encode $\mathtt{J}$	0.29	0.19
Truncate $\mathtt{I}$	0.10	0.06
Truncate $\mathtt{J}$	0.10	0.07
Remains	0.03	0.10

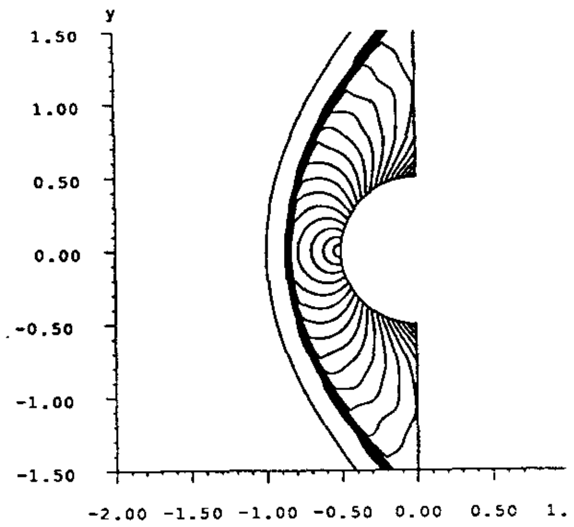

FIG. 6.1. Iso-Mach contours.

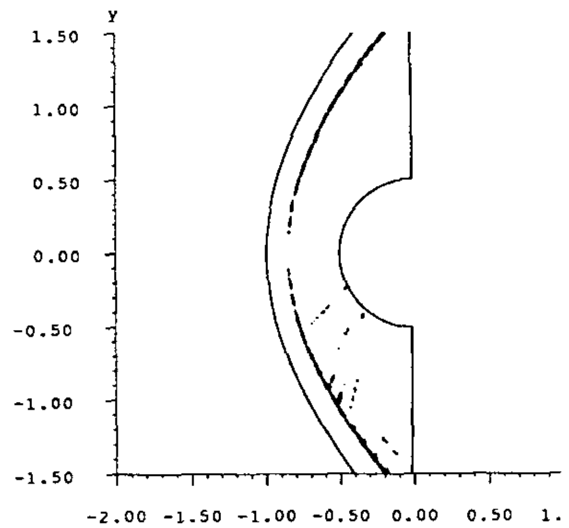

FIG. 6.2. i-direction for d^1 density multiresolution coefficient.

speedup in terms of CPU time. The first-order Roe's method is too inexpensive to make up for the multiresolution overhead as discussed in the previous section. This test problem serves to evaluate other aspects of the method, such as the quality of the computed solution. For the case $\varepsilon = 0$, we computed the fluxes in a straight way, not doing the hierarchical multiresolution ordering. Thus the overhead for multiresolution interpolation is not included in this case.

The distance from the solution with $\varepsilon = 0$ is given; the difference is small. However, when we look at the residual in this steady state computation, we see that the multiresolution method inhibits steady state convergence. The residual entry in Table III gives the level on which the convergence curve flattens out; it is measured in the L^∞ norm. The last entry is the L^2 norm of the error in entropy on the body, i.e., the difference between the exact entropy, which is known analytically, and the computed solution. This is the important quantity which we typically would like to capture in a CFD computation. The entropy on the body is not much affected by the multiresolution manipulations. The reason is that the boundary fluxes are evaluated and not interpolated and that the errors coming from poor steady state convergence are located near the shock wave and not on the body.

In Fig. 6.2 we show the multiresolution coefficient $d^1_{i,j}$ at steady state on the second finest grid, G_1, for the density component in the i-direction multiresolution. We show the corresponding j-direction quantity in Fig. 6.3. For all grids used in this section, the j-direction is the radial direction, and the i-direction is tangential to the boundary of the disk. It is clear that d^1 is large in the nonsmooth part of the solution. However, the shock is almost aligned with the i-direction, and consequently in the i-direction d^1 is only a little larger near the shock. That is the reason why contour lines can be seen outside the shock in Fig. 6.2.

In this computation we have assured that the grid transformation $(x(\xi, \eta), y(\xi, \eta))$ is smooth. However, when this is not the case, d^1 will be large in the parts of the domain where the grid is not smooth. Finally, the quantity $m^1_{i,j}$ is given in Fig. 6.4 and Fig. 6.5.

In Figs. 6.4 and 6.5 we see that a reasonable part of the solution is flagged by the multiresolution method. It is mostly the points around the bow shock which have $m^1_{i,j} = 1$.

Next we consider a less trivial test case, the interaction of two shocks. This is a problem where it could be motivated to use a large number of grid points. We start from the solution of the previous test problem, supersonic flow past a disk, and

TABLE III

Performance for Supersonic Disk Flow, Roe's Method with Multiresolution

ε	Seconds/step	% Roe fluxes	$\|u^\varepsilon - u^0\|$	Residual	Error on body
0	1.10	100	0	0.001	0.025
0.05	1.62	42	0.0047	2.5	0.026
0.1	1.57	35	0.013	4	0.026
0.5	1.50	25	0.058	20	0.025

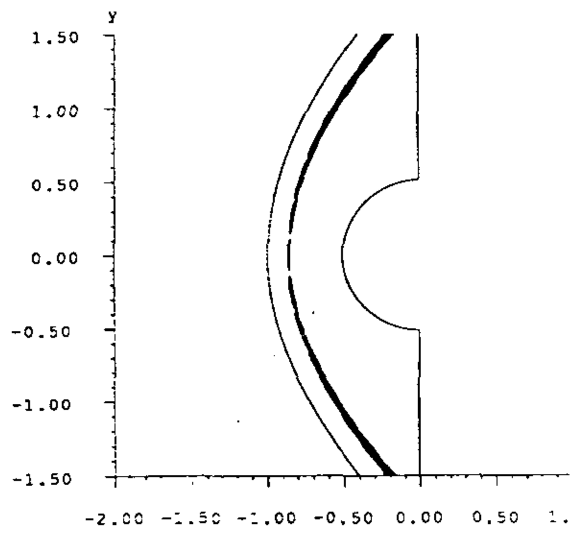

FIG. 6.3. j-direction for d^1 density multiresolution coefficient.

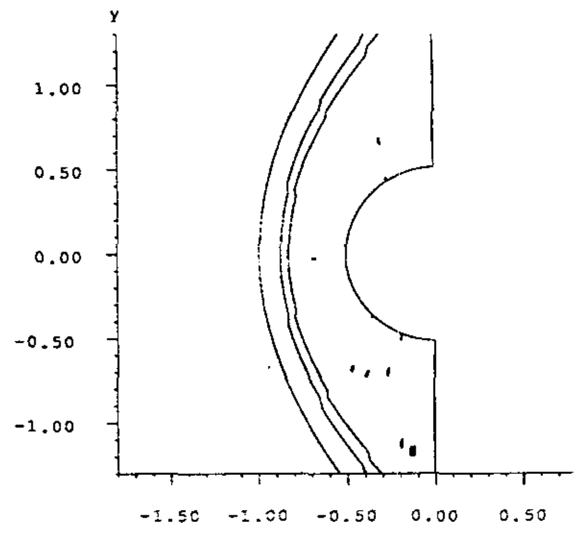

FIG. 6.5. j-direction for one $m^1_{i,j}$ contour line at the value 0.5.

we let a second shock impinge on the bow shock. This set up is described in [5].

We used the second-order accurate Roe's method for this computation. The piecewise linear reconstruction is done in the characteristic variables, and we use the minmod limiter. This problem is very sensitive to the limiting procedure. We have found that limiting in the variables (ρ, u, v, p) leads to substantial oscillations in the density.

We show the iso-Mach contours of a well-resolved solution in Fig. 6.6 on a grid having 257 $\times$ 129 points. In Fig. 6.7 we show the result of a computation using multiresolution, with parameter values in the truncation Algorithm 3.1 $\varepsilon = 0.1, p = 2$. The flag vector $m^1_{i,j}$ for the j-direction decomposition is displayed in Fig. 6.8, indicating the part of the domain where the flux is computed directly. Like in the previous case, we see that the multiresolution scheme flags the non smooth part of the computational domain correctly.

In Fig. 6.9 we have increased ε to 3, which is near the largest possible value, because the computation was unstable and blew up with $\varepsilon = 4$ for the CFL number we used ($\frac{1}{2}$). We see that small oscillations start to appear. In Fig. 6.10 the i-direction flag vector $m^1_{i,j}$ shows that only a small part is flagged. The grid is almost aligned with the bow shock near the lower part of the plot and thus this part is not flagged, not even at the shock. Instead, in the radial j direction the bow shock is clearly flagged; however, for this large value of ε we start losing

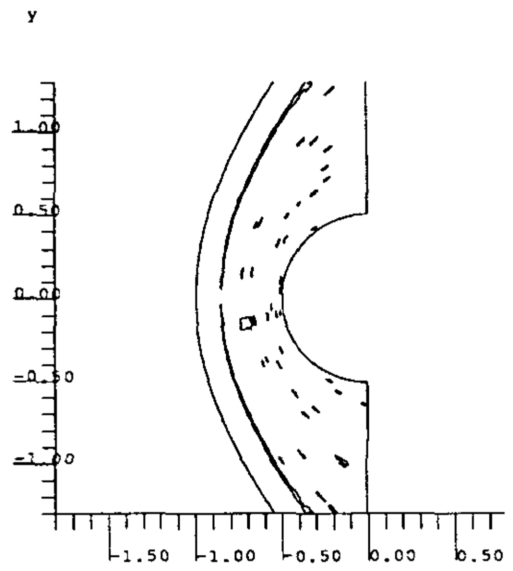

FIG. 6.4. i-direction for one $m^1_{i,j}$ contour line at the value 0.5.

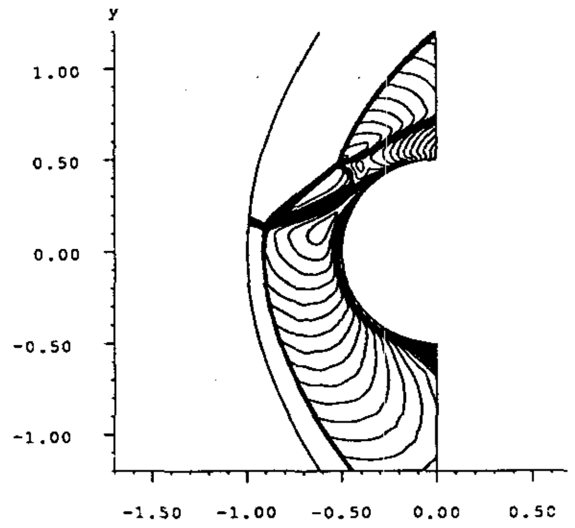

FIG. 6.6. Iso-Mach contours, shock–shock interaction, 257 $\times$ 129 points.

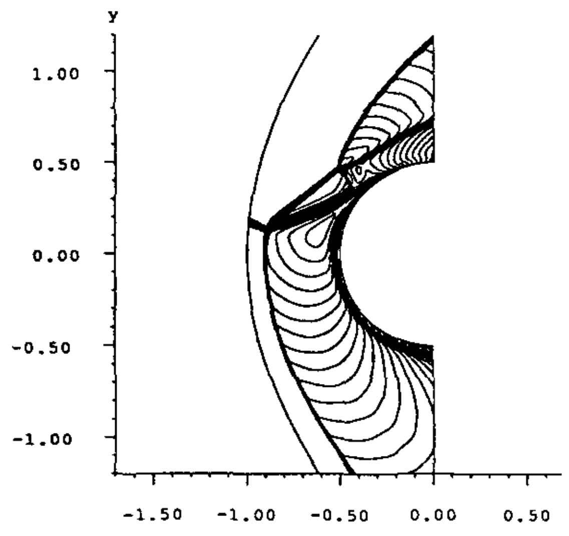

FIG. 6.7. Iso-Mach contours, shock–shock interaction, $\varepsilon = 0.1, p = 2$.

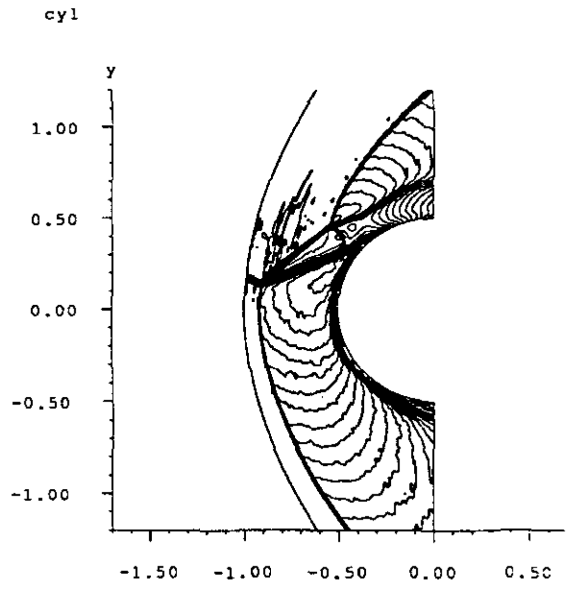

FIG. 6.9. Iso-Mach contours, $\varepsilon = 3, p = 2$.

information from the region of shock interaction. Table IV shows some performance data for this computation.

The first three entries in Table IV show the overhead from reordering the flux computation in the multiresolution way and the overhead from doing the multiresolution decomposition. We see that the overhead for reordering the flux computation is large. In addition to the overhead described for the first-order accurate method in Section 5, we here have an additional source of overhead. The slope in the piecewise linear reconstruction, s_j, in formula

$$h_{j+1/2} = h(u_{j+1} - s_{j+1}/2, u_j + s_j/2)$$

is computed twice, once for the $h_{j+1/2}$ flux and once for the $h_{j-1/2}$ flux. In the straight order of computation, s_j needs only to be computed once for each cell. This is not possible in the multiresolution scheme, since we do not know in advance in which cells s_j will be required. It is possible to precompute the slopes in the multiresolution order, but this will involve additional overhead which does not make it worthwhile. In many simpler problems, limiting can be done in the variables (ρ, u, v, p) and the cost of recomputing s_j would not influence the performance as significantly as for this problem.

Further down in Table IV we show the result from using the multiresolution method with three different values of ε. We

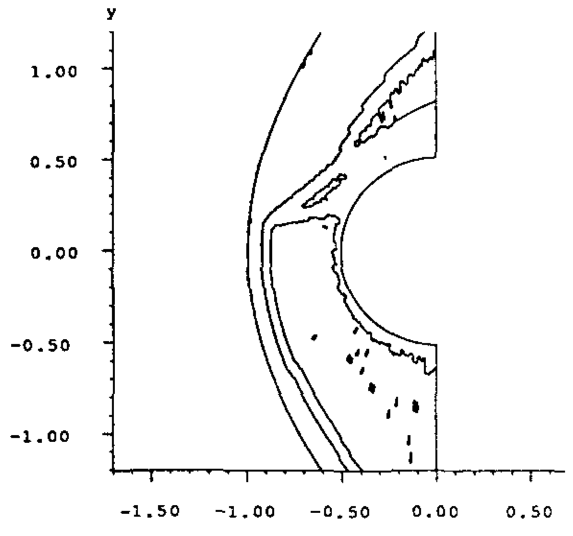

FIG. 6.8. $m^1_{i,j}$, shock–shock interaction, $\varepsilon = 0.1, p = 2$.

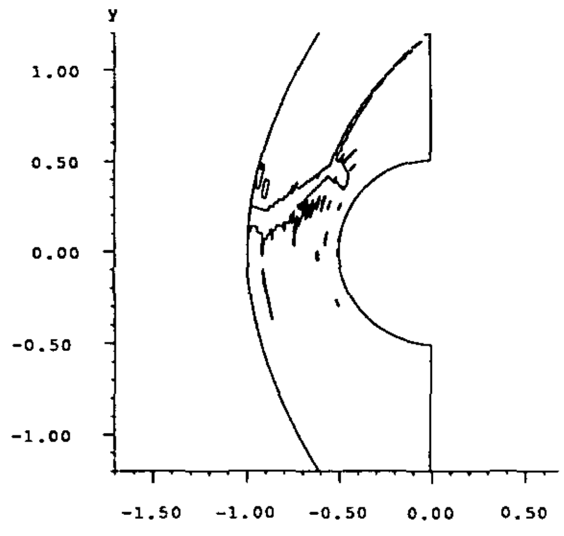

FIG. 6.10. i direction $m^1_{i,j}$.

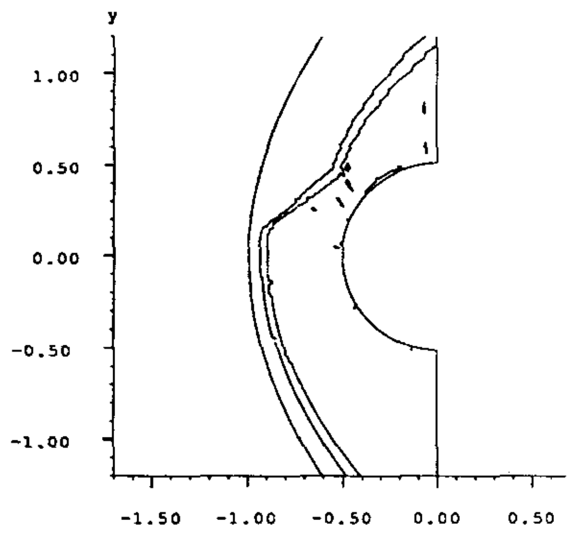

FIG. 6.11. j direction $m_{i,j}^{1}$.

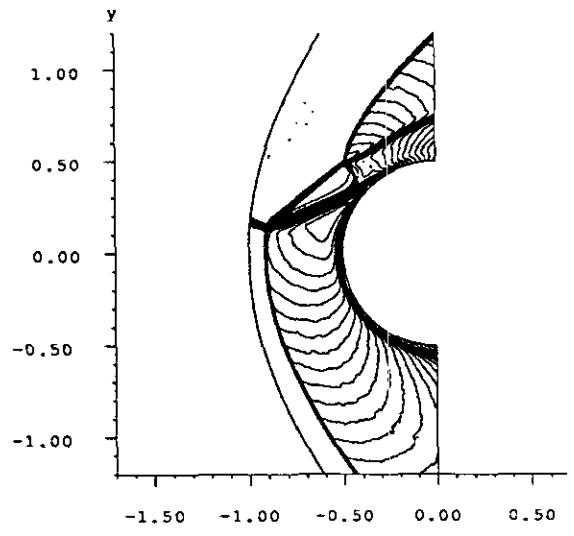

FIG. 6.12. Adaptive fluxes, $\varepsilon = 0.5$.

obtain a significant gain in CPU time. This is contrary to the first-order method, where no gain was obtained. We thus have an example of the obvious fact that the more expensive the flux, the more advantageous is it to use the multiresolution method.

Here we computed the second-order TVD flux everywhere when $m_{i,j}^{k} = 1$. It is described in [2], and in Section 4, how to further gain in CPU time, by changing to a centered difference flux when $d_{i,j}^{1}$ is below some tolerance level of oscillations, ε_{osc}.

We finish by introducing this adaptive flux evaluation (4.2) into the computer code. We follow [2] in taking $\varepsilon_{\text{osc}} = \varepsilon_{1}$. In the previous computations forward Euler was used as time discretization. The centered difference approximation is not stable when forward Euler is used in time. We therefore use a three-stage Runge–Kutta method in the time for the total computation. The figure for the CPU time/step given in Table IV is for this computation CPU time per Runge–Kutta stage.

In Table IV we show results for $\varepsilon = 0.5$ and $\varepsilon = 0.1$. When centered difference fluxes are mixed with TVD fluxes in this way, it turns out that a smaller value of ε is required in order to keep the computation stable. Nevertheless, the results from using $m_{i,j}^{0}$ are considerably more encouraging in terms of CPU time. A reduction of approximately a factor 0.8 in CPU time is obtained.

We see in Fig. 6.12 and Fig. 6.13 that, like in the previous computations, oscillations develop when ε is increased. However, the solution with $\varepsilon = 0.1$ looks almost as good as, if not better than the TVD solution in Fig. 6.6.

A big advantage with the adaptive multiresolution method is that it can avoid the degeneracy to first order at smooth extrema. The degeneracy at extrema is a problem which plagues all second-order accurate TVD methods. The centered difference method does not have the problem and is more accurate

TABLE IV

Performance for Shock–Shock Interaction, Second-Order Roe with Multiresolution

Method	ε	Seconds/step	% Roe	% centered
Natural order	0	2.40	100	0
MR order	0	2.58	100	0
MR order + overhead	0	3.53	100	0
MR	0.1	2.70	61	0
MR	1	2.13	34	0
MR	3	2.04	30	0
Adaptive MR	0.1	2.00	15	74
Adaptive MR	0.5	1.85	9	73
Adaptive MR, 2 levels	0.1	1.60	15	74

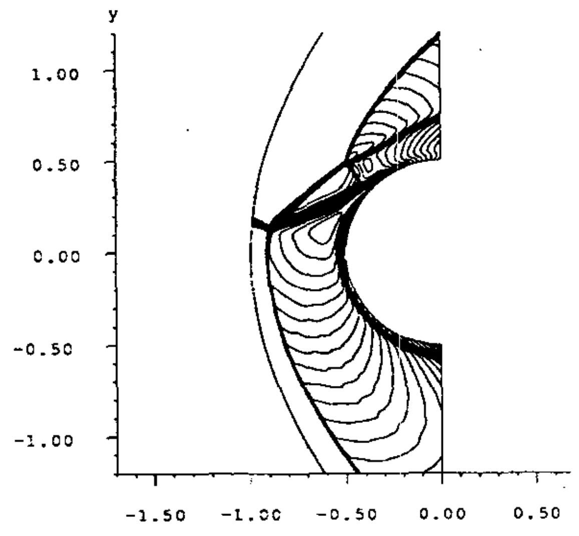

FIG. 6.13. Adaptive fluxes, $\varepsilon = 0.1$.

in the smooth part of the solution than a TVD method. Thus the adpative multiresolution method gives both a decrease in computational time and an increase in accuracy. It is a method which we strongly recommend.

From the numerical experiments we conclude that the biggest advantage comes when centered fluxes are mixed adaptively with TVD fluxes. Actually, it is possible to improve the performance by abandoning the multilevel approach for a two-level algorithm. In the last entry of Table IV we show the time for a computation based on only two multiresolution levels, whereas all the other computations were made with five levels. The overhead becomes smaller, and the number of interpolated fluxes does not change significantly.

The cost of interpolating a numerical flux function is comparable with the cost of evaluating a centered difference flux. We suggest that for optimal performance, we only compute one level of multiresolution coefficients, $d^1_{i,j}$, and use these to switch between centered and TVD fluxes, with no flux interpolation.

7. CONCLUSIONS

The multiresolution method, presented in [1] gives a considerable gain in efficiency, under the following conditions:

1. Large number of grid points.
2. Computationally expensive original method.
3. Nonvector computer.

If these conditions are not satisfied it is not as clear whether it will pay to use the method. More investigations are probably necessary. We presented formulas in Section 5 which can give some guidance to the expected performance.

The method is straightforward to generalize to two dimensions on curvilinear grids. The method can inhibit steady state convergence to machine precision. However, we did not observe that this caused any loss of accuracy; the residual was on the level of the truncation error.

We conclude by numerical experiments that multiresolution does not pay for a first-order TVD method, but gives improvement in execution time for a second-order TVD method. In order to achieve the largest gain, it is necessary to adaptively mix centered and TVD fluxes as described in [2] and to keep the number of multiresolution levels small. For the best case we reduced the CPU time by a factor 0.56. The gain was not as substantial as in the one-dimensional tests in [1]. Nevertheless, the adaptive multiresolution method, where centered and TVD fluxes are mixed, gives sufficient reduction in CPU time that it would be worthwhile implementing it into real production CFD codes.

We have here only implemented the multiresolution scheme for finite volume TVD methods of order one and two. The method is better suited for the more expensive ENO schemes. The reason we did not use the ENO method here was, first, that we wanted to investigate how well the method performs on a state of the art CFD problem and, second, to keep the work manageable. A natural continuation of this work would be to implement the multiresolution method for the flux interpolated ENO scheme in [4].

ACKNOWLEDGMENTS

Part of this work was done while I visited Centre Européen Recherche et de Formation Avancée en Calcul Scientifique (CERFACS) in Toulouse, France.

REFERENCES

1. A. Harten, UCLA, CAM Report 93-03 (unpublished).
2. A. Harten, *J. Comput. Phys.*, submitted.
3. P. L. Roe, *J. Comput. Phys.* **43,** 357 (1981).
4. C-W. Shu and S. Osher, *J. Comput. Phys.* **83,** 32 (1989).
5. H. C. Yee, NASA TM-101088, February 1989 (unpublished).