REPORT 86-14

Vectorization of a multigrid method
for the compressible
Navier-Stokes equations

by

G. Shaw and P. Wesseling

ISSN 0920-2358

Reports of the Department of Mathematics and Informatics no. 86-14

Delft 1986
This work was partially supported by the Netherlands organization for the Advancement of Pure Research.
Abstract.

An efficient multigrid method for the solution of the compressible Navier-Stokes (CNS) equations was described by Shaw and Wesseling (1986). This algorithm was designed such that the resulting computer program was readily amenable to vectorization.

This report describes the subsequent process of vectorization giving comparisons of scalar and vector timings on a CRAY-1S.

Of particular interest is the vectorization of the Van Leer flux splitting method, the implementation of which constitutes one of the most time consuming elements in the algorithm.

The program is found to run approximately three times faster in vector mode than in scalar mode.

1. Introduction.

This report illustrates that multigrid methods of the type described by Shaw and Wesseling (1986) are readily vectorizable. Such algorithms are already highly efficient in scalar mode. Thus very fast methods may be developed for the solution of the CNS equations. Another type of vectorized multigrid algorithm is described by Chima and Johnson (1985). This is based on the Lax-Wendroff method of MacCormack (1969). Convergence to the steady state is accelerated by the use of coarse grids as described by Niel (1982).

Since the original algorithm was designed with a view to its subsequent vectorization much of that process is relatively straightforward. Fundamental features of the algorithm which were chosen for reasons related to vectorization, are described in section 2 of this report.

Section 3 describes the purpose of the time consuming elements of the computer-program, on which vectorization efforts have been concentrated.

Sections 4 and 5 discuss the vectorization of these elements. Section 4 deals with the Van Leer flux splitting, which is the most challenging aspect.

Section 6 gives a comparison of time spent in scalar and vector codes for a series of viscous and inviscid cascade flow problems. It is shown that a speed-up factor of three is obtainable.
2. Algorithmic Features Relevant To Vectorization.

2.1 Smoothing.

Classical multigrid methods require the definition of a procedure by which the error of an approximate solution may be smoothed. For this purpose many iterative methods are available. Their ability to smooth the error is dependent on the particular problem under consideration. In Shaw and Wesseling (1986) it is found that a non-linear generalization of the Point Gauss-Seidel method, known as collective Newton Gauss-Seidel (CNGS), has satisfactory smoothing properties. In terms of vectorization potential this method is not ideal due to its essentially recursive structure. However, the method is vectorizable along lines of cells whose associated unknowns are not coupled by the discrete equations.

The finite volume grid is chosen to be topologically equivalent to a rectangular grid with $N_x$ cells in the $x$-direction and $N_y$ cells in the $y$ direction. The inviscid Euler equations may be discretized on a compact five-point stencil. In this case Gauss-Seidel with red-black ordering vectorizes, with vectors of length $\frac{1}{2}N_xN_y$. Hemker and Spekrijse (1985) found that this scheme is inferior to CNGS for a discretization similar to ours. Furthermore, for the full CNS equations a seven or nine point stencil is required and red-black no longer vectorizes.

If the equations are discretized on a compact seven point stencil CNGS vectorizes along diagonal lines of the grid, with vectors ranging in length from one to $\min(N_x, N_y)$. Such a diagonal is illustrated in figure 2.1 This method is suitable for the CRAY series of computers which may achieve a significant speed-up for short vectors and do not require unit subscript stride. For machines without these features it may be advantageous to consider non-recursive smoothing methods such as underrelaxed Jacobi.

Choosing to use CNGS with diagonal ordering we therefore discretize viscous terms on a seven point stencil as described in Shaw and Wesseling (1986)
2.2 Data Structure.

At each cell of each grid four unknowns and four right hand sides are defined. All these values are stored in a single one-dimensional array. This array is divided into a section for each grid. Each section is further divided into eight regions for the eight variables per cell. In these subregions storage is allocated according to a lexicographic ordering of the cells on the relevant grid. This situation is depicted in figure 2.2.

The data structure described above leads to constant stride array access for all processes in the multigrid algorithm, whilst minimizing paging conflicts.
2.3 Update of Unknowns.

CNS proceeds in three stages:

i) Calculate fluxes and hence the non-linear 4-vector to be set to zero at each cell in the diagonal.

ii) For each cell in the diagonal calculate the 4x4 Jacobian of the relevant 4-vector.

iii) Update the four unknowns simultaneously, for each cell in the diagonal, by performing a single Newton iteration.
These processes are discussed in detail in sections 4 and 5. iii) requires the stable solution of a set of \(4 \times 4\) systems of equations. A solution method requiring interchanges to maintain stability will not readily vectorize, since systems corresponding to different cells in the diagonal will not generally require the same interchange pattern. We therefore choose to use the QR method (see Strang (1980)), which is stable without the need for interchanges.

3 Time Consuming Program Elements.

By means of the flow trace compilation option it was found that typically 95% of the total CPU time is expended in four of the subroutines. Vectorization efforts have therefore been concentrated on these routines, although many others vectorize directly on submission to the CPT 1.13 compiler.

The purpose of the time consuming subroutines is outlined below. Aspects of their vectorization are discussed in the following two sections.

3.1 Subroutine PCTUVL.

The computations variables used throughout the algorithm are:

- \(p\) - density
- \(pu\) - density \(\times\) u - velocity component
- \(pv\) - density \(\times\) v - velocity component
- \(e\) - total energy / unit volume.

During the computation of viscous and inviscid fluxes we also require:

- \(p\) - static pressure
- \(c\) - speed of sound
- \(t\) - temperature
- \(u\) - u - velocity component
- \(v\) - v - velocity component,

which are algebraically obtainable from the above variables. In order to avoid unnecessary recomputation it is advisable to store these quantities for all
cells in the diagonal and their neighbours prior to flux calculation. The subroutine PCTUVL performs this function.

3.2 Subroutine EULER.

Given an approximation to the state vector for each cell in the diagonal, the subroutine EULER calculates the inviscid flux into each cell. Fluxes are approximated by means of the flux splitting method of van Leer (1982).

The Jacobian of the inviscid flux is also calculated.

3.3 Subroutine VISC.

The routine VISC finds the viscous flux into each cell in the diagonal. It has been found by experiment that for realistic Reynolds numbers the Jacobian of the viscous flux need not be calculated.

3.4 Subroutine QR.

QR solves a 4×4 linear system of equations at each cell in the diagonal, in order to update the corresponding unknowns by one Newton iteration. The square of the dynamic residual norm is incremented.

4 The Van Leer Flux Splitting.

As described in Shaw and Wesseling (1986), the computation of the inviscid flux into a cell requires the calculation of eight 4-vectors of the form:

\[
F^{\text{sup}}(q) \quad M(q) > 1 \\
F^{+}(q) = F^{\text{sub}}(q) \quad |M(q)| < 1 \\
0 \quad M(q) < -1
\]

(4.1)

where \( q \) is a state 4-vector, and \( M(q) \) is a scalar one-dimensional local Mach number. \( F^{\text{sup}} \) and \( F^{\text{sub}} \) are distinct vector functions corresponding to supersonic and subsonic flow respectively. There are two essentially different manners in which to vectorize the computation of such functions along a diagonal of the grid. These are outlined in the following subsections.
4.1 Flux Calculation By Merging.

CRAY FORTRAN is simply standard FORTRAN together with some additional intrinsic functions described in the appendix to the CRAY-1 FORTRAN reference manual (1979). Amongst these additional functions are the conditional vector merge functions, which calculate the results of two different vector operations and merge them to form a single vector result; the elements of which depend on some given condition. A conditional vector merge may be used to calculate the function $F^+$ defined by (4.1) in a vectorizable manner. In the interests of code portability it is preferable however to simulate the merge function by means of standard FORTRAN. This may be done as indicated heuristically by algorithm 4.1. The algorithm avoids conditional branching by calculating both $F^{\text{sup}}$ and $F^{\text{sub}}$ for all cells. If the flow state is such that $M(q) < -1$ for a significant number of cells this is clearly a costly feature of the technique. To avoid such a situation we may employ gather and scatter operations as described in the following subsection.

**Algorithm 4.1 - Merged Calculation Of $F^+$**

Let the grid diagonal comprise $L$ cells $\Omega_i$, $i=1(1)L$ with state vectors $q_i$

do 10 $i = 1, L$

calculate: $\text{mach} = M(q_i)$

$s_{\text{sup}} = 0.5*\text{sign}(1., \text{mach}-1.)+1. = 1$. mach > 1.

0. otherwise

temp = $0.5*\text{sign}(1., -\text{mach}-1.)+1. = 1$. mach < -1.

0 otherwise

$s_{\text{sub}} = 1.-s_{\text{sup}}-\text{temp} = 1$. $|\text{mach}| < 1.$

0. otherwise

calculate $F^{\text{sup}}(q_i), F^{\text{sub}}(q_i)$

$F^+(q_i) = s_{\text{sup}}F^{\text{sup}}(q_i) + s_{\text{sub}}F^{\text{sub}}(q_i)$

10 continue.
4.2 Flux Calculation By Gather and Scatter Techniques.

In this approach the mach numbers $M(q)$ are first computed and analysed to
decide the flow type for each cell in the diagonal. $F^{\text{sup}}$ and $F^{\text{sub}}$ are then
calculated only for the cells which require them. The computation and analysis
of $M(q)$ is vectorizable. Calculations of $F^{\text{sup}}$ and $F^{\text{sub}}$ vectorize provided the
state vectors from cells of each type are gathered into consecutive or
constant stride storage. This gathering is necessary in order to avoid the use
of indirectly addressed arrays, which inhibits vectorization. The computed
fluxes are then scattered into the ordering originally defined by their
related state vectors. Thus $F^+$ is found for all cells in the diagonal without
any unnecessary computation.

Problems in fluid dynamics commonly have solutions exhibiting one flow type
throughout the majority of the region of solution. In calculations of flow
around aerfoils for example, a large percentage of the flow may be subsonic
with just a small supersonic region. For such problems the gather and scatter
technique is attractive. If the flow is predominantly supersonic this is
particularly so since the case $M(q) < -1$ occurs frequently, and the merging
technique leads to two flux evaluations where none is necessary. For these
reasons we adopt the second method for the evaluation of $F^+$. It should be
noted, however, that for predominantly subsonic flow problems the cost of the
gathering and scattering may reduce the benefits of vectorization.

5. Treatment Of Other Routines.

For the remaining routines: PCTUVL, VISC and QR the vectorization process
is straightforward. The techniques used are all standard and described, for
instance, in Gentsch (1984). They include:

1) Reordering of DO loops to obtain the longest innermost.
2) Unrolling short interior DO loops as an alternative to 1).
3) Manipulation of arithmetic to increase chaining of vector operations.
4) Replacing single DO loops with long complicated code blocks by several DO
   loops in order to reduce the number of vectors and scalar temporaries
   referenced in any one loop.

No IF statements occur in the scalar versions of the above routines. No
fictitious recursion is suspected by the compiler. All vector addressing has constant stride.

Because no special intrinsic functions are used in the vectorized subroutines the code is portable.

6 Timings.

The scalar and vector codes were tested on three bicircular arc cascade problems. A sequence of five grids $\Omega_k$, $k = 1(1)5$ were employed. $\Omega_k$ has $3 \times 2^k$ cells in the x-direction and $2^k$ cells in the y-direction. The finest grid $\Omega_5$ therefore has $96 \times 32$ cells and its longest diagonal comprises 32 cells.

The first problem is the inviscid GAMM workshop problem as defined by Rizzi and Vivian (1981). The flow is transonic. The second models an inviscid supersonic diffuser, with inlet Mach number 1.4. Finally, the viscous test is for a transonic flow with inlet Mach 0.8. Graphical results for these tests appear in Shaw and Wesseling (1986).

Table 6.1 shows the total time in seconds for the CRAY-15 in both scalar and vector modes. The first figure is the scalar time, the second the vector time, and the figure in parentheses the speed-up factor defined as scalar time/vector time. The overhead has been subtracted.
Table 6.1 Total Times

<table>
<thead>
<tr>
<th>Finest Grid</th>
<th>Problem 1</th>
<th>Problem 2</th>
<th>Problem 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Omega_2$</td>
<td>-</td>
<td>.511</td>
<td>.888</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>.392</td>
<td>.650</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>(1.31)</td>
<td>(1.37)</td>
</tr>
<tr>
<td>$\Omega_3$</td>
<td>3.457</td>
<td>2.156</td>
<td>3.861</td>
</tr>
<tr>
<td></td>
<td>1.749</td>
<td>1.147</td>
<td>1.909</td>
</tr>
<tr>
<td></td>
<td>(1.98)</td>
<td>(1.88)</td>
<td>(2.02)</td>
</tr>
<tr>
<td>$\Omega_4$</td>
<td>14.077</td>
<td>8.786</td>
<td>15.850</td>
</tr>
<tr>
<td></td>
<td>5.297</td>
<td>3.437</td>
<td>5.821</td>
</tr>
<tr>
<td></td>
<td>(2.66)</td>
<td>(2.56)</td>
<td>(2.72)</td>
</tr>
<tr>
<td>$\Omega_5$</td>
<td>56.209</td>
<td>35.374</td>
<td>63.705</td>
</tr>
<tr>
<td></td>
<td>17.470</td>
<td>11.248</td>
<td>18.990</td>
</tr>
<tr>
<td></td>
<td>(3.22)</td>
<td>(3.15)</td>
<td>(3.35)</td>
</tr>
</tbody>
</table>

The speed-up factor on $\Omega_5$ is more than three in all cases. This compares favourably with the results of Chima and Johnson (1985), whose explicit MacCormack scheme yields a speed-up factor of from 2.15 to 2.36.

It is important to consider the effect on the individual costly subroutines. Tables 6.2-5 give the timings for PCTUWL, EULER, VISC and QR respectively. The format of these tables is as for 6.1.
Table 6.2 Time Spent In Subroutine PCTUVL

<table>
<thead>
<tr>
<th>Finest Grid</th>
<th>Problem 1</th>
<th>Problem 2</th>
<th>Problem 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Omega_2$</td>
<td>-</td>
<td>.043</td>
<td>.063</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>.035</td>
<td>.049</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>(1.23)</td>
<td>(1.29)</td>
</tr>
<tr>
<td>$\Omega_3$</td>
<td>.228</td>
<td>.155</td>
<td>.230</td>
</tr>
<tr>
<td></td>
<td>.122</td>
<td>.089</td>
<td>.123</td>
</tr>
<tr>
<td></td>
<td>(1.88)</td>
<td>(1.74)</td>
<td>(1.87)</td>
</tr>
<tr>
<td>$\Omega_4$</td>
<td>.834</td>
<td>.561</td>
<td>.843</td>
</tr>
<tr>
<td></td>
<td>.289</td>
<td>.211</td>
<td>.294</td>
</tr>
<tr>
<td></td>
<td>(2.89)</td>
<td>(2.66)</td>
<td>(2.86)</td>
</tr>
<tr>
<td>$\Omega_5$</td>
<td>3.131</td>
<td>2.096</td>
<td>3.167</td>
</tr>
<tr>
<td></td>
<td>.702</td>
<td>.510</td>
<td>.716</td>
</tr>
<tr>
<td></td>
<td>(4.46)</td>
<td>(4.11)</td>
<td>(4.43)</td>
</tr>
</tbody>
</table>

The first point to be noted from these results is that the speed-up factors for the subroutine EULER are not strongly dependent on flow characteristics. This is naturally also true for the other routines since they perform the same functions for any flow type. Thus we may concentrate our attention on any of the three test problems. We consider problem 3 since this is the case in which VISC is called.

The speed-up factors for QR, which constitutes approximately 30% of the total scalar time, are quite satisfactory. The total time speed-up factor is
Table 6.3 Time Spent In Subroutine EULER

<table>
<thead>
<tr>
<th>Finest Grid</th>
<th>Problem 1</th>
<th>Problem 2</th>
<th>Problem 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Omega_2$</td>
<td>-</td>
<td>.395</td>
<td>.627</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>.401</td>
<td>.605</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(.985)</td>
<td>(1.04)</td>
</tr>
<tr>
<td>$\Omega_3$</td>
<td>2.335</td>
<td>1.458</td>
<td>2.356</td>
</tr>
<tr>
<td></td>
<td>1.743</td>
<td>1.486</td>
<td>1.735</td>
</tr>
<tr>
<td></td>
<td>(1.34)</td>
<td>(.982)</td>
<td>(1.36)</td>
</tr>
<tr>
<td>$\Omega_4$</td>
<td>8.728</td>
<td>5.385</td>
<td>8.734</td>
</tr>
<tr>
<td></td>
<td>5.056</td>
<td>3.299</td>
<td>5.072</td>
</tr>
<tr>
<td></td>
<td>(1.73)</td>
<td>(1.63)</td>
<td>(1.72)</td>
</tr>
<tr>
<td>$\Omega_5$</td>
<td>33.044</td>
<td>20.382</td>
<td>32.931</td>
</tr>
<tr>
<td></td>
<td>15.924</td>
<td>10.290</td>
<td>15.824</td>
</tr>
<tr>
<td></td>
<td>(2.08)</td>
<td>(1.98)</td>
<td>(2.08)</td>
</tr>
</tbody>
</table>

Table 6.4 Time Spent in Subroutine VISC

<table>
<thead>
<tr>
<th>Finest Grid</th>
<th>Problem 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Omega_2$</td>
<td>.151</td>
</tr>
<tr>
<td></td>
<td>.137</td>
</tr>
<tr>
<td></td>
<td>(1.10)</td>
</tr>
<tr>
<td>$\Omega_3$</td>
<td>.582</td>
</tr>
<tr>
<td></td>
<td>.359</td>
</tr>
<tr>
<td></td>
<td>(1.62)</td>
</tr>
<tr>
<td>$\Omega_4$</td>
<td>2.215</td>
</tr>
<tr>
<td></td>
<td>.944</td>
</tr>
<tr>
<td></td>
<td>(2.35)</td>
</tr>
<tr>
<td>$\Omega_5$</td>
<td>8.526</td>
</tr>
<tr>
<td></td>
<td>2.630</td>
</tr>
<tr>
<td></td>
<td>(3.24)</td>
</tr>
</tbody>
</table>
Table 6.5 Time Spent in Subroutine QR

<table>
<thead>
<tr>
<th>Finest Grid</th>
<th>Problem 1</th>
<th>Problem 2</th>
<th>Problem 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Omega_2$</td>
<td>-</td>
<td>.199</td>
<td>.298</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>.087</td>
<td>.118</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>(2.30)</td>
<td>(2.53)</td>
</tr>
<tr>
<td>$\Omega_3$</td>
<td>1.274</td>
<td>.836</td>
<td>1.253</td>
</tr>
<tr>
<td></td>
<td>.303</td>
<td>.224</td>
<td>.304</td>
</tr>
<tr>
<td></td>
<td>(4.21)</td>
<td>(3.73)</td>
<td>(4.12)</td>
</tr>
<tr>
<td>$\Omega_4$</td>
<td>5.162</td>
<td>3.376</td>
<td>5.066</td>
</tr>
<tr>
<td></td>
<td>.754</td>
<td>.552</td>
<td>.759</td>
</tr>
<tr>
<td></td>
<td>(6.85)</td>
<td>(6.12)</td>
<td>(6.67)</td>
</tr>
<tr>
<td>$\Omega_5$</td>
<td>20.597</td>
<td>13.530</td>
<td>20.256</td>
</tr>
<tr>
<td></td>
<td>2.006</td>
<td>1.442</td>
<td>2.010</td>
</tr>
<tr>
<td></td>
<td>(10.27)</td>
<td>(9.38)</td>
<td>(10.08)</td>
</tr>
</tbody>
</table>

donated by that for EULER which is only about two on $\Omega_5$. This factor could possibly be improved by further optimization of the code: reducing scalar temporary references and increasing chaining potential. The same is also true for PCTUVL and VISC. This possibility is currently under investigation and the results presented here are therefore of a preliminary nature.

If may be seen from tables 6.3–4 that the non-vectorizable gather and scatter operations do not significantly degrade the speed-up. Both EULER and VISC are flux calculators and perform similar operations. However VISC contains no conditional branching and hence no gather and scatter technique is necessary. Its factor is not significantly better than that of EULER, which indicates that the gather and scatter process is not dominant.

In conclusion we feel that the results are encouraging. They indicate that the strategies adopted here are satisfactory and that speed-up factors may still be improved by further code optimization.
References


The following reports have appeared in this series:

85-50  Cees Witteveen  Reduction and reconstruction of control knowledge in rule based systems  Part I: Reduction Theory

85-51  Bram van Leer  On numerical dispersion by upwind differencing

85-52  Ph. Clément and C.A. Timmermans  On C0-semigroups generated by differential operators satisfying Ventcel's boundary conditions

86-01  M. Kok and C. Roos  On the feasible region in the objective space of a multiple objective linear programming problem

86-02  Johan H. ter Bekke  OTO-D: Object type oriented data modeling

86-03  Arie H. Boers  The class of n-associative rings

86-04  H.L. Claassen and R.W. Goldbach  The theory of non-symmetric association schemes with three classes

86-05  H.J. van den Herik and I.S. Herschberg  Omniscience, the rulegiver?

86-06  Henk A. van der Vorst  Analysis of a parallel solution method for tridiagonal linear systems

86-07  C. de Wit  Sea level control in a tidal basin

86-08  A. van der Sluis and H.A. van der Vorst  The convergence behaviour of Ritz values in the presence of close eigenvalues

86-09  Geert Jan Olsder  Adaptive Nash strategies for repeated games resulting in Pareto solutions

86-10  Johannes G. Maks  Matrix groups over the transformers in a Clifford algebra and their significance in the conformal and non-Euclidean geometry

86-11  Frits H. Post and Fopke Klok  Deformations of sweep objects in solid modelling

86-12  Michiel van Lambalgen  Von Mises' definition of random sequences reconsidered

86-13  G. Shaw and P. Wesseling  A multigrid method for the compressible Navier-Stokes equations

86-14  G. Shaw and P. Wesseling  Vectorization of a multigrid method for the compressible Navier-Stokes equations

Copies of these reports may be obtained from the bureau of the Department of Mathematics and Informatics, Julianalaan 132, 2628 BL DELFT, phone 015-784568.