Pergamon

Atmospheric Environment Vol. 28, No. 2, pp. 273–284, 1994
Elsevier Science Ltd
Printed in Great Britain. All rights reserved
1352–2310/94 $6.00 + 0.00

SMVGEAR: A SPARSE-MATRIX, VECTORIZED GEAR CODE FOR ATMOSPHERIC MODELS*

MARK Z. JACOBSON and RICHARD P. TURCO

Department of Atmospheric Sciences, 405 Hilgard Ave, University of California, Los Angeles, CA 90024-1565, U.S.A.

(*First received* 7 *January* 1993 *and in final form* 15 *June* 1993)

Abstract—We present a Gear-type code that efficiently solves ordinary differential equations in large grid-domains. To obtain the final code, we modified an original program of C.W. Gear, built and added a sparse-matrix package, and vectorized all loops about the grid-cell dimension. Furthermore, to obtain at least 90% vectorization potential while preventing equations in some regions of the grid from slowing the solution over the entire grid-domain, we divided the domain into blocks of grid-cells and vectorized around these blocks. The sparse-matrix solution reduced the average number of LU-decomposition calculations, compared to a full-matrix solution, by factors of between 20 (for a matrix of order 40) and 120 (for a matrix of order 90). It also reduced both back-substitution calculations and total array space by factors of between 5 and 12 for the above matrix sizes. Vectorization on a CRAY-90 computer increased the speed by another factor of about 120 over the code running in scalar form. We tested the speed and accuracy of the program for several chemical applications on a single processor of the CRAY-90 computer. The code averaged between 1 and 2 min of computer time per day of simulation to solve a smog-chemistry set of 92 species and 222 reactions over a 10,000-cell grid, with continuously changing photorates. It also took 3–4 min per day to solve a stratospheric-chemistry set of 39 species and 108 reactions over a 100,000-cell grid. In addition, we tested the speed of the code while it solved aqueous chemistry in 43 aerosol size bins, along with other physical processes and transport, over a large grid. Finally, we compared the speed and other statistics from SMVGEAR to those of an existing sparse matrix Gear code, LSODES, and to a new method that we call the Multistep Implicit–Explicit (MIE) method.

Key word index: Gear code, air pollution photochemistry, stratospheric photochemistry, aqueous chemistry, ordinary differential equations.

1. INTRODUCTION

Over the years, mathematicians and scientists have developed techniques to solve systems of stiff ordinary differential equations (ODEs). In chemical terms, a stiff system occurs when the lifetimes of some species are many orders of magnitude smaller than the lifetimes of other species. Because explicit ODE solvers require numerous short time-steps in order to maintain stability, most current techniques solve ODEs implicitly or semi-implicitly.

Among the high- and variable-order techniques to solve stiff ODEs include variations of Runge–Kutta, Richardson Extrapolation/Bulirsch–Stoer, and predictor–corrector methods (Press *et al.*, 1992; Gear, 1967, 1969, 1971; Stoer and Bulirsch, 1980). In particular, Gear's backward differentiation predictor–corrector scheme spawned a number of advanced Gear-type codes (e.g. Hindmarsh, 1972, 1974, 1975, 1976, 1977, 1980, 1983; Spellman and Hindmarsh, 1975; Morris and Hindmarsh, 1975; Byrne and Hindmarsh, 1976; Sherman and Hindmarsh, 1980; Brown and Hindmarsh, 1986).

Other techniques to solve stiff ODEs include the midpoint and trapezoidal methods (e.g. Gear, 1971; Press *et al.*, 1992), family methods (e.g. Crutzen, 1971; Turco and Whitten, 1974; Brasseur and Solomon, 1984; Austin, 1991; Elliott *et al.* 1993), parameterization methods (e.g. Jacob *et al.*, 1989), hybrid predictor–corrector methods (e.g. McRae *et al.*, 1982), iterative backward differentiation methods (e.g. Curtiss and Hirschfelder, 1952; Hunt, 1966; Shimazaki and Laird, 1970; Turco and Whitten, 1974; Rosenbaum, 1976), iterative hybrid schemes (Hesstvedt *et al.*; 1978) and many others.

To date, one limitation of most schemes has been their inability to solve equations *both* quickly and with a high order of accuracy in multiple grid-cell models. With the onset of faster vector machines and parallel processors, this limitation is eroding. However, even with a fast machine, accurate solutions in 10^4–10^5 grid-cells over simulation periods of months to years still requires considerable computer time. Here, we introduce SMVGEAR (Sparse-Matrix Vectorized Gear Code), which is designed to reach peak speed performance on vectorized machines when the number of grid-cells is large. On non-vector machines, SMVGEAR also has advantages.

The first building-block of SMVGEAR is an original code of Gear (1971, pp. 158–166). Gear's code

* To obtain a copy of this code, please send a request by either e-mail to jacobson@yosemite.atmos.ucla.edu or regular mail to the authors at the address above.

solves systems of ODEs using a time-step and order dependent on the stiffness of the equations. In summary, Gear's code solves the set of ODEs (e.g. Hindmarsh, 1983).

$$\mathrm{d}\mathbf{y}/\mathrm{d}t = f(t,\mathbf{y}) \tag{1}$$

where $\mathbf{y}$ is a vector of species concentrations, and t is time. To solve the equations, Gear sets up a prediction matrix

$$P \approx I - h\beta_0 J \tag{2}$$

where I is the identity matrix, h the time-step, β_0 a scalar multiplier that depends on the order of the method, and

$$J = \partial f/\partial \mathbf{y} \tag{3}$$

is a Jacobian matrix of partial derivative for the system. With the prediction matrix, the code iteratively solves the equation

$$P\mathbf{x} = B \tag{4}$$

where $\mathbf{x}$ is a vector used to correct $\mathbf{y}$ and its derivatives, and B is a continuously changing vector found from evaluating equation (1) with corrected values of $\mathbf{y}$. When the error from the correction vector $\mathbf{x}$ becomes less than a provided error, the corrector has converged, and the code then checks whether the sum of errors over all iterations for a time-step is less than another error tolerance. If the system of equations passes the second error check, the time-step is complete. Otherwise, the code either re-evaluates the Jacobian, reduces the time-step and re-evaluates the Jacobian, or reduces the order and time-step and re-evaluates the Jacobian.

After a number of successful time-steps dependent on the value of the current order, the code re-estimates the time-step feasible at one order lower than, the same order as, and one order higher than the current order. It then chooses the next time-step as the largest of the three estimates and chooses the order as that order allowing the largest step.

While Gear's original code is elegant, a shortcoming is its need to decompose a matrix many times and perform numerous back-substitutions for each decomposition. Both full matrix decomposition and back-substitution require a significant number of calculations. In response to this problem, researchers at Lawrence Livermore Laboratory and elsewhere began to look for ways to reduce the number of matrix calculations.

A characteristic of most ODE systems is that their initial matrices contain many zero values. To exploit the sparse structure of these matrices, Hindmarsh (1975) introduced the first of several versions of the Gear code that solves Jacobian matrices that have banded structures. In addition, Spellman and Hindmarsh (1975), Sherman and Hindmarsh (1980), and Hindmarsh (1983), introduced Gear codes that solve Jacobians that have arbitrary sparsity. These latter programs used subroutines from the Yale Sparse Matrix Package (Sherman, 1975; Eisenstat *et al.*, 1977a, b). More recently, the group at Livermore has studied matrix-free methods of integrating stiff ODEs (Brown and Hindmarsh 1986; Hindmarsh, 1986; Byrne, 1990).

The modified Gear-code and the accompanying sparse-matrix solver we present, SMVGEAR, have some similarities to, yet several differences from, some of the codes developed at Livermore. In this paper, we describe how we built SMVGEAR, compare SMVGEAR to LSODES, a 1987 sparse-matrix Gear code (see Hindmarsh, 1983, for the most recent update), and compare SMVGEAR to a new forward-backward Euler method, which we call the Multistep Implicit–Explicit (MIE) scheme.

2. BUILDING SMVGEAR

Starting with Gear's 1971 code, we made major modifications in two areas. We vectorized the code around the grid-cell dimension and built and implemented a vectorized sparse-matrix package. Other changes were minor in comparison.

2.1. *Vectorization around the grid-cell dimension*

Vectorization is a process by which many computers speed up calculations in an inner DO ... CONTINUE loop. Up to a point, the longer the inner vectorized loop, the faster the calculations in the loop, and the greater the number of megaflops (million floating point operations per second) the program achieves. On Cray computers, such as the CRAY Y-MP, a vectorized loop reaches 90% of its maximum speed when the inner vector length is about 512 elements (NAS, 1992). Vectorized loops are also sensitive to the types of operations performed within. For example, a loop will often not vectorize if a recursion occurs within it. Also, vectorized loops that contain certain operations or certain pointer arrays are often slower than vectorized loops written strategically with different operations or without the arrays.

At first glance, a Gear-type code appears to have two dimensions around which one can vectorize. One dimension is the number of species, and a less apparent dimension is the number of grid-cells. If one runs a Gear-type code in a single grid-cell, then the only practical vectorization dimension is the number of species. However, running a Gear-code in one grid-cell rarely presents a computational problem in the first place. Also, most atmospheric models contemplate calculations in multiple grid-cells.

Based on experience, we have found that an atmospheric model vectorized around the grid-cell dimension is much faster than one vectorized around the number of species when the model contains more than a few tens of grid-cells. A reason is that all classical matrix decomposition/back substitution codes require several varying-length inner loops. For example, matrix inner loops often march from length of one to

length of either the order or one-half the order of the matrix. Thus, if the model has many grid-cells, it will perform calculations in each cell with many inefficient inner loops.

However, if we vectorize the code around the grid-cell dimension, then the inner loop is efficient so long as the model contains at least a few tens of grid-cells. Theoretically, for a model with 50,000 grid-cells, *every* inner loop can have a length of 50,000. However, for pure chemistry models, this length is unnecessary and undesirable. For example, since the speed of a CRAY Y-MP computer reaches 90% of its maximum when the inner loop is 512 units, we save array space by dividing the *grid-domain* (entire grid) into *grid-blocks* of 500 or so cells.

Furthermore, by limiting the size of each grid-block, we reduce excess calculations. The reason is that, in order to vectorize around grid-cells, we perform the same operations in every cell; thus, the number of operations in a block is limited by the number of operations in the grid-cell containing the stiffest ODEs. Consequently, the smaller a grid-block, the fewer the number of cells forced to iterate at the same pace as the stiffest cell in the block.

To perform the same operations throughout a grid-block, we need to ensure that the time-step and the order of the integration method (different from the order of the matrix) are the same in each cell within the block. To obtain the same time-step and order, we first calculate the time-step required in each cell of a block for (a) the current order, (b) one order lower than the current order, and (c) one order higher than the current order. Next, for each of the three orders, we choose among the time-steps of all the cells in the block for the shortest. Finally, in the same manner as Gear, we choose among the three pairs of shortest steps and corresponding orders for the longest of these steps and for the order allowing the longest step. Thus, the time-step and order of all cells in a block will be identical and limited by the grid-cell with the stiffest equations.

If we include horizontal and vertical transport implicitly in the ODEs, then we still vectorize around grid-cells. However, in such cases, we need to solve all equations, simultaneously, over all grid-cells; thus, the grid-domain can have only one block.

The speed-up from vectorizing loops around the grid-cell dimension is large. For example, on the CRAY-90 computer, the speed of SMVGEAR in vectorized form is, on average, a factor of 120 greater than its speed in scalar form. Individual subroutines have exhibited speed-up factors of 220 or more. To the contrary, the same code vectorized around the species dimension can be up to forty times slower than the code vectorized around the grid dimension.

2.2. *JSPARSE: a package for evaluating sparse Jacobian matrices*

The second major adjustment we made to Gear's original code was to build into SMVGEAR an algorithm (JSPARSE) that reduces the number of matrix calculations yet still vectorizes every matrix calculation around the grid-cell dimension. JSPARSE reduces the number of matrix calculations in several ways. First, it reorders species so that those with the fewest combined production and loss terms appear first in order and those with the most appear last. This ordering allows ODEs with the most terms to reside in the bottom rows of the matrix and ODEs with the fewest terms to reside in the top rows, maximizing the number of matrix multiplications that include a zero term (e.g. Zlatev, 1991, p. 111). Second, JSPARSE eliminates all calculations in which a zero, known in advance, multiplies another number. To determine, in advance, every occurrence of a zero multiplication, the code, at the beginning of the program, runs through a practice matrix decomposition/back-substitution, and sets new arrays eliminating all calculations in which a zero occurs.

Furthermore, since our atmospheric model contemplates running both gas- and aqueous-phase and day and night chemistry, we form different sparse-matrix arrays for each of four chemistry cases: (a) gas-phase day, (b) gas-phase night, (c) aqueous-phase day, and (d) aqueous-phase night. We designed the program to include any number of additional processes as well. Separating day from night chemistry reduces the number of matrix calculations since photo-dissociation equations do not operate at night.

To further reduce the number of matrix calculations, we removed partial pivoting from the decomposition process. The original reason to remove partial pivoting was to maintain the same sequence of calculations in each grid-cell. If we included partial pivoting, then the order of calculations in each grid-cell would differ, and we could not vectorize around the grid-cell dimension. Fortunately, the use of partial pivoting is unnecessary, as confirmed by Sherman and Hindmarsh (1980, p. 195). They state, "... experience has shown that pivoting is only rarely required in certain applications, and, in any case, if the factorization process should fail, a change in the step-size h will usually improve matters because of the form of P." Based on tests of SMVGEAR to date, removing the pivot has had little or no effect on stability or accuracy.

JSPARSE minimizes array space in several ways. First, it pre-determines how many matrix positions will fill in as a result of matrix decomposition and back-substitution. This value is the sparse-matrix array length for a given set of ODEs. For example, in a case of smog chemistry, with 92 species and 222 chemical reactions, the order of the original matrix was 92; thus its dimension was $92 \times 92 = 8464$. However, the initial matrix was sparse, with only 695 positions filled for day chemistry and 671 for night chemistry. As matrix calculations proceeded, JSPARSE filled-in about 150 more matrix positions in each case.

The number of fill-ins depends almost entirely on the ordering of the ODEs in the matrix. For example,

Table 1. Reduction in array space and number of matrix operations resulting from implementing sparse-matrix techniques. We show three cases—a smog-chemistry, a stratospheric-chemistry, and an aqueous-chemistry case. The first row shows the order of the matrix, the second and third rows show the number and percent of initial matrix positions filled, the fourth and fifth rows show the number and percent of final matrix positions filled, and the last four rows show the number of operations in each of four loops occurring during matrix decomposition/back-substitution. *Decomp.* 1 and 2 refer to the first and second loops occurring during each decomposition, while *Back-sub.* 1 and 2 refer to the first and second loops occurring during each back-substitution call. The first column in each chemistry case shows values without sparse-matrix reductions, and the second two columns in each case show values with the reductions, for each day and night chemistry

	Gas-phase smog			Gas-phase stratosphere			Aqueous-phase		
	Initially	After sparse reductions		Initially	After sparse reductions		Initially	After sparse reductions	
		Day	Night		Day	Night		Day	Night
Order of matrix	92	92	92	39	39	39	37	37	37
No. init. matrix spots used	8464	695	671	1521	287	268	1369	273	270
% of initial spots used	100	8.2	7.9	100	18.9	17.6	100	19.9	19.7
No. final matrix spots used	8464	839	815	1521	334	305	1369	361	351
% of final spots used	100	9.9	9.6	100	22.0	20.0	100	26.4	25.6
No. operations Decomp. 1	255,346	1757	1642	19,019	729	595	16,206	951	881
No. operations Decomp. 2	4560	403	383	741	141	122	666	178	171
No. operations Back-sub. 1	4560	403	383	741	141	122	666	178	171
No. operations Back-sub. 2	4560	344	340	741	154	144	666	146	143

in the smog-chemistry case the ODEs originally had no specific ordering. Consequently, the final matrix contained 6255 non-zero terms for day and 6253 non-zero terms for night chemistry. By re-ordering the species (thus the ODEs) so that those with the most combined production and loss terms appeared last in the matrix, we reduced the final number of non-zero terms to 839/815.

The LU-decomposition/back-substitution process we use originated from Press *et al.* (1992). However, we removed partial pivoting (as discussed earlier) and made other adjustments. As a result, each decomposition process requires operations in two major loops (one much longer than the other), and each back-substitution requires operations in two major loops. For each of the four loops, we know in advance the number of operations that occur within; thus, setting up economical arrays to direct traffic within the loops is straightforward.

In summary, we set all sparse-matrix arrays at the beginning of the program, which allows the program to save time even if the matrix is dense. Because the arrays stay unchanged and no pivoting occurs, we easily vectorize the matrix decomposition/back-substitution process around the grid-cell dimension. Also, since JSPARSE minimizes the number of matrix positions that originate with a zero value but eventually fill-in with a non-zero value, it minimizes the total number of required matrix array positions. Table 1 shows how JSPARSE reduced the number of required array positions and the number of calculations within each of four matrix loops, for each a smog-, a stratospheric-, and an aqueous-chemistry system.

Table 1 shows that, even without vectorization, JSPARSE significantly reduced the time required in each of the four matrix loop operations. For example, JSPARSE reduced the number of operations in the first loop of the decomposition process by a factor of 145 in the larger, smog-chemistry case and factors of 26 and 17 for the smaller stratospheric- and aqueous-chemistry cases, respectively. Initially, about 8.2% of all matrix positions were filled in the smog-chemistry case compared to 18.8 and 19.9% for the other two cases. Naturally, the fewer initial spots filled, the greater the reduction in the number of matrix operations.

Finally, Table 1 shows that the ratio of calculations in the decomposition process compared to those in the back-substitution process was approximately 3:1 in all sparse-matrix cases. Since SMVGEAR typically calls the back-substitution process between 3 and 12 times for each call to the decomposition process, SMVGEAR usually performs comparatively more total calculations in the back-substitution process.

3. COMPARISON OF SMVGEAR TO LSODES AND MIE

To measure the efficiency of SMVGEAR, we compared it to another sparse-matrix Gear-code called LSODES (Livermore Solver for Ordinary Differential

Equations with Sparse Matrices), and to a code that we built prior to SMVGEAR, called MIE (Multistep Implicit–Explicit solver). LSODES, written jointly by A. C. Hindmarsh and A. H. Sherman, is part of ODEPACK (Hindmarsh, 1983), a collection of ODE solver software available through NETLIB, a public-domain software collection. LSODES originated as GEARS (Spellman and Hindmarsh, 1975; Sherman and Hindmarsh, 1980), and the version of LSODES we obtained was last updated in 1987.

3.1. *LSODES*

LSODES and SMVGEAR have several similarities. First, the driver routines of both codes are modifications of an original program of Gear (1971). Second, both codes set up sparse-matrix arrays at the beginning of the program. Third, both codes re-order the ODEs in the matrix to reduce fill-in and eliminate all calculations involving multiplication by zero. Fourth, neither code uses partial pivoting during the matrix computations.

On the other hand, the codes currently differ in several areas. First, while SMVGEAR vectorizes around the grid-cell dimension, LSODES vectorizes around the species and other dimensions. By restructuring all arrays, one could probably vectorize LSODES around grid-cells. Second, SMVGEAR sums up several terms at a time within matrix and other loops to further improve vectorization and reduce the number of overhead calculations.

Third, while one can adjust both codes for any ODE application, SMVGEAR includes changeable input data sets for atmospheric chemistry problems and routines that automatically determine first and partial derivatives. Thus, to change the chemical problem, one needs only to change the species and equations in the input data set. On the other hand, to run LSODES as provided, a user needs to write a routine for evaluating the first derivative. LSODES can internally calculate partial derivatives.

Fourth, SMVGEAR contains arrays that permit it to solve any number of ODE systems during the same model run. For example, it can solve gas-phase chemistry separately from aqueous-phase chemistry during the same run while minimizing array requirements. Furthermore, it contains different arrays for day and night chemistry. To run two or more processes in the same model with LSODES, one would have to either restructure all arrays or implement two or more full versions of LSODES in the model.

Fifth, SMVGEAR contains one common-block while LSODES includes COMMON statements in every subroutine. Thus, while the subroutines in LSODES are more modular, one can make changes in SMVGEAR more easily. Sixth, aside from the set-up routines, SMVGEAR has only five subroutines—(a) a driver routine that calculates time-steps, method orders, and final concentrations; (b) a routine that calculates partial derivatives; (c) a routine that performs matrix decomposition; (d) a routine that performs back-substitutions; and (e) a routine that evaluates first derivatives. LSODES, on the other hand, calls about 13 subroutines during the solution-phase of the program.

For the most part, other differences are minor. We want to acknowledge, however, that we borrowed three ideas from LSODES. These include LSODES' method of (a) predicting the first time-step, (b) setting the error weight, and (c) determining convergence of the corrector iteration. For example, LSODES uses a relative and absolute error tolerance to determine convergence in a slightly different manner from Gear (1971), whose convergence test we previously used. LSODES calculates an error weight for each species i at time t, as

$$W_{E,i}^{t} = E_{\mathrm{R}} C_i^t + E_{\mathrm{A}} \tag{5}$$

where C_i^t is the concentration of species i at time t, E_{R} is a relative error tolerance, and E_{A} an absolute error tolerance. LSODES then computes the root-mean-square norm over all species' errors divided by their error weights and uses the norm to determine convergence.

3.2. *MIE*

MIE, on the other hand, is not a Gear-type code. In this section, we describe the algorithm. In sum, to obtain results from MIE, we estimate species concentrations by iterating over a backward-Euler formula, and use the estimates in a forward-Euler to obtain final concentrations for a time-step. The step-size for the MIE method can vary, but currently it is not determined so elaborately as in Gear-type codes. Instead, we usually start with a fixed time-step, gradually increase the step, but reduce it if convergence becomes difficult. There is no reason why, however, we cannot predict a time-step for the MIE code. After a series of time-steps, we complete a time-interval, which has a pre-determined length. At the end of the interval, we go on to another process in the model and then return to MIE for another interval.

Here, we describe the iteration sequences of the MIE algorithm for a single time-step. In the following notation C is concentration, in units of molecules per cubic centimeter of air for gas-phase and moles per liter of water for aqueous-phase species. The superscripts t and $t+1$ indicate value at the beginning and end, respectively, of a time-step. The superscripts *est*, *MAX*, and m, respectively, indicate an estimated value, a maximum estimated value, and an iteration number. Finally, subscripts i and j identify species numbers. Thus, $C_i^{est,m}$ is the mth estimated concentration of species i, and $C_i^{est,1}$ is the first estimated concentration of i. One can solve for each species' final concentration, C_i^{t+1}, during a time-step in the following order:

(i) Set the first and the maximum estimated concentrations of each species to its initial concentration.

$$C_i^{est,1} = C_i^t \tag{6}$$

$$C_i^{MAX,1} = C_i^t. \tag{7}$$

(ii) Iterate over equations (8)–(17). The first step in the sequence is to calculate reaction rates with estimated concentrations. Each rate describes a uni-, bi- (equation (8)), or ter-molecular kinetic reaction, or a photodissociation process (equation (9)), such as

$$R_n^{est,m} = K_n C_i^{est,m} C_j^{est,m} \tag{8}$$

and

$$R_n^{est,m} = J_n C_i^{est,m} \tag{9}$$

where the subscript n is the reaction or photo process number and $R_n^{est,m}$ is the mth estimated overall rate of a reaction (e.g. No. $cm^{-3} s^{-1}$ for gas-phase). Also, K_n is the kinetic reaction rate coefficient (e.g. s^{-1}, cm^3 No.$^{-1}$ s^{-1}, or cm^6 No.$^{-2}$ s^{-1} for gas-phase) and J_n is the coefficient of a photodissociation process (s^{-1}). If the rate coefficient is pressure-dependent, temperature-dependent, or empirical, then we calculate it at the beginning of a time-interval and change it only before the next interval.

(iii) Next, sum up the mth estimated explicit production rate, $P_i^{est,m}$, and explicit loss rate, $L_i^{est,m}$ (e.g. No. $cm^{-3} s^{-1}$ for gas-phase), and calculate the implicit loss coefficient, $l_i^{est,m}$ (s^{-1}), of each species. Below, the subscript $n(p)$ is reaction rate n of the pth production term, and $n(l)$ is rate n of the lth loss term. The production and loss terms are

$$P_i^{est,m} = \sum_{p=1}^{\text{No. prods}} R_{n(p)}^{est,m} \tag{10}$$

$$L_i^{est,m} = \sum_{l=1}^{\text{No. losses}} R_{n(l)}^{est,m} \tag{11}$$

and

$$l_i^{est,m} = L_i^{est,m} / C_i^{est,m}. \tag{12}$$

(iv) With the mth implicit loss and explicit production terms, calculate the $m+1$th estimated concentration of each species with a backward Euler formula (Curtiss and Hirschfelder, 1952). The backward Euler is implicit (Rosenbaum, 1976; Shampine and Gear, 1979) and can be written as

$$C_i^{est,m+1} = [C_i^t + P_i^{est,m} \Delta t] / [1 + l_i^{est,m} \Delta t] \tag{13}$$

where Δt is the time-step, in seconds. Next, use the estimated concentrations to calculate production and loss terms during the subsequent iteration. Equation (13) always yields a positive concentration and contains no exponential terms, minimizing computer time.

In the special case where a species has no production, equation (13a) describes its final concentration. If we calculate estimates with equation (13b) instead of with equation (13) in such cases, the explicit concentration of the species will approach the solution from equation (13a). Equation (13b) gives a slightly more accurate solution than (13) when a species rapidly decays.

$$C_i^{t+1} = C_i^t e^{-l_i^{est,m} \Delta t} \tag{13a}$$

$$C_i^{est,m+1} = C_i^t (1 - e^{-l_i^{est,m} \Delta t}) / l_i^{est,m} \Delta t. \tag{13b}$$

(v) Calculate tentative final concentration of each species explicitly with the forward-Euler formula, written as

$$C_i^{t+1} = C_i^t + [P_i^{est,m} - L_i^{est,m}] \Delta t. \tag{14}$$

(vi) Test whether the system has reasonably converged for the time-step. At some point during the iteration sequence, concentrations calculated from equation (14) should exceed zero and converge. The more iterations the code takes after all explicit concentrations exceed zero, the closer each concentration approaches the converged solution. We define N_P as a number of iterations we instruct the code to take, during which all values from equation (14) exceed zero. When all explicit concentrations exceed zero during a total of N_P iterations, the iteration sequence ends.

If $C_i^{t+1} \geq 0$ for every species i for N_P iterations, then stop iterating and set each final concentration to C_i^{t+1}. If $C_i^{t+1} < 0$ for at least one species, iterate more. (15)

At first, one might think the above convergence criteria is risky. However, to date, it has worked well. For a large set of equations, the criteria works when $N_P = 1$ because MIE usually takes 3–30 iterations before it dampens heavy oscillations and before every explicit concentration exceeds zero during a time-step. Thus, by the time all explicit values exceed zero, most, but not all, have converged.

Furthermore, when using the MIE code, we often take a series of time-steps during a time-interval. At the end of the interval, we go to another process in the model, and come back to MIE for its next interval. Because solutions to first-order rate equations tend to dampen over time, the implicit and, therefore, explicit solutions from equations (13) and (14), respectively, converge from time-step to time-step. Since we treat the implicit loss-coefficient (equation (12)) of second-order, self-reactions, as if they were first-order reactions, these equations also dampen. Thus, because errors tend to dampen each time-step and because we want an accurate solution only by the end of an interval, we need to fully converge concentrations only at the end of an interval.

For small sets of equations we set $N_P > 1$ during every time-step because all explicit solutions may exceed zero before MIE has dampened oscillations. By setting $N_P > 1$, we allow iterations after all explicit values exceed zero, improving the solution.

If convergence does not occur after a maximum number of iterations permitted (which we set to between 100 and 300), then we reduce the time-step. Typically, MIE iterates between three and 30 times per time-step.

(vii) The last iteration step is to "cap" the implicit estimate to prevent some from exploding to infinity upon iteration. A cap can also significantly decrease the number of iterations needed for convergence. The choice of a cap is not unique. Some include holding

the maximum estimate to the larger(est) of the initial concentration and (a) the first estimate, (b) any of the first four estimates, or (c) the value of the previous estimate before being limited by the cap. We use (c). The equations describing cap (c) are

$$C_i^{MAX,m+1} = \mathrm{MAX}\{C_i^{est,m+1}, C_i^t\} \quad (16)$$

and

$$C_i^{est,m+1} = \mathrm{MIN}\{C_i^{est,m+1}, C_i^{MAX,m}\}. \quad (17)$$

(viii) In this scheme, iterated explicit concentrations from equation (14) and iterated implicit concentrations from equation (13) converge to the same value. However, if a species is short-lived ($l_i^{est,m}\Delta t \gg 1$), its implicit concentration converges faster. Thus, a way to reduce computer time is to calculate final concentrations of short-lived species with equation (13) instead of (14). However, calculating too many final concentrations implicitly causes mass loss over time. A way to reduce mass loss is to solve only a few final concentrations implicitly. To do so, we define LT as a unitless number to compare $l_i^{est,m}\Delta t$ against. If $l_i^{est,m}\Delta t \geqslant LT$ for a species, then we set the species' final concentration to its last estimate from equation (13). Thus

$$\text{if} \quad l_i^{est,m}\Delta t \geqslant LT \text{ then } C_i^{t+1} = C_i^{est,last}. \quad (18)$$

We typically set LT and N_P at the start of a time-interval and re-set them for the last step of the interval. For example, at the start, we often set $N_P = 1$ and $LT = 10^6$. For the last step, we either reduce LT to 10^2, increase N_P to between 20 and 250, or both. Generally, to prevent accumulation of error over time, we want to force a sufficient number of iterations during each time-step. Setting $LT = 10^6$ or more for most steps usually forces sufficient iterations.

Finally, when $l_i^{est,m}\Delta t \geqslant LT$, a species' explicit concentration does not need to exceed zero because we use its implicit concentration as the final value. Thus, to speed up the code we can replace (15) with (15m).

If $C_i^{t+1} \geqslant 0$ or $l_i^{est,m}\Delta t \geqslant LT1$ for each species i for N_P iterations, then stop iterating and set final concentrations to C_i^{t+1} when $l_i^{est,m}\Delta t < LT1$ or to $C_i^{est,last}$ when $l_i^{est,m}\Delta t \geqslant LT$. If $C_i^{t+1} < 0$ for at least one species, continue iterating. (15m)

Above, $LT1$ is any number greater than or equal to LT.

In summary, the MIE code solves ODEs by estimating species concentrations with an iterated backward-Euler formula and using the estimates to determine rates with a forward-Euler. While the forward-Euler predicts the final concentration of most species, the backward-Euler predicts the final concentration of the shortest-lived species. We have applied MIE to atmospheric chemistry problems only; however, there is no reason why MIE cannot solve ODEs for other processes.

3.3. *Timing tests*

We tested SMVGEAR against both LSODES and MIE for three cases: two smog chemistry and a stratospheric chemistry case. In the next section, we show additional timings of SMVGEAR solving both gas- and aqueous-chemistry in a full model.

The smog mechanism we tested was a combination of inorganic equations compiled from DeMore *et al.*, 1990, organic equations from the Carbon-Bond IV expanded mechanism (Gery *et al.*, 1988), sulfur equations from Toon *et al.* (1987), and additional reactions for a total of 92 gas-phase species or lumped bond groups, 200 chemical reactions, and 22 photodissociation processes. Of the 200 kinetic rates, we calculated several as three-body pressure-dependent reactions.

For the first smog simulation (SMOG11), we initialized 14 species with concentrations of 5.0×10^{11} cm^{-3}, and most other active species with zero concentration. However, we set the initial concentrations of major atmospheric constituents, such as oxygen, nitrogen, water, carbon monoxide, and methane, at realistic levels. For the second simulation (SMOG12), we increased the initial concentration of 14 species from 5×10^{11} to 5×10^{12} cm^{-3} to increase the stiffness of the system.

For the stratospheric test (STRAT), we used a set of 39 species, 84 chemical reactions, and 24 photodissociation processes compiled primarily by S. Elliott and R.P. Turco (personal communication). We assumed an altitude of 20 km (pressure of 55.3 mb), a constant temperature of 217 K, and fairly realistic initial concentrations.

For both the smog and stratospheric tests, we changed the photorates every time-step, no matter how small. To simplify the tests, we varied the photorates with a sine function during 12 of every 24 h and turned the photorates off every other 12 h. Thus, the only process occurring during the simulations shown in this section was chemistry with photorates continuously changing during each day.

Furthermore, we fixed the relative error tolerance of both SMVGEAR and LSODES to 10^{-3} and the absolute error tolerance of each species to 10^3. Both codes predicted their own time-steps. For the MIE code, we initialized the time-step at 10 s and allowed the step-size to increase by a factor of two every 30 steps. After every 1800-s interval, we re-set the time-step to 10 s. Also, for MIE, we set $LT = 10^6$ and $N_P = 1$ at the beginning of each interval and changed $LT = 10^2$ and $N_P = 20$ for the last time-step of each interval.

Before running speed comparisons among the codes, we determined error resulting from each code's parameter values. We calculated the average absolute-value concentration error over time as

$$ERROR = \frac{100\%}{N_I} \sum_{j=1}^{N_I} \left\{ \frac{1}{N_{S,j}} \sum_{i=1}^{N_{S,j}} \mathrm{ABS}\left(\frac{C_{p,i}^{t(j)} - C_{e,i}^{t(j)}}{C_{e,i}^{t(j)}} \right) \right\} \quad (19)$$

Table 2. Average absolute-value percent error (as determined by equation (19)) in concentration for SMVGEAR and MIE resulting from the choice of parameter values and/or error tolerances. In each case, SMOG11, SMOG12, and STRAT, we averaged errors over all 1800-s time-invervals during the given simulation period. The separate columns under SMVGEAR and MIE indicate the lowest concentrations we included in the error calculation

		SMVGEAR		MIE	
	Simulation period (d)	Include concs $>10^1$ cm^{-3} (%)	Include concs $>10^{-5}$ cm^{-3} (%)	Include concs $>10^1$ cm^{-3} (%)	Include concs $>10^{-5}$ cm^{-3} (%)
SMOG11	1	0.55	1.35	1.44	2.84
SMOG12	1	0.46	1.08	1.42	2.61
STRAT	9	0.25	0.51	1.20	1.42

where N_I is the number of time-intervals during a model run, $N_{S,j}$ is the number of species whose concentration are above a minimum value after the jth time interval, $C^{t(j)}$ represents a concentration at time t, corresponding to the end of the jth interval, and the subscripts p, i and e, i refer to predicted and exact concentrations, respectively, of species i. In all cases, we used SMVGEAR with a stiff error tolerance to obtain "exact" solutions after each time-interval.

Table 2 shows the time- and species-averaged errors from MIE and SMVGEAR. For SMVGEAR, we tested the effect of assuming a relative error tolerance of 10^{-3} and an absolute tolerance of 10^3 instead of more stringent tolerances. For MIE, we tested the effect of using the values for the time-steps, LT and N_P, given above. Since we used LSODES with the same error tolerances as SMVGEAR, we assumed it gave errors similar to those of SMVGEAR. For these tests, we ran simulations of either one or nine days. For both SMVGEAR and MIE, we divided the simulation period into intervals of 1800 s in order to obtain the error at the end of these intervals. The results from Table 2 indicates that SMVGEAR gave lower errors than MIE in all cases.

Finally, we tested the three codes running in a one grid-cell model and tested SMVGEAR and MIE in a 500 grid-cell model. Because the chemistry in each grid-cell was the same and LSODES does not vectorize around grid-cells, LSODES takes the same time per grid-cell to solve in 10,000 or 100,000 cells as it does to solve in 1. Thus, to obtain LSODES results for many grid-cells, we multiplied its time to solve in one cell by the total number of cells in the domain. Similarly, because we limited the block-size of SMVGEAR and MIE to 500 cells in these examples, we extrapolated its time to solve in 500 cells to its time to solve in the entire domain.

For the two smog tests, we simulated chemistry for 3-day periods. For the stratospheric test, we simulated for a 30-day period. During these tests, only MIE divided the simulation period into intervals of 1800 s. The other two codes integrated continuously.

Table 3 shows results from the three simulations and characteristics of the different codes. First, in large grid-domains, SMVGEAR was about 60-times faster than LSODES and 30–55-times faster than MIE. While some of SMVGEAR's speed-up compared to LSODES resulted from fewer matrix calculations, most resulted from vectorization. Because MIE vectorizes around grid-cells, it, too, runs faster over large domains than it does over small domains.

Second, in one grid-cell, LSODES was fastest because it vectorizes around the species dimension while both SMVGEAR and MIE vectorize around the grid-cell dimension. In fact, because SMVGEAR vectorizes around grid-cells, every inner loop is inefficient in a one grid-cell model. However, one can make SMVGEAR more efficient in one grid-cell simply by removing all the inner grid-cell loops.

Although SMVGEAR was slower in one grid-cell, it performed fewer matrix decomposition and back-substitution calculations, in all simulations, than did LSODES. Thus, the greater speed of LSODES in one grid-cell was attributable to its type of vectorization. In fact, row (e) of Table 3 shows that, if we remove the effects of vectorization, SMVGEAR performed slightly faster than LSODES. However, the number of matrix fill-ins in each case was similar for the two codes. Meanwhile, the MIE code was far slower than either of the other two codes in a one grid-cell model.

In sum, when we ran SMVGEAR over a grid-domain, and the only process was gas-phase chemistry with diurnally changing photorates, SMVGEAR was very fast on a vectorized machine compared to the other two codes tested. Furthermore, without vectorization, its time was slightly better than LSODES, one of the fastest available ODE solvers. From Table 3, we calculate that SMVGEAR required about 1.4 min of simulation time per day of smog-chemistry simulation over 10,000 grid-cells and 3.4 min per day of stratospheric-chemistry simulation over 100,000 cells. In the next section, we discuss the timing of SMVGEAR in a model containing other processes.

4. APPLICATION OF SMVGEAR

In this section, we report the speed of SMVGEAR in two other atmospheric applications. Because

Table 3. Statistics from two smog-chemistry and one stratospheric-chemistry comparisons on a CRAY-90 computer. The smog-chemistry tests, described in the text, were for 3 simulation days, and the stratospheric test was for 30 simulation days. For smog-chemistry, we tested the three solvers over both 10,000 grid-cells and 1 cell, while for the stratospheric case, we tested over 100,000 cells and 1 cell. Row (e) in each case normalizes the time of each code to a common speed of one megaflop since, with different numbers of grid-cells, each code vectorizes differently. For all three simulations, SMVGEAR and LSODES used the same error tolerances and a similar initial time-step

	SMVGEAR	LSODES	MIE
SMOG11 CHEMISTRY			
(a) CPU time to solve 3 days, 10,000 cells	3.9 min	4.08 h	2.16 h
(b) Megaflops for 3 days, 10,000 cells	342	10.9	341
(c) CPU time to solve 3 days, 1 cell	2.92 s	1.47 s	43.2 s
(d) Megaflops for 3 days, 1 cell	3.1	10.9	4.7
(e) CPU time 3 days, 1 cell, at 1 megaflop	9.1 s	16.0 s	203 s
(f) Vectorization speed-up — (d) × 10,000/(a)	124.8	1	55.6
(g) Number of time-steps taken, 1 cell	808	876	10,296
(h) Number of matrix decompositions, 1 cell	211	274	—
(i) Number of matrix back-substitutions, 1 cell	1394	1738	—
(j) Matrix size after fill-ins, 1 cell (day/night)	839/815	839	—
SMOG12 CHEMISTRY			
(a) CPU time to solve 3 days, 10,000 cells	4.24 min	4.58 h	3.86 h
(b) Megaflops for 3 days, 10,000 cells	342	10.97	340.5
(c) CPU time to solve 3 days, 1 cell	3.31 s	1.65 s	62.4 s
(d) Megaflops for 3 days, 1 cell	3.07	10.97	4.7
(e) CPU time 3 days, 1 cell, at 1 megaflop	10.2 s	18.1 s	293 s
(f) Vectorization speed-up — (d) × 10,000/(a)	130.1	1	44.9
(g) Number of time-steps taken, 1 cell	915	932	10,472
(h) Number of matrix decompositions, 1 cell	247	317	—
(i) Number of matrix back-substitutions, 1 cell	1531	1925	—
(j) Matrix size after fill-ins, 1 cell (day/night)	839/815	839	—
STRATOSPHERIC CHEMISTRY			
(a) CPU time to solve 30 days, 100,000 cells	1.7 h	115 h	51.1 h
(b) Megaflops for 30 days, 100,000 cells	346.7	9.83	342
(c) CPU time to solve 30 days, 1 cell	7.1 s	4.14 s	88.4 s
(d) Megaflops for 30 days, 1 cell	3.61	9.83	4.6
(e) CPU time 30 days, 1 cell, at 1 megaflop	25.6 s	40.7 s	293 s
(f) Vectorization speed-up — (d) × 100,000/(a)	116.0	1	48
(g) Number of time-steps taken, 1 cell	5188	4548	102,960
(h) Number of matrix decompositions, 1 cell	1743	2044	—
(i) Number of matrix back-substitutions, 1 cell	9008	9776	—
(j) Matrix size after fill-ins, 1 cell (day/night)	334/305	344	—

SMVGEAR solves ordinary differential equations, we can include advection, diffusion, emissions, deposition, aerosol growth and evaporation, and many other physical processes as terms in the equations. Furthermore, if we include advection and diffusion terms in the ODEs, we can still vectorize around grid-cells; but, we need to solve the equations in every grid-cell simultaneously. For a purely gas-phase atmospheric model, solving every process at the same time, using SMVGEAR, appears feasible, depending on the speed and memory abilities of the computer.

However, when we add multicomponent, size-resolved aerosols to the model, solving all processes simultaneously becomes a formidable task. For example, our air pollution model contains on the order of 70 aerosol species resolved into 40 or more size bins, and 92 gases. Thus, for a 10,000-cell domain we would need to solve almost three million ordinary differential equations simultaneously if we wanted SMVGEAR to take responsibility for the entire model. Furthermore, some aerosol processes, such as multicomponent coagulation, require significant matrix fill-in.

Consequently, a more feasible approach to comprehensive atmospheric modeling is time-splitting different processes. A disadvantage of time-splitting is that, if we solve some processes with SMVGEAR, we need to re-start calculations in SMVGEAR every time-interval under conditions that often differ significantly from the conditions at the end of the previous interval. Because of the abrupt transition, SMVGEAR will often restart with a time-step and order smaller than those used at the end of the previous interval and the overall time spent in SMVGEAR will increase.

To test the speed of SMVGEAR in a time-split model, we ran a 24-h simulation in a 10,000 grid-cell domain (40 latitudinal cells × 50 longitudinal cells × 5 vertical layers), which we divided into blocks of 500 grid-cells. The model included gas-phase smog-chemistry, gridded emissions, horizontal advection, vertical diffusion, and dry deposition. The resolution of each horizontal grid-cell was about 4.5 × 5 km, and the

emissions data, provided by the California Air Resources Board, included emissions rates for 17 gases and lumped carbon-bond groups that differed in each surface grid-cell and for each hour. We added the emissions rates as terms into the ODEs of SMVGEAR. Because emissions rates during a given time-step are constant, we added the terms to the right-hand side of equation (4) but not to the Jacobian matrix, J.

Every SMVGEAR time-step, we interpolated the emissions rate in each grid-cell of each species between the rate for the current hour and that for either the hour ahead or behind. The reason we interpolated the emissions was to create a smooth profile. No matter how small the time-step, we performed this interpolation. The time taken to recalculate emissions every time-step was *de minimus* compared to the savings in iterations it permitted.

Next, we time-split the advection and remaining vertical transport terms. The advection code we used originated from Toon *et al.* (1988), and the vertical transport code originated from Turco *et al.* (1979a,b) and Toon *et al.* (*ibid.*). During the 24-h simulation, we solved the horizontal and vertical transport using 300-s intervals, and we solved the 92 chemical ODEs (with continuously changing photorates and emissions) using 1800-s intervals. In other words, every 1800 s, we stopped the calculations in SMVGEAR, performed six transport steps, and returned to SMVGEAR for another interval.

For the 24-h simulation over 10,000 grid-cells, SMVGEAR took a total of 6.7 min at an average speed of 326 megaflops to perform the chemistry and emissions calculations. We stress that, during this simulation, species concentrations differed widely in each grid-cell. Table 4 displays the speed of the different SMVGEAR processes during this application. The table shows that SMVGEAR performed an average of 3.97 back-substitutions each time it decomposed a matrix. Because the ratio of decomposition to back-substitution operations is about 3:1 (Table 1), we would expect back-substitution to take more time. However, the back-substitution coding is about 1.8-times faster than the decomposition coding; thus, decomposition took more time.

Finally, if all processes in this simulation had achieved 420 megaflops (about the speed of the fastest routine) SMVGEAR would have taken about 5.2 min, or 22%, less time than it actually took. Even better, if some SMVGEAR routines achieved over 600 megaflops, the speed of two other routines in the time-split model, the time to solve would decrease even more. Thus, a future job is to further increase the speed of individual processes in SMVGEAR.

In the second application, we used SMVGEAR to solve both gas- and aqueous-phase chemistry in the same model. Other processes we included during the run were horizontal advection, vertical diffusion, condensational and dissolutional growth, chemical equilibrium, and dry and wet deposition. Furthermore, we resolved the aerosols into 43 size bins. While each size bin contained 50 species for this simulation, 37 of the species were active in aqueous chemical reactions, and almost all of the 50 were included in either gas–aqueous transfer or aqueous–aqueous equilibrium reactions. We obtained most aqueous chemical equations from Pandis and Seinfeld (1989) and Jacob (1986).

Initially, the aerosols contained only solid matter and some water. We then increased the relative humidity to above 100% and allowed the 43 size bins to grow to various fog-size drops. Meanwhile, we initialized the gas-phase with concentrations similar to those in Pandis and Seinfeld (1989), and allowed SMVGEAR to solve gas-phase chemistry. Also, in time-split operations, we calculated the transfer of gases simultaneously to all size bins, calculated chemical equilibrium among aqueous species in each bin, and calculated the forward aqueous-phase chemistry in each bin. We used SMVGEAR to solve the aqueous chemistry and used codes we developed to compute the other processes. For this application, we solved all aerosol processes during 24 3600-s time-intervals, and calculated 12 transport intervals for every chemical and micro-physical interval.

While SMVGEAR performed the calculations in 500-cell grid, we extrapolated the results to 10,000 cells. For gas- and 43 size bins of aqueous-phase chemistry combined, the total time SMVGEAR needed for a 10,000-cell grid and a 24-h simulation period was 3.4 h. The gas-phase chemistry required less than 1.8% of the total time. The average speed of SMVGEAR on the CRAY-90 computer for this application was 348.6 megaflops, and the ratio of back-substitution to matrix decomposition calls was 5.7.

Table 4. Timing and speed of different processes in SMVGEAR during a 24-h simulation over 10,000 grid-cells. During the simulation SMVGEAR solved 92 ODEs that included chemical and emissions terms. Horizontal advection, vertical diffusion, and dry deposition were time-split and solved by other methods. Thus, the timings here are those for the chemical and emissions processes only

	Driver routine	Partial derivative evaluation	Matrix decomposition	Matrix back-substitution	First derivative evaluation	Total time/ average speed
Time speed	100 s	50.7 s	95.1 s	77.1 s	78.7 s	6.7 min
(megaflops)	427	205.7	226.7	414.8	308.5	326

The primary reason SMVGEAR took longer to solve aqueous reactions than gas-phase reactions is that aqueous reactions are much stiffer than gas reactions. A lesser reason is that we solved the aqueous chemistry in 43 size bins, compared to the equivalent of one size bin for the gas-phase.

An important difference between solving gas- or aqueous-phase chemistry with SMVGEAR is setting the absolute error tolerance. While we set similar relative error tolerances with gas and aqueous chemistry, the absolute tolerance depends on typical concentration values. Since we used units of No. cm^{-3} for the gas-phase and mol ℓ^{-1} for the aqueous-phase, the concentration values can differ by more than 30 orders of magnitude. Thus, while we usually set the absolute tolerance to 10^3 cm^{-3} for the gas-phase, we changed it to between 10^{-13} M and 10^{-15} M for the aqueous-phase. If a significant number of aqueous concentrations decay below the latter value, the aqueous error tolerance should be set to even smaller values.

5. CONCLUSION

We have presented a new sparse-matrix, vectorized Gear-type code (SMVGEAR). Because SMVGEAR vectorizes around grid-cells, one can use it on vector machines to solve multidimensional atmospheric problems. For problems in single grid-cells and for problems on non-vector machines, SMVGEAR performs similarly to LSODES, another sparse-matrix Gear-code. Based on CRAY-90 timing tests, SMVGEAR can solve chemistry in 10^2–10^5 grid-cells in good speed, and it is much faster than the MIE (Multistep Implicit–Explicit) method, which is the method we used previously.

To vectorize around the grid-cell dimension, SMVGEAR performs the same operations in each cell of each grid-block, where a grid-block is a chunk of cells within an overall grid-domain. Thus, the time-step and order of SMVGEAR in a grid-block are limited by the grid-cell with the stiffest ODEs. However, the speed-up from vectorizing over grid-blocks is far greater than the loss of speed from performing extra calculations in some cells of a block.

Furthermore, the grid-block concept is ideal for parallel computing. We can send each block to an individual processor, and if the processor is vectorized, computational time can decrease dramatically. For example, if a machine has 32 processors, 4 vector units per processor, and each vector unit achieves 32 megaflops at peak performance. Then, the maximum speed on this computer would be more than 4000 megaflops. Even if the practical speed of the CM-5 is 1000 megaflops, SMVGEAR would compute almost three-times faster than it currently computes on the CRAY-90.

Although SMVGEAR has been optimized, its speed can be improved. For example, some sections of the code do not achieve the same speed as other sections. By working on the loops of the less efficient sections, we can increase the program's speed. Furthermore, we can re-order the grid-cells, each time-step, by putting cells with the stiffest equations together in the same grid-block.

Currently, we use SMVGEAR to solve gas-phase and size bin-resolved aqueous-phase chemistry in an air pollution model that also computes the effects of many other processes. A challenge is to include more terms in the ODEs of SMVGEAR in order to reduce the discontinuities caused by time-splitting. As computer speed and memory increases, solving gas and aerosol processes together with transport in large grids will become more and more practical.

Acknowledgements—We thank Dr Robert Chatfield of NASA Ames Research Center, Mountain View, California, Dr Scott Elliott of the Los Alamos National Laboratory, New Mexico, and Drs Alan Hindmarsh and Doug Rotman of Lawrence Livermore Laboratory for their help. We particularly thank Dr Hindmarsh for providing us with the LSODES code and a wealth of literature relating to ODE solvers. Finally, we thank the Environmental Protection Agency for support under grant CR-812771-03-0, the National Science Foundation for partial support under grant A7M-89-11836, and the NAS computer center at NASA Ames Research Center, Mountain View, California, for permitting our use of a CRAY Y-MP and a CRAY-90 computer.

REFERENCES

Austin J. (1991) On the explicit versus family solution of the fully diurnal photochemical equations of the stratosphere. *J. geophys. Res.* **96,** 12,941–12,974.

Brown P. N. and Hindmarsh A. C. (1986) Matrix-free methods for stiff systems of ODEs. *SIAM J. numer. Anal.* **23,** 610–638.

Brasseur G. and Solomon S. (1984) *Aeronomy of the Middle Atmosphere.* D. Reidel, Dordrecht.

Byrne G. D. (1990) Pragmatic experiments with Krylov methods in the stiff ODE setting. Presented at the IMA Conference on Computational Ordinary Differential Equations, Imperial College of Science and Technology, London, 3–7 July, 1989.

Byrne G. D. and Hindmarsh A. C. (1976) EPISODEB: an experimental package for the integration of systems of ordinary differential equations with banded Jacobians. Lawrence Livermore Laboratory Report UCID-30132.

Byrne G. D., Hindmarsh A. C., Jackson K. R. and Brown H. G. (1977) A comparison of two ODE codes: GEAR and EPISODE. *Comput. chem. Engng* **1,** 133–147.

Crutzen P. J. (1971) Ozone production rates in an oxygen–hydrogen–nitrogen oxide atmosphere. *J. geophys. Res.* **76,** 7311–7327.

Curtiss C. F. and Hirschfelder J. O. (1952) Integration of stiff equations. *Proc. natn. Acad. Sci. U.S.A.* **38,** 235–243.

DeMore W. B., Sander S. P., Golden D. M., Molina M. J., Hampson R. F., Kurylo M. J., Howard C. J. and Ravishankara A. R. (1990) Chemical kinetics and photochemical data for use in stratospheric modeling. Evaluation number 9. Rep. 90-1. Jet Propul. Lab., Pasadena, California, U.S.A.

Eisenstat S. C., Gursky M. C., Schultz M. H. and Sherman A. H. (1977a) Yale Sparse Matrix Package: II. The symmetric codes. Research Report No. 112, Dept. of Computer Sciences, Yale University.

Eisenstat S. C., Gursky M. C., Schultz M. H. and Sherman A. H. (1977b) Yale Sparse Matrix Package: II. The nonsymmetric codes. Research Report No. 114, Dept. of Computer Sciences, Yale University.

Elliott S., Turco R. P. and Jacobson M. Z. (1993) Tests on combined projection/forward differencing integration for stiff photochemical family systems at long time step. *Comput. Chem.* **17,** 91–102.

Gear C. W. (1967) The numerical integration of ordinary differential equations. *Math. Comp.* **21,** 146–156.

Gear C. W. (1969) The automatic integration of stiff ordinary differential equations. In *Information Processing* **68,** (edited by Morrel A. J. H.), pp. 187–193. North Holland Publishing Company, Amsterdam.

Gear C. W. (1971) *Numerical Initial Value Problems in Ordinary Differential Equations.* Prentice-Hall, Englewood Cliffs, NJ.

Gery M. W., Whitten G. Z. and Killus J. P. (1988) Development and testing of the CBM-IV for urban and regional modeling. Rep. EPA-600/3-88-012, U. S. Environ. Prot. Agency, Research Triangle Park, N. C.

Hesstvedt E., Hov O. and Isaksen I. S. A. (1978) Quasi-steady-state approximations in air pollution modeling: Comparison of two numerical schemes for oxidant prediction. *Int. J. chem. Kinet.* **10,** 971–994.

Hindmarsh A. C. (1972) Linear multistep methods for ordinary differential equations: method formulations, stability, and the methods of Nordsieck and Gear. Lawrence Livermore Laboratory Report UCRL-51186, Rev. 1.

Hindmarsh A. C. (1974) GEAR: ordinary differential equation system solver. Lawrence Livermore Laboratory Report UCID-30001, Rev. 3.

Hindmarsh A. C. (1975) GEARB: solution of ordinary differential equations having banded Jacobians. Lawrence Livermore Laboratory Report UCID-30059, Rev. 1.

Hindmarsh A. C. (1976) Preliminary documentation of GEARIB: solution of implicit systems of ordinary differential equations with banded Jacobian. Lawrence Livermore Laboratory Report UCID-30130.

Hindmarsh A. C. (1977) GEARB: solution of ordinary differential equations having banded Jacobians. Lawrence Livermore Laboratory Report UCID-30059, Rev. 2.

Hindmarsh A. C. (1980) LSODE and LSODI, two new initial value ordinary differential equation solvers. *ACM Newsl.* **15,** 10–11.

Hindmarsh A. C. (1983) ODEPACK, a systematized collection of ODE solvers. In *Scientific Computing* (edited by Stepleman R. S. *et al.*), pp. 55–74. North-Holland, Amsterdam.

Hunt B. G. (1966) Photochemistry of ozone in a moist atmosphere. *J. geophys. Res.* **71,** 1385–1398.

Jacob D. J. (1986) Chemistry of OH in remote clouds and its role in the production of formic acid and peroxymonosulfonate. *J. geophys. Res.* **91,** 9807–9826.

Jacob D. J., Sillman S., Logan J. A. and Wofsy S. C. (1989) Least independent variables method for simulation of tropospheric ozone. *J. geophys. Res.* **94,** 8497–8509.

McRae G. J., Goodin W. R. and Seinfeld J. H. (1982). Numerical solution of the atmospheric diffusion equation for chemically reacting flows. *J. comp. Phys.* **45,** 1–42.

Morris D. B. and Hindmarsh A. C. (1975) GEARV: a vectorized ordinary differential equations solver. Lawrence Livermore Laboratory Report UCID-30119.

NAS (1992) *NAS User Guide.* Vol. 2, version 6.0, section 4, p. 22. Moffett Field, California.

Pandis S. N. and Seinfeld J. H. (1989) Sensitivity analysis of a chemical mechanism for aqueous-phase atmospheric chemistry. *J. geophys. Res.* **94,** 1105–1126.

Press W. H., Teukolsky S. A., Vetterling W. T. and Flannery B. P. (1992) *Numerical Recipes in Fortran.* Cambridge University Press, New York.

Rosenbaum J. S. (1976) Conservation properties of numerical integration methods for systems of ordinary differential equations. *J. comp. Phys.* **20,** 259–267.

Shampine L. F. and Gear C. W. (1979) A user's view of solving stiff ordinary differential equations. *SIAM Rev.* **21,** 1–17.

Sherman A. H. (1975) Yale Sparse Matrix Package User's Guide. Lawrence Livermore Laboratory Report UCID-30114.

Sherman A. H. and Hindmarsh A. C. (1980) GEARS: a package for the solution of sparse, stiff ordinary differential equations. Lawrence Livermore Laboratory Report UCRL-84102.

Shimazaki T. and Laird A. R. (1970) A model calculation of the diurnal variation in minor neutral constituents in the mesosphere and lower thermosphere including transport effects. *J. geophys. Res.* **75,** 3221–3235.

Spellmann J. W. and Hindmarsh A. C. (1975) GEARS: solution of ordinary differential equations having a sparse Jacobian Matrix. Lawrence Livermore Laboratory Report UCID-3011.

Stoer J. and Bulirsch R. (1980) *Introduction of Numerical Analysis.* Springer, New York.

Toon O. B., Kasting J. F., Turco R. P. and Liu M. S. (1987) The sulfur cycle in the marine atmosphere. *J. geophys. Res.* **92,** 943–963.

Toon O. B., Turco R. P., Westphal D., Malone R. and Liu M. S. (1988) A multidimensional model for aerosols: description of computational analogs. *J. atmos. Sci* **45,** 2123–2143.

Turco R. P. and Whitten R. C. (1974) A comparison of several computational techniques for solving some common aeronomic problems. *J. geophys. Res.* **79,** 3179–3185.

Turco R. P., Hamill P., Toon O. B., Whitten R. C. and Kiang C. S. (1979a) A one-dimensional model describing aerosol formation and evolution in the stratosphere. Part I: physical processes and mathematical analogs. *J. atmos. Sci.* **36,** 699–717.

Turco R. P., Hamill P., Toon O. B., Whitten R. C. and Kiang C. S. (1979b) The NASA-Ames Research Center Stratospheric Aerosol Model: I Physical processes and Computational Analogs. *NASA Tech. Publ. (TP)* 1362, iii–94.

Zlatev Z. (1991) *Computational Methods for General Sparse Matrices.* Kluwer, Dordrecht.