# Vectorization Aspects of a Spherical Advection Scheme on a Reduced Grid

J.G. Blom, W. Hundsdorfer, J.G. Verwer

Department of Numerical Mathematics

**NM-R9418 1994**

# Vectorization Aspects of a Spherical Advection Scheme on a Reduced Grid

J.G. Blom, W. Hundsdorfer, J.G. Verwer

*CWI*

*P.O. Box 94079, 1090 GB Amsterdam, The Netherlands*

## Abstract

This paper deals with the numerical solution of the two-dimensional transport equation on a sphere. Two different mass-conservative advection schemes are considered The first is the standard first-order scheme using forward Euler in time and first-order upwind in space. The second scheme uses for time stepping a second-order explicit two-step method and for the space discretization a flux-limited, third-order upwind biased scheme. Both schemes are implemented on a uniform longitude-latitude grid and on a so-called reduced grid, where less cells are used near the poles than at the equator. Since we consider to use Fortran 90 for a software project where the transport equation is part of the problem, we implemented the schemes in Fortran 77 and in Fortran 90 to compare the performance both on scalar and on vector processors. To compare the numerical and the computational performance of all implementations we used a standard test problem describing a solid-body rotation on the sphere (cf. [9]). The results are obtained on a Cray C90.

*AMS Subject Classification (1991):* Primary: 65M20. Secondary: 65M50, 65Y20.
*CR Subject Classification (1991):* G.1.8
*Keywords & Phrases:* hyperbolic partial differential equations, advection on a sphere, method of lines, vectorization.
*Note:* This work was supported by Cray Research, Inc., under grant CRG 94.01, via the Stichting Nationale Computerfaciliteiten (National Computing Facilities Foundation, NCF).

## 1. Introduction

This paper deals with the numerical solution of two-dimensional advective transport of chemical species on the sphere. The model equation describing this process is

$$\frac{\partial c}{\partial t} + \frac{1}{a\cos\phi}\left[\frac{\partial(uc)}{\partial\lambda} + \frac{\partial(vc\cos\phi)}{\partial\phi}\right] = 0, \tag{1.1}$$

where $\lambda \in [0, 2\pi]$ and $\phi \in [-\pi/2, \pi/2]$ are the longitude and the latitude coordinates, and $a$ is the radius of the sphere. $c$ is the vector containing the concentrations of the constituents.

This research is part of the CIRK project [2], which has as ultimate goal a full-scale implementation of a mathematical model for global transport and chemistry of trace constituents in the atmosphere. In realistic simulations of atmospheric transport-chemistry problems the following properties are important for a numerical advection scheme

- mass conservation for long-term computations
- preservation of the positivity of solutions, since negative concentrations can give unstable chemical reactions
- accuracy and computational efficiency

Equation (1.1) has a singularity at the poles ($\phi = \pm\pi/2$). If (1.1) is discretized on a uniform longitude-latitude grid and if an explicit time-stepping scheme is used, this results in a stepsize restriction, due to the CFL condition, that is much stronger near the poles than near the equator. To eliminate this restriction we discretize equation (1.1) also on a *reduced* grid (cf. [12]), which is a grid where the $\Delta\lambda$ is increased approaching the poles.

Two different mass-conservative advection schemes have been implemented on the two grid configurations. The first is the standard first-order scheme using the forward Euler method in time and the first-order upwind discretization in space. The second scheme uses for time stepping the second-order explicit, two-step method which is obtained from the second-order implicit backward differentiation formula by a simple extrapolation step. Here the space discretization is based on the flux-limited, third-order upwind biased scheme examined in [6, 8].

As implementation language for a large software project like CIRK Fortran 90 [1] appears to be a good candidate, since some of its features, viz., dynamic space allocation, pointers, modules, and array syntax, make it easy to implement, e.g., a variable number of grid reductions in standard language elements, this in contrast with Fortran 77. As a preliminary exercise we have implemented the schemes in this paper both in Fortran 90 and in Fortran 77.

The paper addresses three issues:

- Numerical comparison of the two discretization schemes on the uniform longitude-latitude grid and on a reduced grid
- Suitability as implementation language for the CIRK project of Fortran 90 versus Fortran 77
- Performance of the different implementations on scalar and vector processors

As test example we use a 2D standard test problem describing a rotation of a solid body on the sphere (see, e.g., [9, 7]). The performance evaluations were done on a Cray C90 in scalar and in vector mode.

## 2. Discretization of the PDE

Let $\Gamma$ be a uniform longitude-latitude grid on the sphere

$$\Gamma = \{(\lambda_i, \phi_j) \,|\, i = 1, \cdots, N_\lambda,\ j = 1, \cdots, N_\phi\} \tag{2.1}$$

with $\Delta\lambda = 2\pi/N_\lambda$, $\lambda_1 = \Delta\lambda/2$ and $\Delta\phi = \pi/N_\phi$, $\phi_1 = -\pi/2 + \Delta\phi/2$. The spatial discretization on $\Gamma$ of equation (1.1) is given by

$$\frac{\partial c_{ij}}{\partial t} = \frac{-1}{a\cos\phi_j}\left[\frac{\mathrm{fL}_{i+\frac{1}{2}j} - \mathrm{fL}_{i-\frac{1}{2}j}}{\Delta\lambda} + \frac{\mathrm{fP}_{ij+\frac{1}{2}} - \mathrm{fP}_{ij-\frac{1}{2}}}{\Delta\phi}\right], \tag{2.2}$$

where the fluxes on the cell boundaries are given in the state interpolation form

$$\begin{aligned}
\mathrm{fL}_{i+\frac{1}{2}j} &= \max(u(\lambda_{i+\frac{1}{2}}, \phi_j), 0)\, \mathrm{fL}^{+}_{i+\frac{1}{2}j} + \min(u(\lambda_{i+\frac{1}{2}}, \phi_j), 0)\, \mathrm{fL}^{-}_{i+\frac{1}{2}j} \\
\mathrm{fP}_{ij+\frac{1}{2}} &= [\max(v(\lambda_i, \phi_{j+\frac{1}{2}}), 0)\, \mathrm{fP}^{+}_{ij+\frac{1}{2}} + \min(v(\lambda_i, \phi_{j+\frac{1}{2}}), 0)\, \mathrm{fP}^{-}_{ij+\frac{1}{2}}] \cos(\phi_{j+\frac{1}{2}}).
\end{aligned} \tag{2.3}$$

At the poles we impose zero fP-fluxes since $\Delta\lambda = 0$ there.

The simplest form of a conservative and positive discretization scheme is first-order upwind where the fluxes are given by

$$\begin{aligned} \mathrm{fL}^+_{i+\frac{1}{2}j} &= c_{ij}; \qquad \mathrm{fL}^-_{i+\frac{1}{2}j} = c_{i+1j}, \\ \mathrm{fP}^+_{ij+\frac{1}{2}} &= c_{ij}; \qquad \mathrm{fP}^-_{ij+\frac{1}{2}} = c_{ij+1}. \end{aligned} \tag{2.4}$$

If this spatial discretization scheme is combined with forward Euler in time we get the so-called Donor-Cell algorithm:

$$c^{n+1}_{ij} = c^n_{ij} - \frac{\tau}{a\cos\phi_j}\left[\frac{\mathrm{fL}^n_{i+\frac{1}{2}j} - \mathrm{fL}^n_{i-\frac{1}{2}j}}{\Delta\lambda} + \frac{\mathrm{fP}^n_{ij+\frac{1}{2}} - \mathrm{fP}^n_{ij-\frac{1}{2}}}{\Delta\phi}\right], \tag{2.5}$$

with the fluxes given by (2.3) and (2.4).

The second scheme we implemented for the spatial discretization is the mass-conservative, flux-limited, third-order upwind-biased method which was also used in [6]. In this case the fluxes are given by

$$\begin{aligned} \mathrm{fL}^+_{i+\frac{1}{2}j} &= c_{ij} + \Psi(\theta_{i+\frac{1}{2}j})\cdot(c_{i+1j} - c_{ij}); \qquad \mathrm{fL}^-_{i+\frac{1}{2}j} = c_{i+1j} + \Psi(1/\theta_{i+\frac{3}{2}j})\cdot(c_{ij} - c_{i+1j}), \\ \mathrm{fP}^+_{ij+\frac{1}{2}} &= c_{ij} + \Psi(\theta_{ij+\frac{1}{2}})\cdot(c_{ij+1} - c_{ij}); \qquad \mathrm{fP}^-_{ij+\frac{1}{2}} = c_{ij+1} + \Psi(1/\theta_{ij+\frac{3}{2}})\cdot(c_{ij} - c_{ij+1}). \end{aligned} \tag{2.6}$$

with

$$\Psi(\theta) = \max(0, \min(1, \theta, \frac{1}{3} + \frac{1}{6}\theta)), \quad \theta_{i+\frac{1}{2}j} = \frac{c_{ij} - c_{i-1j}}{c_{i+1j} - c_{ij}}, \quad \theta_{ij+\frac{1}{2}} = \frac{c_{ij} - c_{ij-1}}{c_{ij+1} - c_{ij}}. \tag{2.7}$$

To compute the $\theta_{iN_\phi+\frac{1}{2}}$-values at the North Pole we need concentration values across the pole: $c_{iN_\phi+1} \overset{\text{def}}{=} c_{(N_\lambda/2+i-1 \bmod N_\lambda)N_\phi}$ and analogous at the South-Pole.

The time-integration method used in combination with this scheme is an explicit second-order two-step scheme. This scheme is the advective part of the extrapolated explicit-implicit BDF2 scheme that we plan to use for the full-scale model (cf. [10, 11]). The fully discretized version of (1.1) is in this case

$$c^{n+1}_{ij} = \frac{(4c^n_{ij} - c^{n-1}_{ij})}{3} - \frac{2\tau}{3a\cos\phi_j}\left[\frac{\mathrm{fL}^*_{i+\frac{1}{2}j} - \mathrm{fL}^*_{i-\frac{1}{2}j}}{\Delta\lambda} + \frac{\mathrm{fP}^*_{ij+\frac{1}{2}} - \mathrm{fP}^*_{ij-\frac{1}{2}}}{\Delta\phi}\right], \tag{2.8}$$

with $c^* = 2c^n - c^{n-1}$ and the fluxes given by (2.3) and (2.6). For the first time step we use forward Euler as time integrator. Note that in contrast to the Donor-Cell algorithm this scheme is not monotonous.

The Donor-Cell algorithm applied to a linear advection problem in 2D is stable when

$$\max_{i,j}(\nu_\lambda + \nu_\phi) < 1.0, \tag{2.9}$$

with $\nu_\lambda$ and $\nu_\phi$ the 1D Courant numbers. The relevant Courant numbers for horizontal advection on a sphere are

$$\nu_\lambda = \max_{i,j}\frac{\tau}{\Delta\lambda}\frac{|u(\lambda_{i+\frac{1}{2}}, \phi_j)|}{a\cos\phi_j}, \quad \nu_\phi = \max_{i,j}\frac{\tau}{\Delta\phi}\frac{|v(\lambda_i, \phi_{j+\frac{1}{2}})\cos\phi_{j+\frac{1}{2}}|}{a\cos\phi_j}. \tag{2.10}$$

It is obvious that $\nu_\lambda$ can become arbitrarily large near the poles for nonzero $u$. For scheme (2.8) a similar condition holds, only here the maximum is restricted to 0.45 instead of 1.0 (cf. [10]).

## *2.1 Reduced Grid*

To obviate the strong time-step restriction due to the (numerical) singularities near the poles we use the *reduced-grid* approach [12]. In our implementation the cell-width along the latitudes is doubled each time the time-step restriction at that latitude becomes a factor two larger than the time-step restriction at the equator. This results in a domain decomposition approach where uniform grid blocks of different latitude cell-widths are connected. At the interface boundaries the required concentrations are obtained by linear interpolation. To preserve mass conservation we impose as boundary conditions for the coarser blocks the fluxes computed on the boundary of the neighboring finer domains.

If we assume one grid coarsening at $\phi_{J+\frac{1}{2}}$, and consequently also at $\phi_{N_\phi-J+\frac{1}{2}}$, we would get the following implementation for the Donor-Cell algorithm:

- First handle the equatorial area, i.e., compute (2.5) for $i = 1, \cdots, N_\lambda$ and $j = J+1, \cdots, N_\phi - J$. If values $c_{iJ}$ or $c_{iN_\phi-J+1}$ are needed they are obtained by linear interpolation from the available values

$$\begin{aligned} c_{ij} &= (3c_{\frac{i}{2}j} + c_{\frac{i}{2}+1j})/4, \quad i \text{ even} \\ c_{ij} &= (3c_{\frac{i}{2}+1j} + c_{\frac{i}{2}j})/4, \quad i \text{ odd} \end{aligned}$$

- Next, compute the boundary conditions for the South-Pole area

$$\begin{aligned} \mathrm{fP}_{iJ+\frac{1}{2}} &= (\mathrm{fP}_{2i-1J+\frac{1}{2}} + \mathrm{fP}_{2iJ+\frac{1}{2}})/2 \\ \mathrm{fP}_{i\frac{1}{2}} &= 0, \qquad\qquad \text{for } i = 1, \cdots, N_\lambda/2 \end{aligned}$$

  and compute the concentration values in the South-Pole area with (2.5), $\Delta\lambda/2$, for $i = 1, \cdots, N_\lambda/2$ and $j = 1, \cdots, J$.

- Finally, compute the boundary conditions for the North-Pole area

$$\begin{aligned} \mathrm{fP}_{iN_\phi-J+\frac{1}{2}} &= (\mathrm{fP}_{2i-1N_\phi-J+\frac{1}{2}} + \mathrm{fP}_{2iN_\phi-J+\frac{1}{2}})/2 \\ \mathrm{fP}_{iN_\phi+\frac{1}{2}} &= 0, \qquad\qquad \text{for } i = 1, \cdots, N_\lambda/2 \end{aligned}$$

  and compute the concentration values in the North-Pole area with (2.5), $\Delta\lambda/2$, for $i = 1, \cdots, N_\lambda/2$ and $j = N_\phi - J + 1, \cdots, N_\phi$.

The implementation for more domains or for the second-order scheme (2.8) is analogous.

Note that the Donor-Cell implemented on a reduced grid is no longer monotonous even for constant concentrations. This can be explained by the fact that the windfield is no longer divergence free (in the numerical sense). The spatial discretization scheme (2.6) implemented in this way can even result in negative values.

## 3. Implementation

For a simple method like the Donor-Cell scheme on a uniform grid, where the computation of the fluxes and the time integration can be done inline without being detrimental to the readability of the program, the choice of language is not very important. For the development of a full-scale implementation of a mathematical model for global transport and chemistry of trace constituents in the atmosphere one needs a language which offers the flexibility to replace modules and adapt data structures without rewriting the whole program or having to write links between one part of the program and another.

### *3.1 Fortran 90*

Fortran 90 offers a number of attractive language elements that ease the development of large scale applications. One of the goals of this paper is to see in what extent these theoretical advantages can be translated into practice, and especially at what computational and programmer costs.

Here we first enumerate some of the Fortran 90 features and compare them to their Fortran 77 counterparts.

- To define the problem, the size of the grid, etc., Fortran 90 offers the MODULE and USE construct. This makes the Fortran 77 common blocks obsolete and replaces the non-standard INCLUDE statement. Moreover, the language allows dependent compilation, which means, e.g., that one does not have to re-compile the solver if it needs for a different grid size more memory storage. Typical use of a MODULE is for global data (like common blocks were) and for the packaging of derived data types and the operators and subroutines defined for these data types. The latter has no equivalent in Fortran 77, but is comparable to the Ada [5] **package** idea.
- Both dynamic and automatic space allocation is possible, so no longer large workarrays have to be declared and subdivided in the program, which is error-prone and unreadable.
- Fortran 90 supports array syntax. With respect to vectorization efficiency this means that the compiler can obtain more and easier information about what part of the array is needed; a disadvantage is that more loop fusion capabilities of the compiler are required. For the programmer it means that for instance rows of a two-dimensional array are as easily accessible as columns and that non-contiguous parts of an array can act as actual parameter for a subprogram.
- In Fortran 90 functions can return other than scalar types, such as arrays. This has the advantage that many 'standard' subprograms in ODE/PDE programming, like flux computation, right hand-side evaluation, etc., can be PURE functions, although the PURE concept itself is not yet a part of the language. A compiler then knows that since this subprogram has no side effects it can be executed in parallel with another subprogram.
- A variable of any data type can have the POINTER attribute. This is not only convenient when building non-regular structures (like reduced grids), but also to prevent copying large arrays when implementing a multi-step integration formula. Another prospect for pointers is the use as an alias for a specific array section.
- Some other handy features of Fortran 90 are:
  - IMPLICIT NONE is part of the standard
  - Possibility to use a data type with a given numeric requirement
  - Numeric inquiry functions
  - Derived data types (records, etc.)
  - Operator overloading

As datastructure for windfields and concentrations living on a reduced grid we chose a one-dimensional array with dimension equal to the number of subdomains. In this array pointers are stored to the relevant data that is stored in regular arrays with (first) dimension $N_{\lambda l} \times N_\phi$ where $N_{\lambda l} = 2\pi/\Delta\lambda_l$ and $\Delta\lambda_l$ the cell width along the latitudes in that specific subdomain. This structure has the advantage that the computations can be easily parallelized over the subdomains and within the subdomains vectorize as for a uniform grid.

In our implementation three MODULEs are used:

1. For the definition and initialization of the grid parameters and variables. In the case of a uniform grid this module contains mainly $N_\lambda, N_\phi, \Delta\lambda, \Delta\phi$ and $\cos\phi_j$. For the reduced grid the indices of the $\phi$-coordinates where the grid reduction takes place are added.

2. For the definition and initialization of the windfield. In case of a reduced grid this involves also the allocation of the memory space for the windfield on the specific subdomains.
3. To define and initialize the concentration values. On a reduced grid memory space has to be allocated for the datastructures of the concentrations at all required time levels. In this module also the special operations are defined for expanding and contracting concentration values near the boundaries of a subdomain.

Concentration values are stored in arrays with the POINTER attribute, so that after a time step only the pointers have to be changed and not whole arrays have to be copied. This is only slightly more expensive than unrolling the time loop, but the latter makes the program less flexible.

For the flux computations one has the choice between three different implementations which are in our opinion all three equally readable

1. the MAX MIN formulation like in (2.4) and (2.6)
2. a WHERE construct
3. the IF THEN ELSE construct

Of these three we use the first, since this resulted in the most optimal vector code. However, we emphasize that the current compiler version has an erratic vectorization optimization, which makes it difficult to make a proper choice between various constructs.

The upwind flux (2.4) is implemented inline. For the computation of the third-order upwind-biased flux (2.6) we did not use functions with a pointer or array result, but subroutines. This to avoid repeated memory allocations in the flux function or the need to copy the result arrays. Again, with a different compiler it is conceivable that a PURE function with a pointer-to-array result can be equally efficient.

The time integration is implemented analogous to the flux computation for the same reasons as above.

### *3.2 Fortran 77*

The framework of the Fortran 77 program is similar to that of the Fortran 90 program. The MODULEs are replaced by - nonstandard - INCLUDE files with common blocks that contain the global values and by separate sources that define the initialization, etc..

The data on a reduced grid is stored in an array with dimension $N_{\lambda 0} \times N_\phi$. Although this can lead to a large amount of unused memory reservation (especially for systems with a large number of components), we preferred this option because of its flexibility and ease of programming.

## 4. Numerical experiments

### *4.1 Problem definition*

To measure the accuracy and the performance of the algorithms we used a standard test problem describing a solid-body rotation on the sphere (see, e.g., [9, 7]). The initial profile for the test problem is

$$c^0(\lambda,\phi) = \begin{cases} 1 - \gamma \frac{r^0(\lambda,\phi)}{R} & \text{if } r^0(\lambda,\phi) < R \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

which is a cone ($\gamma = 1$) or a cylinder ($\gamma = 0$) with a radius of $R = 7\,\Delta\lambda$. The cylinder or cone is centered in $(\lambda_0, \phi_0)$ and $r^0$ is given by

| | $1/\Delta t$ | $\|\cdot\|_\infty$ | EMIN | EMAX | ERR0 | ERR1 | ERR2 |
|---|---|---|---|---|---|---|---|
| DC | 5215 | 0.75E+00 | 0.35E-09 | -0.83E+00 | 0.63E-01 | 0.34E-12 | -0.86E+00 |
| $\mathrm{DC}_{rg}$ | 652 | 0.76E+00 | 0.53E-10 | -0.84E+00 | 0.66E-01 | -0.14E-13 | -0.86E+00 |
| B2K3 | 11587 | 0.24E+00 | 0 | -0.26E+00 | 0.15E-01 | -0.54E-10 | -0.19E+00 |
| $\mathrm{B2K3}_{rg}$ | 1449 | 0.26E+00 | -0.32E-05 | -0.29E+00 | 0.16E-01 | -0.68E-11 | -0.21E+00 |
| DC | 3687 | 0.76E+00 | 0.23E-15 | -0.84E+00 | 0.63E-01 | 0.36E-12 | -0.86E+00 |
| $\mathrm{DC}_{rg}$ | 461 | 0.75E+00 | 0.15E-15 | -0.83E+00 | 0.64E-01 | -0.71E-14 | -0.85E+00 |
| B2K3 | 8194 | 0.25E+00 | 0 | -0.28E+00 | 0.13E-01 | -0.38E-10 | -0.18E+00 |
| $\mathrm{B2K3}_{rg}$ | 1025 | 0.26E+00 | -0.36E-05 | -0.28E+00 | 0.14E-01 | -0.47E-11 | -0.18E+00 |

Table 1: Results for one revolution of the cone around the sphere over both poles (above) and along a trajectory that intersects the parallels of latitude at an angle of $45^o$ (below)

.

| | $1/\Delta t$ | $\|\cdot\|_\infty$ | EMIN | EMAX | ERR0 | ERR1 | ERR2 |
|---|---|---|---|---|---|---|---|
| DC | 5215 | 0.79E+00 | 0.15E-08 | -0.30E+00 | 0.67E-01 | -0.19E-11 | -0.23E-01 |
| $\mathrm{DC}_{rg}$ | 652 | 0.77E+00 | -0.19E-02 | -0.31E+00 | 0.69E-01 | -0.30E-12 | -0.23E-01 |
| B2K3 | 11587 | 0.52E+00 | -0.27E-10 | -0.85E-03 | 0.30E-01 | -0.44E-10 | -0.75E-02 |
| $\mathrm{B2K3}_{rg}$ | 1449 | 0.53E+00 | -0.24E-02 | 0.15E-02 | 0.32E-01 | -0.65E-11 | -0.77E-02 |
| DC | 3687 | 0.76E+00 | 0.00E+00 | -0.30E+00 | 0.65E-01 | -0.13E-11 | -0.23E-01 |
| $\mathrm{DC}_{rg}$ | 461 | 0.74E+00 | -0.34E-02 | -0.28E+00 | 0.66E-01 | -0.18E-12 | -0.22E-01 |
| B2K3 | 8194 | 0.52E+00 | -0.19E-10 | -0.14E-03 | 0.29E-01 | -0.33E-10 | -0.72E-02 |
| $\mathrm{B2K3}_{rg}$ | 1025 | 0.53E+00 | -0.28E-02 | 0.14E-02 | 0.31E-01 | -0.46E-11 | -0.73E-02 |

Table 2: Results for one revolution of the cylinder (with background) around the sphere over both poles (above) and along a trajectory that intersects the parallels of latitude at an angle of $45^o$ (below)

.

$$r^0(\lambda,\phi) = 2\{\cos^2\phi\, sin^2[(\lambda-\lambda_0)/2] + sin^2[(\phi-\phi_0)/2]\}^{\frac{1}{2}}. \tag{4.2}$$

The profile is advected in a stationary windfield

$$\begin{aligned} u(\lambda,\phi) &= 2\pi(\cos\beta\cos\phi + \sin\beta\sin\phi\cos\lambda) && (4.3)\\ v(\lambda,\phi) &= -2\pi\sin\beta\sin\lambda, && (4.4) \end{aligned}$$

where $\beta$ is an arbitrary angle with the equator. This velocity field implies that at time $t = 1$ the solid-body is back at its initial position.

### *4.2 Numerical results*

We performed two different tests. One where the solid-body traverses one full rotation over both poles ($\beta = \frac{1}{2}\pi$, $(\lambda_0,\phi_0) = (\frac{3}{2}\pi, 0)$), and one where the profile passes the North Pole and its trajectory intersects the parallels of latitude at an angle of 45 degrees ($\beta = \frac{1}{4}\pi$, $(\lambda_0,\phi_0) = (\pi, 0)$).

| | $1/\Delta t$ | $\|\cdot\|_\infty$ | EMIN | EMAX | ERR0 | ERR1 | ERR2 |
|---|---|---|---|---|---|---|---|
| DC | 5215 | 0.79E+00 | 0.30E-08 | -0.61E+00 | 0.13E+00 | 0.62E-12 | -0.81E+00 |
| $\mathrm{DC}_{rg}$ | 652 | 0.77E+00 | 0.35E-09 | -0.61E+00 | 0.14E+00 | -0.71E-14 | -0.81E+00 |
| B2K3 | 11587 | 0.52E+0 | 0 | -0.17E-02 | 0.60E-01 | -0.59E-10 | -0.26E+00 |
| $\mathrm{B2K3}_{rg}$ | 1449 | 0.53E+00 | -0.42E-05 | 0.22E-02 | 0.64E-01 | -0.78E-11 | -0.27E+00 |

Table 3: Results for one revolution of the cylinder (without background) around the sphere over both poles.

For the uniform grid we took a grid distance of $2.8125^o$ in both directions, i.e., $N_\lambda = 128$ and $N_\phi = 64$. The time step sizes were taken as large as allowed by the CFL condition. For the reduced grid we double the latitude cell width at $\phi = 61.875^o, 75.9375^o, 84.375^o$. This results in a minimum value for $\Delta\lambda \cos\phi$ which is ca. 2.5 times smaller than the value at the equator. The total number of grid points on this grid is 6304, which is approximately 75% of the number of grid points on the uniform grid.

In Tables 1-3 we give the errors for the two profiles approximated with the Donor-Cell scheme ((2.5), indicated by 'DC' in the tables) and the second-order scheme ((2.8), indicated by 'B2K3'). Results on a reduced grid are indicated by $.._{rg}$. The error measures used in Tables 1-3 are the same as in [9, 7]

$$\text{EMIN} = \frac{\min c_{ij} - \min c^0_{ij}}{\max c^0_{ij}} \tag{4.5}$$

$$\text{EMAX} = \frac{\max c_{ij} - \max c^0_{ij}}{\max c^0_{ij}} \tag{4.6}$$

$$\text{ERR0} = \frac{(\sum \cos\phi_j (c_{ij} - c^0_{ij})^2)^{\frac{1}{2}}}{(\sum \cos\phi_j)^{\frac{1}{2}} \cdot \max c^0_{ij}} \tag{4.7}$$

$$\text{ERR1} = \frac{\sum \cos\phi_j c_{ij}}{\sum \cos\phi_j c^0_{ij}} - 1 \tag{4.8}$$

$$\text{ERR2} = \frac{\sum \cos\phi_j c^2_{ij}}{\sum \cos\phi_j (c^0_{ij})^2} - 1 \tag{4.9}$$

where the maximum, minimum and sum operators are taken over all grid points. ERR0 acts as an $L_2$-norm on a sphere, ERR1 is an indicator for the mass conservation, and ERR2 is a measure for the diffusion of the initial profile. $\|\cdot\|_\infty$ denotes the usual maximum norm.

In Table 1 the errors are given for the cone profile. It is obvious that the third-order upwind-biased scheme is much less diffusive than the first-order upwind scheme, even in its limited form. Of course the computational costs are also higher, per time step B2K3 is ca. 4-5 times as expensive as DC (see also the next subsection). The errors on the reduced grid are only slightly worse than the errors obtained on the uniform grid. The negative values that result from the computations with the third-order upwind-biased scheme are acceptably small. Note that the negative values that arise when we do not limit the third-order scheme are much larger, ca. $-0.03$. For neither method nor grid there is much difference between the results for a trajectory perpendicular to the parallels of latitude or for a trajectory that intersects these parallels at an angle of $45^o$.

Table 2 lists the results for the cylinder profile, but placed on a background of 1, so the top of the cylinder is 2. It is obvious that the computations on a reduced grid result in a non-monotonous solution. That this is due to the grid reductions and not to the different shape can be seen from the results in Table 3. In fact it occurs even for a constant solution. Closer examination shows that this non-monotonicity is due to the fact that the windfield is no longer (numerically) divergence free. Since the top of this profile is much wider the diffusive effects are less pronounced. Again for this test it shows that the mass conservation is almost up to numerical precision.

## *4.3 Global Performance*

The performance evaluation was done on a Cray C90. Scalar results were obtained using `f90 -O scalar3 -O vector0 -O task0` and `cf77 -Wf"-o novector"`, and vector results with `f90 -O vector3` and `cf77 -Zv`, for the Fortran 90 and Fortran 77 code, respectively. The timings were done on 1 processor with a clock cycle time of 4.2ns and a double vector pipe. This gives a theoretical peak performance on 1 processor of 476 Mflop/s and 952 when chaining an add and a multiply. To measure the Megaflop rate and the CPU time of a routine we used the Cray utility Perftrace[3], that gives the hardware performance by program unit.

| | scalar | | | | vector | | | |
|---|---|---|---|---|---|---|---|---|
| | Fortran 90 | | Fortran 77 | | Fortran 90 | | Fortran 77 | |
| | CP sec | Mflop | CP sec | Mflop | CP sec | Mflop | CP sec | Mflop |
| DC | 32.4 | 20 | 25.4 | 30 | 1.8 | 420 | 1.6 | 450 |
| B2K3 | 276.0 | 25 | 195.0 | 35 | 17.0 | 420 | 15.6 | 460 |
| $\mathrm{DC}_{rg}$ | 4.2 | 10 | 3.1 | 25 | 0.8 | 60 | 0.2 | 320 |
| $\mathrm{B2K3}_{rg}$ | 32.6 | 25 | 20.4 | 35 | 5.7 | 140 | 2.2 | 320 |

Table 4: Global performance.

| | Fortran 90 | | | | | Fortran 77 | | | |
|---|---|---|---|---|---|---|---|---|---|
| | # calls | Avg.time | ACM % | Mflop | | # calls | Avg.time | ACM % | Mflop |
| FLUXL | 11587 | 5.5E-4 | 37.3 | 477 | FLUXL | 11587 | 5.5E-4 | 41.0 | 476 |
| FLUXP | 11587 | 5.1E-4 | 72.2 | 506 | FLUXP | 11587 | 4.9E-4 | 77.9 | 524 |
| BDF2K3 | 1 | 2.4E+0 | 86.1 | 80 | BDF2K3 | 1 | 1.8E+0 | 89.2 | 108 |
| BDF2E | 11586 | 2.0E-4 | 100.0 | 405 | BDF2E | 11586 | 1.5E-4 | 100.0 | 568 |

Table 5: Vector performance of B2K3 on a uniform grid
ACM %: Accumulated percentage of CPU-time spent

We want to emphasize that the CF90 programming environment is the first release of a Fortran 90 compiler on a Cray and thus can not be expected to be as good as the seasoned CF77 5.0 compiler. According to the CF90 Programming Environment 1.0 Release Overview [4] the execution time of CF90 1.0 programs generated with default options (i.e., vectorized) are typically 5-10% slower than CF77 5.0 programs.

First we give a global survey of the efficiency of the four different implementations. In Table 4 the CPU time and the Mflop rate is given for the Fortran 90 and for the Fortran 77 codes, both on a uniform and on the reduced grid. On a uniform grid both implementations are alike, and indeed, the difference in execution time for the vectorized versions fits in the 5-10% margin for both DC and B2K3. In scalar mode the difference, however, is much larger. But it is likely that scalar optimization is not the first issue for compiler builders on a supercomputer. As mentioned in the previous subsection the difference in computational costs per time step between the first-order upwind scheme and the third-order scheme is a factor 5. Since for stability reasons also approximately twice as much time steps are needed the B2K3 scheme is ca. 10 times as expensive as the Donor-Cell scheme.

To compute one revolution on a reduced grid is approximately 8 times as cheap as on a uniform grid for both methods and for all language / optimization combinations, except for the vectorized Fortran 90 version. We will discuss this disappointing result in the next subsection. Note that also ca. 8 times as few time steps are required to satisfy the CFL conditions. This means that gain in efficiency due to the reduction in grid points roughly balances the extra computational overhead for the reduced grid implementation.

## *4.4 Vector Performance*

In this section we will discuss the vector performance of all combinations. As was mentioned previously the Donor-Cell scheme is essentially one program without any subroutines so the performance given in Table 4 cannot be discussed in detail. There is one exception: the implementation of the DC scheme on a reduced grid in Fortran 90 contains subroutines for the flux computations and for the time integration just as the implementation of the B2K3 scheme. If these subroutines are made inline the total CPUtime would be 14 seconds instead of 0.8 seconds. This is most likely caused by the pointer-to-array structure which is ‘unwrapped’ when a subroutine is called but not when the code is inline. A disadvantage of subroutines is in this case naturally the simplicity of the flux computations resulting in a Megaflop rate of 40 for the flux subroutines. However, it is likely that in future compiler

| | Fortran 90 | | | | Fortran 77 | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | # calls | Avg.time | ACM % | Mflop | | # calls | Avg.time | ACM % | Mflop |
| BDF2K3 | 1 | 2.2E+0 | 39.1 | 9 | FLUXL | 10143 | 7.2E-5 | 33.1 | 398 |
| FLUXL | 10143 | 1.0E-4 | 57.1 | 317 | FLUXP | 10143 | 6.5E-5 | 62.6 | 462 |
| FLUXP | 10143 | 9.7E-5 | 74.5 | 336 | BDF2K3 | 1 | 6.3E-1 | 90.9 | 38 |
| EXPAND | 17388 | 4.9E-5 | 89.4 | 5 | BDF2E | 1448 | 1.2E-4 | 98.5 | 552 |
| BDF2E | 10136 | 3.0E-5 | 94.8 | 303 | EXPAND | 17388 | 1.2E-6 | 99.4 | 153 |

Table 6: Vector performance of B2K3 on a reduced grid
ACM %: Accumulated percentage of CPU-time spent

releases this 'unwrapping' of arrays 'behind' pointers will also be done for inline code.

Table 5 gives the vector performance of the B2K3 scheme on a uniform grid. It shows that the computationally expensive part is very efficient both in Fortran 90 and in Fortran 77. The main program is significantly slower in Fortran 90, again presumably caused by the use of pointers to array, compared with the copying of the arrays that is done in Fortran 77. In that case the difference between the two programs should be less for the full-scale model, since then the copying of the arrays takes much more time.

The vector performance for the 'computing subroutines' on a reduced grid is still very satisfactorily as can be seen from Table 6. However, in this case the calling program in Fortran 90 is exceptionally more expensive than when implemented in Fortran 77.

## 5. Fortran 90 versus Fortran 77

It is clear that the possibilities to write flexible and easy-to-understand programs in Fortran has increased tremendously with the new Fortran 90 standard. Some attractive features are copied from ALGOL 68 and Ada resulting in a modern language well-designed for (vector) computers. A disadvantage could be that the old standard, although describing an essentially different language, is for compatibility reasons a subset of the new language definition, making the life for compiler builders much harder.

The first CF90 release contained some bugs that made the programming less easy, but none was so serious that there could not be found an alternative implementation that was acceptable. However, as yet it is not so easy to write an efficiently vectorizable program. The performance is erratical, e.g., loops with known, constant, upperbound can be slower than loops where the upperbound is a variable; array syntax can result in a twice as slow performance than when the loops are simply written out, etc.. But no doubt a next release will meet most of these objections.

## References

1. Fortran 90, ISO/IEC 1539: 1991, 1992.
2. The CIRK project: Mathematical modeling of global transport and chemistry of trace constituents in the troposphere. Internal CWI note, unpublished, May 1993.
3. Cray Research, Inc. *UNICOS Performance Utilities Reference Manual*, SR-2040 6.0 edition, 1991.
4. Cray Research, Inc. *CF90 Programming Environment 1.0 Release Overview*, RO-5030 1.0 edition, 1993.
5. G. Goos and J. Hartmanis, editors. *The Programming Language Ada, Reference Manual*, volume 106 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin Heidelberg New York, 1981.
6. W. Hundsdorfer, B. Koren, M. van Loon, and J.G. Verwer. A positive finite-difference advection scheme. *(to appear in J. Comput. Phys.)*, 1994. Revision of CWI Report NM-R9309.
7. W. Hundsdorfer and E.J. Spee. Dimensional splitting with unconditional stability for advection on a sphere. Report NM-R9416, CWI, Amsterdam, 1994.

8. B. Koren. A robust upwind discretization method for advection, diffusion and source terms. In C.B. Vreugdenhil and B. Koren, editors, *Numerical Methods for Advection-Diffusion Problems, Notes on Numerical Fluid Mechanics 45*. Vieweg, Braunschweig, 1993.
9. P.K. Smolarkiewicz and P.J. Rasch. Monotone advection on the sphere: An Eulerian versus semi-Lagrangian approach. *J. Atmos. Sci.*, 48(6):793–810, 1991.
10. J.G. Verwer and J.G.Blom. An implicit-explicit method for atmospheric transport-chemistry problems. Report NM-R94xx, CWI, Amsterdam, 1994. In preparation.
11. J.G. Verwer and D. Simpson. Explicit methods for stiff ODEs from atmospheric chemistry. Report NM-R9409, CWI, Amsterdam, 1994.
12. D.L. Williamson. Review of numerical approaches for modeling global transport. In H. van Dop and G. Kallos, editors, *Air Pollution Modeling and Its Application IX*. Plenum Press, New York, 1992.