JOURNAL OF COMPUTATIONAL PHYSICS **129,** 101–110 (1996)
ARTICLE NO. 0236

Efficient Implementation of Fully Implicit Methods for Atmospheric Chemical Kinetics

A. Sandu,[1] F. A. Potra,[2] G. R. Carmichael,[3] and V. Damian[4]

Center for Global and Regional Environmental Research, The University of Iowa, Iowa City, Iowa 52246

Received January 22, 1996; revised May 30, 1996

Implicit integrators are very useful in efficiently solving stiff systems of ODEs arising from atmospheric chemistry kinetics, provided that they are modified to take full advantage of the structure of the problem. A systematic way of treating sparsity for reducing the linear algebra cost is presented. © 1996 Academic Press, Inc.

1. INTRODUCTION

It is well known that the equations arising from chemical kinetics comprise a system of stiff ordinary differential equations (ODEs). For solving these equations numerically, implicit integrators with infinite stability regions are likely to work with relatively large step sizes when the accuracy requirements are not too stringent. However, at each integration step, a nonlinear system of equations has to be solved. This involves the repeated evaluation of Jacobians and the solution of linear algebraic systems of dimension n, the number of species considered in the model.

General stiff ODE solvers do not take advantage of the sparsity pattern of the Jacobian, and the number of arithmetic operations required for the numerical solution of the corresponding linear system is proportional to n^3. This is one of the reasons why general stiff ODE solvers are not very efficient for integration of chemical rate equations with a moderate to large number of species. A comparison of the exactness and time efficiency of different integrators can be found in the paper of Shieh *et al.* [18].

On the other hand, exploitation of sparsity may significantly reduce the linear algebra overhead. Recently, several authors showed promising results with sparse BDF codes in atmospheric chemistry models. In [13] Jacobson and Turco describe SMVGEAR—a BDF code that uses vectorization around the grid cell-dimension. Treatment of sparsity is an essential ingredient in decreasing the computational time. In [21] Verwer *et al.* use a sparse version of VODE. Their treatment of sparsity is based on the netlib package SLAP.

In this paper we develop a systematic way of exploiting sparsity when integrating atmospheric chemistry equations. Unlike [13], our target is not a specialized architecture; we concentrate on developing machine-independent algorithms. In section 3 we discuss and evaluate reordering techniques that lead to minimal fill-in during LU decomposition. We then (Section 4.3) test various linear system solvers, in particular showing that the chosen routine is twice as fast as the one used in [21]; finally (Section 5) we demonstrate how the chosen solver can improve the efficiency of some state-of-the-art stiff ODE solvers. These ideas are tested on two comprehensive chemical mechanisms used to study stratospheric and tropospheric chemistry; both models are described in Section 4.6.

2. ABOUT PIVOTING

Implicit algorithms advance the numerical solution of differential equations one step in time by solving a nonlinear system of equations. This is done by using a (modified) Newton method, which results in solving a sequence of linear systems. The numerical solution of the linear system is usually done by a direct method, i.e., by employing a (LU) factorization. The matrix that is to be factorized in implicit solvers (sometimes called "prediction matrix") is of the form

$$P = I - h \cdot \gamma \cdot J,$$

where I is the identity matrix, J is an approximation to the Jacobian, h is the attempted step-size, and γ is a coefficient dependent on the method. This form of the prediction matrix holds for multistep schemes and, after an equivalence transformation, for Runge–Kutta methods as well.

Several authors [6, 13, 21] report good results with non-pivoting sparse linear algebra solvers when integrating atmospheric chemistry equations. This saves computational time. Several arguments sustain this practice:

[1] sandu@cgrer.uiowa.edu.
[2] potra@math.uiowa.edu.
[3] gcarmich@cgrer.uiowa.edu.
[4] vdamian@cgrer.uiowa.edu.

- The presence of I ensures that diagonal elements are not structurally zero;
- If a pivotal element is zero or very small, then the step size is rejected and a new P is constructed, with a smaller h. P is diagonally dominant for all h sufficiently small, say $0 \leq h \leq h_0$, so that, at least in the limit case, no pivoting is required. We should point out here that the restriction on step size needed for diagonal dominance may be as severe as the stability restriction imposed by an explicit method.
- Reordering the species (see below) is equivalent to performing a diagonal pivoting; of course, this does not take into account numerical values, but it helps, because a multiplier with absolute value greater than one will lead to an error amplification, and this error will corrupt each row processed at that stage. Since the columns with less elements come first, fewer row operations are needed in the initial stages; hence the error introduced by a very small pivot is not greatly amplified.
- A solution of the linear system corrupted by numerical errors (due to nonpivoting) may be thought of as the exact solution of a system with inexact Jacobian. The convergence of Newton iterations may not be affected by this approximation. This argument is, of course, not valid with Rosenbrock methods.

Theoretically, because of nonpivoting, the matrix P may be falsely "detected" as singular, and unnecessary step-size rejections may occur. Moreover, even if the step size is accepted, more iterations per Newton step may be needed because of the increased errors in the solution of linear systems. Numerical experience (ours and that of the authors cited earlier) shows that the above phenomena are not that important in practice.

3. SPECIES ORDERING

The sparsity structure of the matrix P is given by the sparsity structure of the Jacobian J, which in its turn is determined by the chemical interactions. Thus, for a given chemical system, the sparsity structure is constant and can be predetermined. More exactly, a maximal sparsity structure can be predetermined; some entries in J can become zero during computation, as is the case with the photolysis terms during night.

To take full advantage of the sparse structure of P, we need to permute its rows and columns such that the sparsity of the L and U factors is maximized. Among different possible strategies, we look at symmetric permutations:

$$P \leftarrow \Pi \cdot P \cdot \Pi^{\mathrm{T}} = I - h \cdot \gamma \cdot \Pi \cdot J \cdot \Pi^{\mathrm{T}}$$

which preserve the presence of ones in the diagonal elements and hence give the possibility of factorization without pivoting. Such symmetric permutations can be viewed as a rearrangement (renumbering) of the species involved in the chemical mechanism (if the initial ordering was $[1, ..., n]^{\mathrm{T}}$, the new ordering will be $\Pi \cdot [1, ..., n]^{\mathrm{T}}$). We further restrict the class of possible permutations to those given by a global strategy; more exactly, we want to compute the permutation off-line, in the preprocessing stage (hence considering only the structure of the matrix and not particular numerical entries). The same permutation is then used throughout the computation, thus reducing the workload associated with sparse data structure manipulation.

The following strategies were considered:

1. *Intuitive.* A reordering based on the reactivity of the species. This requires specific knowledge regarding the chemical mechanism and has been applied successfully in [19]. Since our aim here is to minimize the fill-in, we ordered the species decreasingly after their characteristic lifetime (the inverse of average destruction term, the average being taken over the two day integration interval);
2. *Row.* The rows of P are sorted in increasing order, according to the number of their nonzero elements. If two or more rows have the same number of nonzero elements, their initial relative ordering is preserved.
3. *Column.* Same strategy as above, but applied on columns.
4. *Row*Col.* For each diagonal element k, the value $\alpha(k) = r(k) \cdot c(k)$ is computed, where $r(k)$ is the number of nonzero elements in row k and $c(k)$ is the number of nonzero elements in column k. The permutation Π is such that the diagonal elements of $\Pi \cdot P \cdot \Pi^{\mathrm{T}}$ are sorted in increasing order with $\alpha(k)$.
5. *Diagonal Markowitz.* The algorithm emulates the LU decomposition (say, the column-oriented version) of the initial matrix P. Consider that at the current step, the first $k - 1$ columns have been processed.

   - *Step k.* To minimize the fill-in, look at the diagonal elements of the active submatrix $P(k:n, k:n)$ and choose the one with the lowest $\beta(k) = (r(k) - 1) \cdot (c(k) - 1)$, where $r(k), c(k)$ are relative to the submatrix $P(k:n, k:n)$. If the diagonal element found is (i, i), permute rows k and i and columns k and i. The procedure is equivalent to a diagonal pivoting; unlike the standard Markowitz rule, this diagonal strategy does not use numerical values; hence, it is a global strategy.
   - *Step* $k + \frac{1}{2}$. Do a symbolic decomposition step on column k, i.e. emulate the kth step of a real decomposition, counting the fill-in (zero elements becoming nonzero) in $P(k:n, k:n)$.
   - *Step* $k + 1$. Repeat step k with the new active submatrix $P(k + 1:n, k + 1:n)$.

TABLE I

Resulting Fill-Ins (Number of Nonzeros after an In-Place Factorization) for the Different Reorderings Analyzed

	# of nonzeros in LU	
Strategy	Model 1	Model 2
Initial	243	673
Intuitive	385	2900
Row	287	810
Column	278	806
Row * Col	275	789
Diag Markowitz	275	768
Local min fill-in	274	761

Note. The test problems are discussed in Section 4.6.

A nice feature with this algorithm is that, as a by-product, one obtains the sparsity pattern of the L and U factors.

6. *Local minimum fill-in.* Resembles the Markowitz strategy, except that, at each step, the diagonal element (i, i) is chosen such that the fill-in of $P(k:n, k:n)$ after performing step k of elimination is minimized. This strategy is much more expensive than Markowitz, since at stage k, there are k (symbolic) factorizations of dimension $(n - k)^2$ to be performed. The gain, compared to the diagonal Markowitz strategy, is minimal.

The results for the test problems described in Section 4.6 are presented in Table I. We report the number of nonzeros resulting after reordering and an in-place LU factorization. The real target of reordering is to minimize this fill-in. The results show that all the considered strategies (except the "Intuitive" mode) perform similarly. Interestingly, the species considered most reactive appear last in the "pure numerical" re-orderings as well. For example, for model 2 the last species (in reversed column order) are: OH, NO, HO_2, NO_2, NO_3, O_3, HCHO, ALD_2, ...

Conclusion. The diagonal Markowitz criterion has a slight advantage over the others, so we recommend its use.

4. AN EVALUATION OF DIFFERENT SPARSE SUBROUTINES

4.1. Test Systems

In order to evaluate the performance of different sparse solvers, we employed two test linear systems based on the chemical problems described in Section 4.6. One numerical value of the Jacobian (corresponding to noon-time, first day) was computed. The test systems correspond to $\gamma \cdot h = 1$ and the exact solution is a vector of ones:

$$P = I - J,$$

$$b = P \cdot (1, ..., 1)^T.$$

The sparsity pattern of P was evaluated off-line, and the data structures needed by each tested solver were generated using a small MATLAB routine. Each of the routines described in Section 4.3 received the linear system in its own input format. For all the tests in the next subsection, the exact solution was recovered within an error of 10^{-9}.

4.2. Test Methodology

Usually, the solution of a sparse linear system is done in three distinct steps: (a) analysis of sparsity pattern and preparation of data structures; (b) sparse LU decomposition, using the information gathered at the first step; and (c) solution of resulting pair of triangular systems.

The following particularities appear when solving ODEs arising from atmospheric chemistry:

• The sparsity structure of the Jacobian (and hence, of the prediction matrix) is given by the interactions among chemical species and hence is constant on large time intervals. This structure may be sparser during nighttime when photolysis reactions shut down; since it is cumbersome to either work with different sparsity structures on different intervals or update this structure periodically, we take the simpler approach [21] of moving the analysis step off-line, computing a (maximal) sparsity structure, and working with it throughout the integration time interval.

• Because chord iterations are used when solving the nonlinear system, the same prediction matrix is used for more than one iteration. Hence, to each LU decomposition (step (b)) correspond several calls to the substitution routine (step (c)). For example, with VODE on atmospheric chemistry problems, the number of calls is about six or seven.

To emulate these characteristics, the codes were benchmarked as follows: the ANALYSE routine (corresponding to step (a)) was called once, in the beginning; then the DEC and SOL (corresponding to phases (b) and (c), respectively) were called 10^5 times. Each call to DEC was followed by 0 and by 7 calls to SOL (for timing DEC only and for simulating the calls made by an implicit integrator).

4.3. Short Description of Linear System Solvers Tested

4.3.1. *Off-the-Shelf Solvers*

1. LINPACK. The code is available on netlib; see [7]. LINPACK uses column-oriented algorithms to increase efficiency by preserving locality of reference. Here we test the routines `dgefa` and `dgesl.` LINPACK_U is LINPACK, acting on the *unordered* matrix. More exactly, the order of the species is the reverse of that resulting from the Markowitz diagonal criterion. As a consequence,

the LU factors are almost full. LINPACK_O is LINPACK acting on the *ordered* matrix. Without any further intervention, other than species reordering, the decomposition time may be cut down significantly, as seen in Tables II and III. This is explained by the sparsity of L.

2. LAPACK. The code is available on netlib; see [1]. LAPACK is a collection of Fortran subroutines that supersedes both LINPACK and EISPACK. Tested routines: `dgetrf` and `dgetrs.` Results for both the ordered and the unordered system are given.

3. HARWELL MA28. The code, available on netlib, is written by I. S. Duff, Computing and Information Systems Department, Rutherford Appleton Laboratory, and J. K. Reid. MA28 is a Markowitz general-purpose linear algebra package. Tests with the Harwell package were made by calling `ma28ad` once and then calling the sequence `ma28bd, ma28cd` 10^5 times.

4. Y12M of Z. Zlatev, University of Copenhagen, is a general package for sparse systems of linear equations and is also available on netlib. The routines used: `y12mb` (analyse), `y12mc` (decompose), `y12md` (solve).

5. SuperLU, written by J. W. Demmel, J. R. Gilbert S. Eisenstat, X. S. Li, J. Liu, and J. Teo, uses a supernodal approach to sparse partial pivoting. Routines used in tests: `dgstrf` (*refact = "Y" first call, then with refactorization option) for factorization, and `dgstrs` for solution of triangular systems.

6. UMFPACK2 of T. A. Davis, Computer and Information Science and Engineering Department, University of Florida, and I. S. Duff. Version 2.0 from September 13, 1995 is available on netlib. UMFPACK2 uses an unsymmetric-pattern multifrontal approach (wherefrom it derives its name). `udm2fa` was called once, then `udm2rf` and `udm2so` were called 10^5 times. Different combinations of input parameters were tested (with/without permutation to block triangular form, with/without preferring diagonal pivots, with different number of columns to be examined during the pivot search, with/without iterative refinement). The timings correspond to the default values of parameters.

7. SLAP was written by A. Greenbaum, Courant Institute, and M. K. Seager, Lawrence Livermore National Laboratory, and is available on netlib. The following routines were used: `dsilus` (for performing an in-place LDU decomposition), and `dslui2` (for performing the back and forward substitutions). `dsilus` and `dslui2` were designed to perform an incomplete LDU decomposition of a sparse matrix, thus providing a preconditioner for iterative methods. No permutations are performed, and no fill-in is considered (i.e., only the elements in L and U that correspond to a nonzero position in A are computed). This feature makes the code fast. However, its performance is affected by the large number of indirect addressings in the inner loops. In order to use the pair `dsilus` and `dslui2` as an exact (complete) solver, we predicted off-line the sparsity structure of the L and U factors; then we "extended" the matrix A to this structure, explicitly inserting zeros on the fill-in positions. Two versions (the original and a modified one) were considered: SLAP_1 represents the routines `dsilus` and `dslui2` without any modification. SLAP_2 represents the routine `dsilus` with the following modification: the ANALYSE phase (corresponding to first part of `dsilus`) is done once (off-line); the routine receives directly A (copied into L, D, and U), as well as the sparsity pointers (IL, JL, etc.). As a consequence, a substantial improvement (as compared to SLAP_1) was obtained.

The vendor BLAS version was used with all the routines that required it, except for SuperLU (where some incompatibilities appeared and we used the provided BLAS library).

4.3.2. *Tailored Solvers*

Since the main idea is to do the analysis step off-line and then to use the resulting data structures throughout the computation, we consider several implementations of Gaussian elimination without pivoting.

1. Mod_Gauss_1 is a modified (column-oriented) Gauss algorithm. Its distinctive feature is that in order to reduce the number of indirect addressings, we work directly on the square matrix, rather than on a compressed row or column or on a linked list format. The sparsity patterns of L and U are computed in the preprocessing stage. All the algebraic manipulations are done in a sparse mode. The factorization (decomposition) is done in-place, and the resulting "triangular" L and U factors are used. This lead to a loss in performance in the SOLVE phase, so that the following hybrid data structure was considered:

2. Mod_Gauss_2 uses the full representation in the DECOMPOSITION phase, but then it copies the nonzero elements into (the vectors) L and U. Then the SOLVE phase uses a sparse data structure. This results in a small overhead when DECOMPOSE-ing, and a large speedup when solving. This version is more efficient when a large number of SOLVEs follow each decomposition.

3. DOOLITTLE. This routine uses a row-oriented Doolittle factorization (see [8] for more details on this algorithm). Line k in L is computed, followed by the computation of line k in U. The initial matrix is stored in compressed row format; in addition to the vector of pointers IROW (IROW(k) indicates the beginning of line k) there is a vector of pointers IDIAG (IDIAG(k) is the position of kth diagonal element). The presence of both IROW and IDIAG enables the code to perform an in-

TABLE II

Model A: Times per Call (10^{-6} s) for Different Solvers on a HP-UX A 9000/735 with 160-M RAM Machine

Model A			
Routine	DEC	SOL	1D + 7S
LINPACK_U	714	73	1225
LINPACK_O	411	73	922
LAPACK_U	694	102	1408
LAPACK_O	341	102	1055
HARWELL	393	39	666
Y12	832	28	1028
UMFPACK2	900	73	1411
SuperLU	948	95	1613
SLAP_1	432	25	607
SLAP_2	263	25	438
Mod_Gauss_1	205	55	590
Mod_Gauss_2	228	26	410
DOOLITTLE_1	135	29	338
DOOLITTLE_2	135	10	205

Note. "DEC" is the time for one decomposition, "SOL" is the time for one backward–forward substitution, and "1D + 7S" the time for one decomposition, followed by seven backward–forward substitutions.

place LU decomposition, IROW being associated with the beginning of lines in L and IDIAG with the beginning of lines in U. No pivoting is performed, and no fill-in is considered. Thus, the initial matrix A has to be "extended," in the sense that the positions of fill-ins are computed off-line; then explicit zero entries are considered in these positions. Two versions were tested:

• DOOLITTLE_1. The SOLVE phase works with the sparse data structure resulting from LU decomposition.

• DOOLITTLE_2. In order to completely avoid indirect addressing, a loop-free code is generated (via the KPP symbolic preprocessor) for the SOLVE phase only.

4.4. Results

Timing for solving the systems resulting from our test problems with the above routines are presented in Tables II and III. All the routines solved the linear system

$$P \cdot x = (I - h \cdot \gamma \cdot Jac) \cdot x = b$$

with $h \cdot \gamma = 1$ and $b = P \cdot (1, ..., 1)^T$. In all cases the exact solution $(1, ..., 1)^T$ was recovered within an error of 10^{-9}. Several remarks:

• The new solvers SuperLU and UMFPACK are designed for very large systems of equations (several thousands by several thousands); their use is not justified for small to moderate size problems, as those arising from present-day atmospheric chemistry models.

• General purpose solvers like Harwell's MA package and Y12M have a significant overhead associated with pivoting and handling more general data structures. Their results are reliable; again, this does not seem to pay for small systems (at most several hundreds by several hundreds) arising from atmospheric chemistry kinetics. However, the reasonable performance of MA28 gives us the hint that a Markowitz code (working with simplified data structures) may be well suited for the application.

• Modified Gauss. According to the results from Tables II and III the strategy of performing sparse operations on the full structure seems to work well.

One obvious disadvantage of working with the whole matrix is the increase in the required amount of memory. More tests on different machines show a dramatic degrada-

TABLE III

Model B: Times per Call (10^{-6} s) for Different Solvers on a HP-UX A 9000/735 with 160-M RAM Machine

Model B			
Routine	DEC	SOL	1D + 7S
LINPACK_U	6870	355	9355
LINPACK_O	2240	355	4725
LAPACK_U	8000	493	11451
LAPACK_O	1900	493	5351
HARWELL	1150	103	1871
Y12	2880	82	3454
UMFPACK2	2720	176	3952
SuperLU	2660	246	4382
SLAP_1	2030	67	2499
SLAP_2	1100	67	1569
Mod_Gauss_1	730	138	1696
Mod_Gauss_2	840	67	1309
DOOLITTLE_1	440	93	1091
DOOLITTLE_2	440	30	650

Note. "DEC" is the time for one decomposition, "SOL" the time for one backward–forward substitution, and "1D + 7S" the time for one decomposition, followed by seven backward–forward substitutions.

tion in performance if the amount of RAM is restricted. For example, Mod_Gauss on the 160-M RAM machine (see Tables II, III), performs comparable with MA28. However, on a 64-M RAM machine, it is two times slower than Harwell's MA. This may be explained as follows: the elements of the matrix are addressed directly, but, because of sparsity, successive references require big jumps in the $n * n$ vector (the internal representation of the whole matrix). Thus, the references are no longer local, and the amount of cache memory influences very much the performance.

• Doolittle. This has the advantage of working on a uniform representation of the matrix (one vector of nonzero elements, unlike SLAP, which requires one vector for L, one for U, and one for D). Each row is decompressed before, and recompressed again after, processing (see [8]). This moves most of the indirect addressing from the $O(n^2)$ loop to two $O(n)$ loops. The technique of generating loop-free code for the SOLVE phase (as with Doolittle_2) speeds up considerably the code (in our test problem, a factor of 3 is gained in the SOLVE phase). It has the disadvantage of requiring specialized preprocessing software (in this study we generated the code with the symbolic preprocessor KPP, but the user may write a very simple C routine for this purpose).

A loop-free code for the DECOMPOSE phase may, in principle, substantially improve the timing; on the other hand, this solution would significantly increase the resulting code, and memory problems associated with that may counterbalance the speed gain.

Conclusion. Doolittle with a loop-free code generated for phase (c) seems to be the fastest routine available for atmospheric chemistry applications (this conclusion by no means applies to general linear systems).

4.5. Integrators Used

Since the off-line estimation has shown Doolittle_2 to be the most promising sparse solver, we used it in all the numerical experiments which will be reported here. Three off-the-shelf integrators were used in the numerical experiments in both original and modified (sparse) versions. Each code is based on a different numerical scheme:

• VODE, the variable coefficient ODE solver of Hindmarsch, Brown, and Byrne, a BDF code. For details see [3].

• SDIRK4, written by Hairer and Wanner, part of [10], is based on a stiffly accurate, five-stage, order-4, singly diagonally implicit Runge–Kutta method.

• RODAS, written by Hairer and Wanner, part of [10], is based on a stiffly accurate Rosenbrock method of order 4 with six stages.

TABLE IV

Initial Concentrations for Stratospheric Model A

Species name	Initial (ppb)	Species name	Initial (ppb)
O	8.15	O_3	656
NO	10.7	NO_2	2.75
HNO_3	0.35	H_2O	6100
OH	0.2	HO_2	0.14
H_2	370	CH_4	490
CO	20	ClO	1
HCl	2.15	HOCl	0.22

4.6. Test Problems

TEST PROBLEM A. This corresponds to a stratospheric (altitude 40 km) box model. It is available at NASA ftp site (contact Douglas E. Kinnison, kinnison1@llnl.gov). There are 38 species involved in 84 thermal and 25 photolytic reactions at the following physical and geographical conditions: latitude 65N, temperature 241.43 K, pressure 2.7 hPa, air density 8.12×10^{16} molecules/cm^3. This mechanism and the rate constants have been used in the NASA HSRP/AESA stratospheric chemistry model intercomparisons. The values of initial concentrations for the most important species are given in Table IV.

TEST PROBLEM B. This employs the chemical mechanism that is presently used in the STEM-II regional-scale transport/chemistry/removal model (Carmichael *et al.* [4]), consisting of 86 chemical species involved in 142 thermal and 36 photolytic reactions. The mechanism, based on the work of Lurmann *et al.* [14] and Atkinson *et al.* [2], is representative of those presently being used in the study of chemically perturbed environments. It represents the major features of the photochemical oxidant cycle in the troposphere and can be used to study the chemistry of both highly polluted (e.g., near urban centers) and remote (e.g., marine) environments. The photochemical oxidant cycle is driven by solar energy and involves nitrogen oxides, reactive hydrocarbons, sulfur oxides, and water vapor. The chemistry also involves naturally occurring species, as well as those produced by anthropogenic activities. The model was used to simulate urban conditions at ground level, temperature of 288 K, and an air density of 2.55×10^{19} molecules/cm^3. The values of initial concentrations and the values of hourly emissions are given in Table V. These emissions were performed in equal quantities at the beginning of each restart time interval. The chemical simulations were run for 5 days.

5. NUMERICAL RESULTS

Each of the modified codes was tested for different tolerances and different restart intervals. In this section the

TABLE V

Initial Concentrations and Hourly Emissions for Tropospheric Model B

Species name	Initial (ppb)	Emission (ppb/h)
NO	50	1
NO_2	20	0.2
HONO	1	0
O_3	100	0
H_2O_2	1	0
CO	300	0
HCHO	10	0.2
Aldehyde	10	0.2
PAN	1	0
Alkans	50	2
Alkens	10	1
Ethene	10	0.2
Aromatics	20	4
Isoprene	10	1.0

results for the test problems are compared to the solutions computed by the code RADAU5 of Hairer and Wanner [10] with very tight tolerances rtol = 10^{-12} and atol = 10^{-10} (mlc/cm^3).

As a measure of the accuracy we have employed the *number of accurate digits* (NAD) computed as

$$\mathrm{NAD} = \frac{1}{N}\sum_{i=1}^{N} \mathrm{NAD}_i, \quad \mathrm{NAD}_i = -\log_{10}(\mathrm{ERR}_i),$$

where N is the number of species, ERR_i is a measure of the relative error in the numerical solution of species i, and NAD_i is the corresponding number of accurate digits. With the reference solution $y(t)$ (computed by RADAU5) and the numerical solution $\hat{y}(t)$ at hand at discrete times $\{t_j = t_0 + j \cdot \Delta t,\ 0 \le j \le M\}$ the measure of the relative error is computed as

$$\mathscr{I}_i = \{0 \le j \le M : |y_i(t_j)| \ge a\},$$
$$\mathrm{ERR}_i = \sqrt{\frac{1}{|\mathscr{I}_i|} \cdot \sum_{j \in \mathscr{I}_i} \left| \frac{y_i(t_j) - \hat{y}_i(t_j)}{y_i(t_j)} \right|^2}.$$

The threshold factor used here is $a = 100$ mlc/cm^3. If the set $\mathscr{I}_i$ is empty, the value of ERR_i is neglected. The purpose of considering the above-defined error measure, instead of the root mean square norm ($a = 0$ mlc/cm^3) is to suppress from the error calculation the times where the absolute value of the concentration falls below $a = 100$ mlc/cm^3; these values are very likely corrupted and the corresponding large relative errors say nothing about the general computational accuracy. From a physical standpoint, for atmospheric chemistry applications, values of $a = 100$ mlc/cm^3 or less can be assumed to correspond to the complete disappearance of the species.

In addition to presenting the results for sparse implicit integrators, we perform a comparison with the widely used algorithms QSSA (see [11]), CHEMEQ (see [22]) and the sparse BDF code LSODES (see [12]):

• QSSA is used with a dynamic partitioning of the species into slow, fast, and normal, depending on the step-size h, and the species life-time $\tau_i = 1/D_i$.

—If $\tau_i > 100 \cdot h$ the species is slow and is integrated with forward Euler formula;

—If $\tau_i < 0.1 \cdot h$ the species is fast and is considered at steady state;

—Otherwise, exponential QSSA formula is applied.

• CHEMEQ is used as specified in [17]:

—If $\tau_i < 0.2 \cdot h$ the species is fast and is considered at steady state;

—If $\tau_i > 5 \cdot h$ the species is slow and is integrated with the nonstiff CHEMEQ formula;

—For all other species the CHEMEQ stiff formula is used.

• LSODES is the sparse version of the popular BDF code LSODE. LSODE and LSODES are often used to solve the atmospheric chemical kinetics equations (see [17]). The code was used with $MF = 121$, i.e., the analytical Jacobian with an inner estimation of the sparsity structure.

For implicit integrators, the same parameter setting for both sparse and off-the-shelf versions was used. The accuracy of sparse codes is (despite nonpivoting) very similar to that of the original ones.

Timings and accuracies for the original codes, the sparse versions, and the explicit algorithms are presented in Figs. 1 (test problem A) and 2 (test problem B). The numerical accuracy (expressed as the *number of accurate digits*) is plotted versus the CPU time (in *seconds*, as given by the UNIX routine `timex`). The important points are:

• In Figs. 1 and 2 the BDF codes are represented by solid lines, Runge–Kutta by dashed lines, Rosenbrock by dash–dots, and explicit methods by dotted curves.

• The slopes of these work-precision diagrams measure the orders of convergence of different methods; implicit integrators used here have higher convergence orders; hence they display higher slopes compared to QSSA and CHEMEQ. For less accuracy, the latter are faster, while for higher accuracy the former become preferable. Thus, when more accuracy is desired, the higher convergence order of the implicit methods used here pays off.

• To compare the performance of different methods on

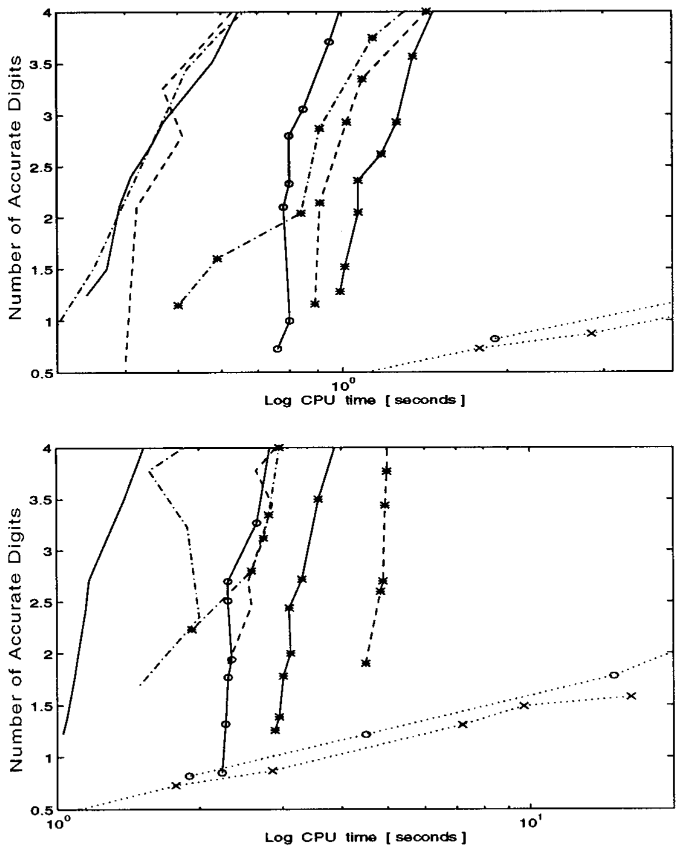

FIG. 1. Model A. Work-precision diagram. A restart was carried each 1 h (upper diagram) and each 15 min (lower diagram). Sparse VODE (solid) VODE (solid with "*"), LSODES (solid with "o"), sparse RODAS (dash–dots), RODAS (dash–dots with "*"), sparse SDIRK4 (dashed), SDIRK4 (dashed with "*"), QSSA (dots with "x"), and CHEMEQ (dots with "o").

the work-precision diagram, "draw" an imaginary horizontal line through the desired level of accuracy (say, two digits) and read from its intersection with the code plots the necessary CPU time for achieving that level of accuracy; the code that gives the leftmost intersection point will be the fastest for that problem.

• Note the shift in the time scales for the lower diagrams versus the upper ones. This shows the increase in CPU time associated with lowering the restart time. This increase affects mainly the performance of implicit methods; although they are capable of working with very large step sizes, frequent restarts and the transient regimes force lower step sizes and a large number of LU decompositions right after each restart.

• Figure 2 shows nicely why CHEMEQ may be preferred to VODE for the tropospheric test problem B when the restart time is 15 min or less. The deterioration of performance with frequent restarts precluded BDF methods to be used in 3D air pollution models. Moreover, if the implicit codes are not supplied analytical Jacobians their performance further decreases.

• Decreasing the transport step and increasing the dimension of the chemical mechanism results in a relative advantage for special explicit methods. However, even with a restart time of 15 min and fairly large chemical mechanisms the standard integrators endowed with the above presented sparse linear algebra techniques are competitive with CHEMEQ and QSSA, when the requested computational accuracy is one significant digit or more.

• Each implicit code traces a pair of almost parallel lines in the diagrams, one for the off-the-shelf version and one for the sparse version; roughly speaking, these lines differ by a translation along the time axis (which corresponds to the speedup). The line parallelism shows that both code versions perform similarly in terms of accuracy; in particular, the accuracy does not seem to be affected by nonpivoting. This experimental conclusion is in agreement with the findings of other authors (see [6, 13, 21]).

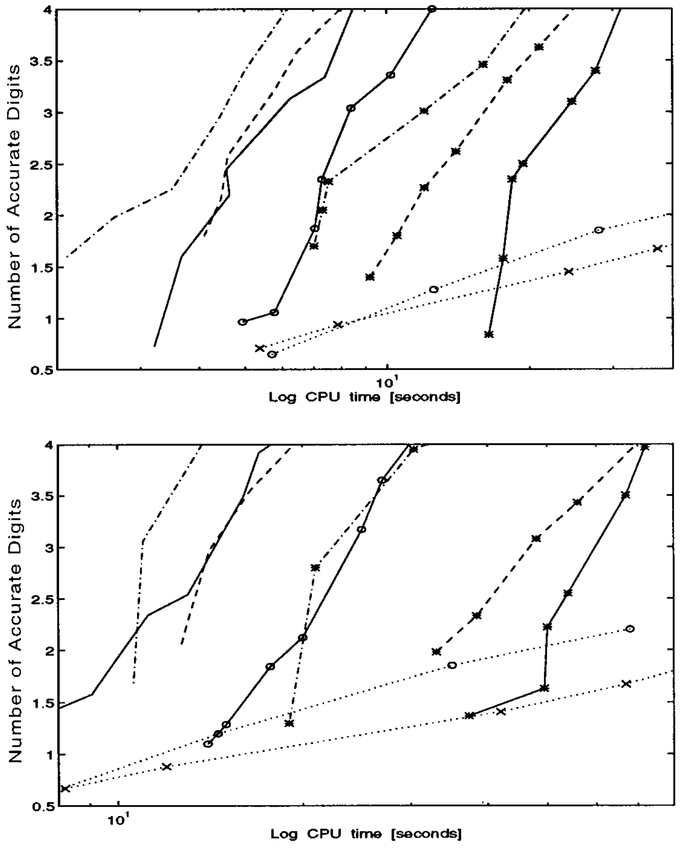

FIG. 2. Model B. Work-precision diagram. A restart was carried each 1 h (upper diagram) and each 15 min (lower diagram). Sparse VODE (solid), VODE (solid with "*") LSODES (solid with "o"), sparse RODAS (dash–dots), RODAS (dash–dots with "*"), sparse SDIRK4 (dashed), SDIRK4 (dashed with "*"), QSSA (dots with "x"), and CHEMEQ (dots with "o").

TABLE VI

Average Speedups Obtained

	Model A		Model B	
Δt [s]	900	3600	900	3600
VODE	2.7	2.6	4.33	4.12
SDIRK4	1.7	2.2	3.25	3.00
RODAS	1.4	1.5	2.50	2.90

To summarize, Figs. 1 and 2 show that careful exploitation of sparsity leads to significant improvements in the efficiency of implicit numerical integrators. These improvements depend on the size of the problem and on the percent of the total CPU time a particular code spends in solving linear algebra. In Table VI we report the average speedups obtained with the considered codes and problems.

The results presented here are for transport time steps (i.e., restart times) of 15 min (a typical value for regional scale models) and 1 h (a typical value for global scale models). Many models may use a transport time step between 15 min and 1 h. The results in Figs. 1 and 2 point to the conclusion that, if the transport time is sufficiently large, sparse implicit methods outperform dedicated integrators in terms of accuracy/time ratio.

6. CONCLUSIONS

In comprehensive air pollution models it is of interest to replace traditional explicit integrators (QSSA, CHEMEQ) by more robust implicit integrators. This interest is motivated by the following arguments:

• The wide range of chemical conditions that are to be simulated can cause numerical problems when explicit integrators are used; on the other hand, implicit methods offer uniformity in performance for equations of variable stiffness and difficulty. This uniformity is important as comprehensive air quality models usually contain chemical conditions ranging from ground level to upper troposphere and from marine environments to heavily polluted urban centers.

• For problems involving interphase mass transfer explicit codes may become unstable; an example of gas–liquid chemistry for which all explicit integrators completely fail can be found in [16].

• The higher orders of consistency of standard implicit methods lead to substantial improvements in accuracy. Using higher order methods based on explicit formulas may not pay off, because of the order reduction phenomenon (see [15, 20]) and since higher accuracy and stability are, in general, contradictory requirements (see [10]).

• Multistep, Runge–Kutta, and standard Rosenbrock methods all enjoy the property of conserving the linear invariants of the system (for example, they are *structurally* mass-conservative). Neither QSSA nor CHEMEQ have this desirable property.

What prevented the so-far implicit methods from being widely used in three-dimensional, comprehensive atmospheric models is the fact that they are considered too slow for this type of application (except for the case when special hardware is available—see the SMVGEAR code [13], running on CRAY-YMP). However, we show that *this is not the case* when the linear algebra is carefully implemented. One can enjoy all the above benefits of implicit methods while remaining computationally very competitive.

The specialized techniques for treating the sparsity described here lead to significant improvements over a general sparse code like LSODES when integrating chemical systems. For the model problems considered here, the standard (full linear algebra) versions of LSODE and VODE perform almost similarly in terms of work/accuracy ratio; in contrast, sparse VODE is about two times faster than LSODES.

The treatment of sparsity described here is rather conservative, since the off-line analysis of the chemical system counts every possible nonzero entry in the Jacobian. Further improvements seem possible by dynamically approximating the Jacobian with a matrix of higher sparsity.

So far, the implicit methods widely used in atmospheric modelling are Gear (BDF) methods. This work shows that methods from other families, like Runge–Kutta and Rosenbrock, can be equally competitive. We have considered three off-the-shelf codes endowed with the above described sparse techniques. Their good performance motivates further search for integrators for atmospheric modelling within the class of implicit methods.

Finally, let us mention that this treatment of sparsity is not restricted to atmospheric modelling; it is applicable to the numerical solution of general chemical systems.

ACKNOWLEDGMENTS

The authors thank Jan Verwer for suggesting the use of RODAS in this context. This work was supported in part by grants from DOE (DEFG02–94ER61855), NASA (NAGW–2428), and the Center for Global and Regional Environmental Research.

REFERENCES

1. E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK User's Guide*, 2nd ed. (SIAM, Philadelphia, 1995).
2. R. D. Atkinson, D. L. Baulch, R. A. Cox, R. F. Jr. Hampson, J. A. Kerr, and J. Troe, *J. Chem. Kinet.* **21,** 115 (1989).
3. P. N. Brown, G. D. Byrne, and A. C. Hindmarsh, *SIAM J. Sci. Stat. Comput.* **10,** 1038 (1989).

4. G. R. Carmichael, L. K. Peters, and T. Kitada, *Atmos. Environ.* **20,** 173 (1986).
5. V. Damian-Iordache and A. Sandu, Technical report, The University of Iowa, 1995 (unpublished).
6. J. J. B. de Swart and J. G. Blom, Technical report, Centrum voor Wiskunde en Informatica, Amsterdam, 1995 (unpublished).
7. J. J. Dongarra, J. R. Bunch, C. B. Moller, and G. W. Stewart, *LINPACK User's Guide (SIAM, Philadelphia, 1979).*
8. I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*, Oxford Sci. Publ. (Clarendon Press, Oxford, 1986).
9. G. Golub and C. F. van Loan, *Matrix Computations* (Johns Hopkins Press, Baltimore/London, 1983).
10. E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems* (Springer-Verlag, Berlin, 1991).
11. E. Hesstvedt, O. Hov, and I. Isaacsen, *Int. J. Chem. Kinet.* **10,** 971 (1978).
12. A. Hindmarsch, *ODEPACK: A Systematized Collection of ODE Solvers* (North Holland, Amsterdam, 1983).
13. M. Z. Jacobson and R. P. Turco. *Atmos. Environ.* **17,** 273 (1994).
14. F. W. Lurmann, A. C. Loyd, and R. Atkinson, *J. Geophys. Res.* **91,** 10,905 (1986).
15. L. O. Jay, A. Sandu, F. A. Potra, G. R. Carmichael, *SIAM J. Sci. Comput.*, accepted.
16. A. Sandu, J. G. Verwer, M. van Loon, F. A. Potra, and G. R. Carmichael, *Rep. Comput. Math.*, No. **85,** University of Iowa, Department of Mathematics; CWI *Report* NM-*9603*, Amsterdam.
17. R. D. Saylor and G. D. Ford, *Atmos. Environ*. **29,** 2585 (1995).
18. D. S.-S. Shieh, Y. Chang, and G. R. Carmichael, *Environ. Software* **3,** (1988).
19. S. Sillman, *J. Geophys. Res.* **96,** 20,735 (1991).
20. J. G. Verwer and M. van Loon, *J. Comput. Phys.* **113,** 347 (1994).
21. J. Verwer, J. G. Blom, M. van Loon, and E. J. Spee, *Atmos. Environ.* **30,** 49 (1996).
22. T. R. Young and J. P. Boris, *J. Phys. Chem.* **81,** 2424 (1977).