

The kinetic preprocessor KPP—a software environment for solving chemical kinetics

Valeriu Damian^a, Adrian Sandu^b, Mirela Damian^c, Florian Potra^d, Gregory R. Carmichael^{e,*}

^a Glaxo SmithKline Beecham Pharmaceuticals, King of Prussia, PA 19406, USA

^b Department of Computer Science, Michigan Technological University, 1400 Townsend Drive, Houghton, MI 49931, USA

^c Department of Computer Science, Villanova University, Villanova, PA 19085, USA

^d Department of Mathematics, University of Maryland Baltimore County, Baltimore, MD 21250, USA

^e Department of Chemical and Biochemical Engineering, University of Iowa, Iowa City, IA 52242, USA

Accepted 10 June 2002

Abstract

The kinetic preprocessor (KPP) is a software tool that assists the computer simulation of chemical kinetic systems. The concentrations of a chemical system evolve in time according to the differential law of mass action kinetics. A computer simulation requires the implementation of the differential system and its numerical integration in time. KPP translates a specification of the chemical mechanism into FORTRAN or C simulation code that implement the concentration time derivative function and its Jacobian, together with a suitable numerical integration scheme. Sparsity in Jacobian is carefully exploited in order to obtain computational efficiency. KPP incorporates a library with several widely used atmospheric chemistry mechanisms and users can add their own chemical mechanisms to the library. KPP also includes a comprehensive suite of stiff numerical integrators. The KPP development environment is designed in a modular fashion and allows for rapid prototyping of new chemical kinetic schemes as well as new numerical integration methods.

© 2002 Elsevier Science Ltd. All rights reserved.

Keywords: Chemical kinetics; Automatic code generation; Sparsity; Numerical integration

1. Introduction

The solution of large numbers of coupled partial differential equations is necessary to analyze coupled transport/chemistry problems associated with many chemical systems (e.g. air pollution problems such as acid rain and photochemical smog, catalysis and reactor design, combustion). These are often computationally intensive calculations involving millions of variables. For example, a basic description of the photochemical oxidant cycle in the atmosphere requires the treatment of some 70 species involved in 200 coupled non-linear

chemical reactions. The numerical integration of the time evolution of the chemical species involved in reaction networks is a challenging task, and often the computational time is dominated by the solution of the coupled and stiff equations arising from the chemical reactions. Writing the code that simulates the chemical evolution in time of the species involved in the chemical mechanism is tedious and error prone work. In addition, the problem solving effort often involves the evaluation of several different integrators and/or exploration of alternative reaction mechanisms (e.g. a reduced form or a more explicit chemical representation).

The kinetic preprocessor (KPP) was designed as a general analysis tool to facilitate the numerical solution of chemical reaction network problems. KPP automatically generates FORTRAN or C code that computes the time-evolution of chemical species, starting with a specification of the chemical mechanism. It also gen-

* Corresponding author

E-mail addresses: damiav01@gsk.com (V. Damian), asandu@mtu.edu (A. Sandu), mirela.damianiordache@villanova.edu (M. Damian), potra@math.umbc.edu (F. Potra), gcarmich@cgrer.uiowa.edu (G.R. Carmichael).

erates the Jacobian and other quantities needed to interface with numerical integration schemes. KPP further allows a rich selection of numerical integration schemes and provides a framework for evaluation new integrators and chemical mechanisms.

2. Simulation of chemical kinetics

To better understand the general kinetic problem, we start with a simple example from stratospheric chemistry. Then we give the general formulation of the law of mass action kinetics. At the end of this section we review several approaches for solving the chemical kinetic problem.

2.1. A chemical kinetic example

KPP was successfully used to treat many chemical mechanisms from tropospheric and stratospheric chemistry, including CBM-IV (Gery, Whitten, Killus & Dodge, 1989; Atkinson, Baulch, Cox, Hampson, Kerr & Troe, 1989; Lurmann, Loyd & Atkinson, 1986), SAPRC (Carter, 1999), NASA HSRP/AESA, etc.

For illustration purposes in this paper we consider a very simple example, namely the generalized Chapman-type mechanism presented in Appendix C. The photolysis rates depend on the normalized sun intensity $SUN(t)$, which is 0 at night and 1 at noon. This mechanism looks at an important subset of stratospheric O_3 chemistry; i.e. that involves photolysis of molecular oxygen (O_2), creation of ozone (O_3) from atomic (O) and molecular oxygen, and destruction of ozone by photolysis, by interaction with excited atomic oxygen (O^{ID}), and by nitrogen oxides (NO_2 and NO) catalytic cycle.

For all chemical species in the model we want to trace the evolution of their concentrations in time. The initial concentrations are given (in molecules per cubic centimeter) in Appendix A. The differential equation of mass action kinetics together with the initial conditions completely determine the concentrations at any future moment. The time evolution of the system is presented in Fig. 1. Note that KPP places no restriction on the units used for concentrations, time, and rate coefficients. The user must ensure, however, that the units used throughout the description of the same chemical mechanism are consistent; adding the units as comments is highly recommended.

We now describe the general chemical kinetic problem.

2.2. The chemical kinetic problem

Consider a system of n chemical species with R chemical reactions $r_1, \dots, r_R$. We denote the concentra-

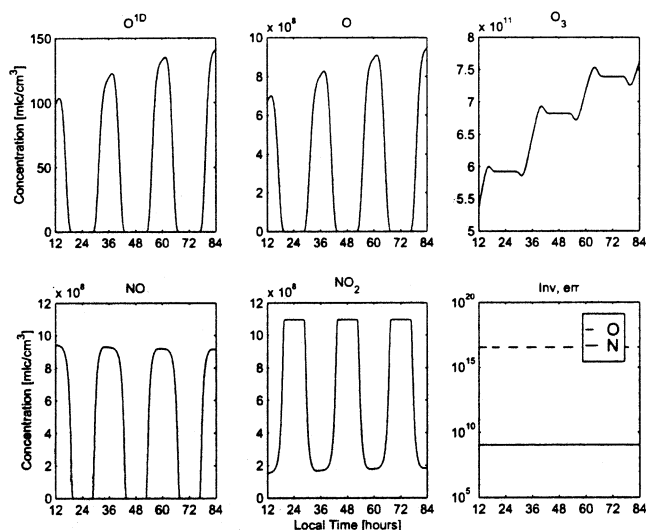


Fig. 1. Time evolution of concentrations of the extended Chapman system. The total amounts of O and N are conserved as seen in the lower right plot.

tion of species i by y_i . Let y be the vector of concentrations of all species involved in the chemical mechanism,

$$y = [y_1, \dots, y_n]^T$$

We define k_j to be the rate coefficient of reaction r_j . We also define the stoichiometric coefficients. $s_{i,j}$ as follows. The stoichiometric coefficient $s_{i,j}^-$ is the number of molecules of species y_i that react (are consumed) in reaction r_j . Similarly, the stoichiometric coefficient $s_{i,j}^+$ is the number of molecules of species y_i that are produced in reaction r_j . Clearly, if y_i is not involved at all in reaction r_j then $s_{i,j}^- = s_{i,j}^+ = 0$.

The principle of mass action kinetics states that each chemical reaction—say the j th reaction in our mechanism.

$$(r_j) \quad \sum s_{i,j}^- y_i \xrightarrow{k_j} \sum s_{i,j}^+ y_i, \quad 1 \leq j \leq R \quad (1)$$

progresses at a rate proportional with the concentrations of the reactants; the proportionality constant is the reaction rate coefficient k_j . In general the rate coefficients are time dependent, $k_j = k_j(t)$; for example photolysis reactions r_1, r_3, r_5 , and r_{10} are strongest at noon and zero at night.

The reaction velocity (the number of molecules performing the chemical transformation each time unit) is, therefore,

$$\omega_j(t, y) = k_j(t) \prod_{i=1}^n y_i^{s_{i,j}^-} \quad (\text{molecules per time unit}).$$

The concentration of species y_i changes at a rate given

by the cumulative effect of all chemical reactions:

$$\frac{d}{dt} y_i = \sum_{j=1}^R (s_{ij}^+ - s_{ij}^-) \omega_j(t, y), \quad i = 1, \dots, n \quad (2)$$

We organize the stoichiometric coefficients in two matrices:

$$S^- = (s_{ij}^-)_{1 \leq i \leq n, 1 \leq j \leq R}, \quad S^+ = (s_{ij}^+)_{1 \leq i \leq n, 1 \leq j \leq R}.$$

Eq. (2) can be rewritten as:

$$\frac{d}{dt} y = (S^+ - S^-) \omega(t, y) = S \omega(t, y) = f(t, y), \quad (3)$$

where, $S = S^+ - S^-$ and $\omega(t, y)$ is the vector of all chemical reaction velocities.

$$\omega = [\omega_1, \dots, \omega_R]^T.$$

Clearly some reactions produce y_i , and their rates give the production rates vector:

$$S^+ \omega(t, y) = P(t, y).$$

Other reactions consume y_i ; note that y_i is consumed only if it is a reactant in such a reaction, therefore, the destruction rate contains y_i as a factor. We convene to leave out the y_i factors from the definition of the destruction terms. Therefore, the destruction term of species i is:

$$D_i(t, y) = \sum_{j=1}^R \frac{s_{ij}^- \omega_j(t, y)}{y_i}.$$

The diagonal matrix of destruction terms:

$$D(t, y) = \text{diag}[D_1(t, y), \dots, D_n(t, y)]$$

allows the loss rates to be expressed as:

$$S^- \omega(t, y) = D(t, y)y.$$

If we explicitly separate the positive terms from the negative ones, we derive the production–destruction form of the equation:

$$\frac{d}{dt} y = P(t, y) - D(t, y)y. \quad (4)$$

To describe the time evolution of the concentrations y_i as given by the mass action kinetics we need the time derivative function either in standard aggregate form $f(t, y)$ as in Eq. (3), or in split production–destruction form $P(t, y)$, $D(t, y)$ as in Eq. (4). Given the initial concentrations, the solution of the ordinary differential equations can be traced in time using a numerical integration method. Implicit integration methods, suitable for stiff chemical kinetics, also require the evaluation of the Jacobian of the derivative function:

$$J = \frac{\partial f(t, y)}{\partial y} = \begin{bmatrix} \frac{\partial f_1}{\partial y_1} & \frac{\partial f_1}{\partial y_2} & \dots & \frac{\partial f_1}{\partial y_n} \\ \frac{\partial f_2}{\partial y_1} & \frac{\partial f_2}{\partial y_2} & \dots & \frac{\partial f_2}{\partial y_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial y_1} & \frac{\partial f_n}{\partial y_2} & \dots & \frac{\partial f_n}{\partial y_n} \end{bmatrix}.$$

2.3. Solving the chemical kinetic problem

There are several possible approaches to translate the chemical reactions into differential equations and numerically solve the latter. The most important ones are summarized below:

- i) The hard-coded approach. In this approach the production and the destruction functions related to the chemical equations are derived and coded by hand. This approach has been used in STEM-I and -II developed by Carmichael, Peters and Kitada (1986) using FORTRAN-77. The advantage of this method is that the subroutines that implement the production and the destruction terms can be coded in a way that is computationally very efficient and machine-specific. The major disadvantage of this method is that minor changes to the chemical mechanism involve rewriting the whole code from scratch. This makes the hard-coded method very unreliable. Writing such a code is a tedious and error prone work, which poses a big problem to real cases that have hundreds of equations with hundreds of species.
- ii) At the other end is the totally integrated approach. In this approach the chemical equations are given in a special file, written in a specific description language. A program parses the equations and stores them in memory as arrays of coefficients, which are used by the production and destruction functions at run time. Thus the code adapts easily to any chemical mechanism. This approach was used by LARKIN in Nowak (1982). The drawback of this approach is that all computation is performed at run time, which results in reduced speed.
- iii) A third approach is the preprocessing approach. This is the approach that we describe in this paper. Here, as with the totally integrated method, the chemical mechanism is described in a specific language. Then a preprocessing step parses the chemical equations and generates appropriate code that simulates the chemistry kinetics in a high level language (FORTRAN-77 or C). This code is almost as efficient as the code produced using the hard-coded method and is guaranteed to be correct. The code also adapts easily to any changes to the chemical

mechanism. This method was partially adopted by CHEMKIN in [Kee, Rupley and Miller \(1989\)](#).

The KPP described in this paper fully implements the preprocessing method. It defines a specific language for describing the chemical mechanism, the initial values and the integration options, including the capability to select the numerical integration method and the integration driver. This language is called the KPP-language, and is presented in detail in [Section 4](#). KPP generates FORTRAN or C optimized code for the production and destruction functions in split or aggregate form, as well as the Jacobian in either sparse or full format. Special purpose sparse linear algebra routines have been developed ([Sandu, Potra, Damian & Carmichael, 1996](#)) and are implemented in KPP. KPP also includes an impressive suite of modern integrators such as VODE, LSODES, RODAS, ROS4, SDIRK, SEULEX, QSSA, EXQSSA, RADAU5, RODAS3 and ROS3. See [Sandu, van Loon, Potra, Carmichael and Verwer \(1997\)](#), [Sandu, Blom, Spee, Verwer, Potra and Carmichael \(1997\)](#) for a benchmarking of these integrators and references to full descriptions and numerical results.

Other chemical mechanism processing software tools have also been developed and are currently widely used. The chemical mechanism processing software of [Carter \(1988\)](#) processes input chemical files and produces FORTRAN simulation code. This software is able to lump organics and their emissions, to calculate photolysis rates, to treat variable stoichiometric coefficients, to distinguish between active, build-up, steady state, constant, saved dummy, and deleted dummy species, to treat emissions, etc. Jacobson and Turco's SMVGEAR ([Jacobson & Turco, 1994](#); [Jacobson, 1998](#)) is a sparse, vectorized Gear code for atmospheric models. SMVGEAR automatically sets arrays of derivatives from the reactions described in a user file. The code automatically determines the sparsity structure of the Jacobian and reorders the species to enhance the sparsity treatment, calculates mass balances automatically by summing atoms across all species, etc. Vectorization of the computations across grid cells result in a remarkable efficiency in three dimensional simulations. Models-3 is a complex framework for environmental modeling and assessment. The Community Multi-Scale Modeling System (CMAQ) includes preprocessors for land-use, meteorology–chemistry interface, emissions–chemistry interface, photolysis rates, initial and boundary conditions, and chemical-transport modeling. It allows the user to select one of several chemical mechanisms and one of several chemical solvers (SMVGEAR or the quasi-steady-state-approximation QSSA).

The strength of KPP compared with the chemical processing tools discussed above lies in the integration with very efficient numerical analysis routines and its

ability to automatically generate the complete simulation FORTRAN and C code, KPP allows the selection of a numerical integrator from a rich set of state of the art integration schemes. The modular design of KPP also allows for rapid prototyping of new integration schemes and models. Thus KPP can be used as an accurate benchmarking platform for evaluating new integrators.

In what follows, we describe KPP from a user perspective and show how to accomplish chemical simulations using the KPP preprocessor.

3. KPP overview

This section contains a high level description of the KPP modules from the user's perspective. The user, presumably a chemist or chemical engineer, describes the application in terms of logical modules such as chemistry model, numerical integrator and driver. For a better understanding of the KPP preprocessor, the following subsections first describe the input files, then the output file and finally the KPP internal modules.

The following subsections present each of these modules starting with the input files, continuing with the output files and concluding with the KPP internal modules.

3.1. KPP input files

Input to KPP is provided in description files, which contain commands defined in the KPP language. The KPP language allows the user to specify a set of chemical equations, initial concentration values for all species involved, a numerical integration algorithm and a set of integration parameters. The KPP language is described in detail in [Section 4](#).

3.1.1. Kinetic description file

The kinetic description file (*.def) contains a detailed description of the chemical model including the atoms, chemical species and kinetic equations. It also contains information about the integration method used and the desired type of results. KPP incorporates descriptions of several widely used atmospheric chemistry models: smog model, stratospheric model, tropospheric model, wet deposition model and cbm4 model, as illustrated in [Fig. 2](#). These chemical models contain default initial values for chemical species and default options, including the best integrator and driver for the model.

In its simplest form, a kinetic description file may contain a single line selecting one of the predefined models. All default values and options related to the selected model are then used. The default settings can be overridden by selecting a different integrator or driver in the kinetic description file.

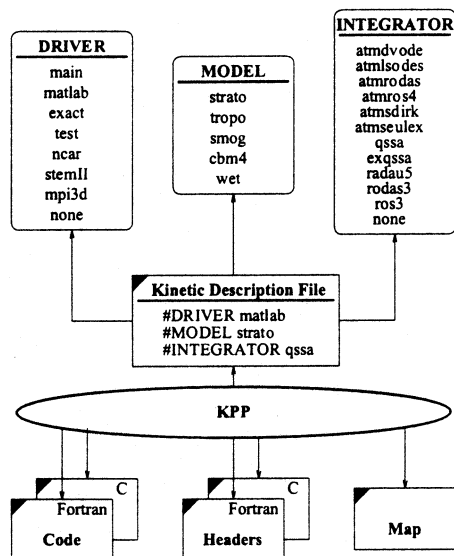


Fig. 2. KPP modules.

KPP accepts only one kinetic description file as input, but we can put together descriptions from separate files using the `#INCLUDE` command as described in Section 4.2. These ideas are illustrated in Appendix A, which shows the main kinetic description file for a small stratospheric small_strato. Note that the chemical species and chemical equations are defined in two separate kinetic description files (Appendices B and C, respectively), which are included in the main description file using the `#INCLUDE` command.

By carefully splitting the description of the chemical model, KPP can be configured for a broad range of users. Each user can directly access parts of the chemical model of interest, and keep unwanted details hidden in several include files. A detailed description of the contents of a chemical model definition file is given in Section 4.

3.1.2. Integrator description file

The kinetic description file points to an integrator description file, also written in the KPP language. The integrator description file in turn contains a reference to a C or FORTRAN source file (depending on the language used) that implements the integrator. In terms of files, the structure of the chemical model is detailed in Figs. 3 and 4. In Fig. 3, the integrator description file is `int/rodas3.def` and points to the FORTRAN source file `int/rodas3.f` that contains the code for the integrator routine.

Each numerical integrator may require KPP to generate different functions. For instance, some integrators need the derivative function in the aggregate form, while others require the split (production/destruction) form. In addition, implicit integrators require the Jacobian; some integrators use standard linear algebra

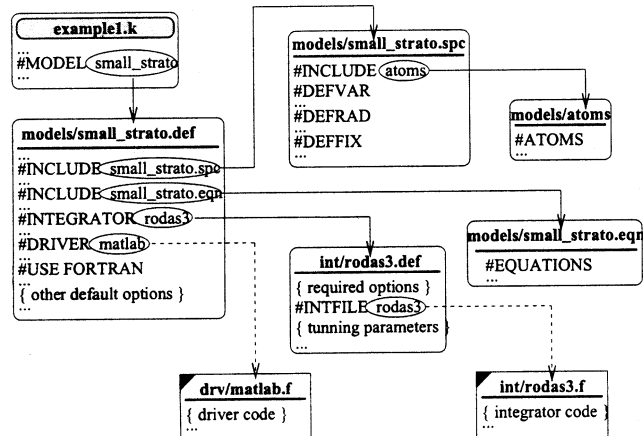


Fig. 3. Input files details (default integrator).

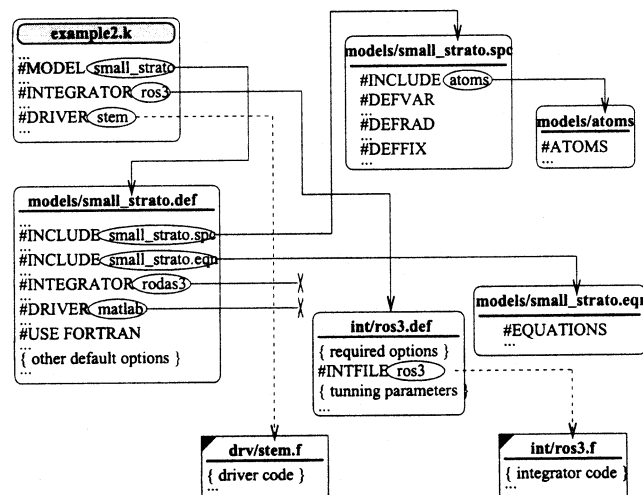


Fig. 4. Input files details (selected integrator).

and need the full Jacobian, while others use sparse linear algebra and require the Jacobian in sparse format. These options are selected through appropriate parameters given in the integrator description file. Some integrators have additional parameters that can be fine tuned for better performance. Values for these parameters are also included in the integrator definition file. KPP includes a predefined library that contains an integrator definition file for each integrator implemented in KPP. The default integrator definition file selects the parameters appropriate to the integrator, so the user does not need any knowledge on the formulation and numerical aspects of the integration techniques. However, KPP is flexible enough to allow more advanced users overwrite the default settings. It is important, though, that the new settings are properly set in the integrator description file (for example, Jacobian sparse should be selected for an implicit method with sparse linear algebra). KPP also allows the user to add new integrators to the library. For

the small stratospheric chemistry mechanism, the integrator description file is given in [Appendix D](#).

3.1.3. Driver file

The driver is the main program. The driver is responsible for calling the integrator routine, for reading data from files and writing out the results. Several drivers are built in KPP and they differ from one another by the format of their input and output data files, or by auxiliary files created for interfacing with visualization tools. A program that performs 3D atmospheric chemistry simulation is also called a driver, if it calls the integrator routine generated by KPP for the chemistry integration. Both the driver and integrator code files are regular FORTRAN or C files that implement different drivers and numerical integration algorithms, respectively.

3.2. KPP output files

KPP generates several output files: a code file, one or more header files and a map file. Each file has a time stamp and a complete description of how it was generated. The time stamps are grouped as comments in the target language at the beginning of each output file.

The naming convention used by KPP is that each file generated has the same root name as the kinetic description file and appropriate extension .f or .c for the code file (depending on whether FORTRAN or C is used), .h and _s.h for the header files and .map for the map file. For instance, if the name of the kinetic description file is `small_strato.def`, then the map file is called `small_strato.map` and the header files are called `small_strato.h` and `small_strato_s.h`. If FORTRAN is used as target language, then the code file is named `small_strato.f`. If C is used as target language, then the code file is named `small_strato.c`. In the following, we give a brief description of the contents of the files generated by KPP.

3.2.1. Code file

This is the main file generated by KPP. It is a complete and correct FORTRAN or C program code that integrates the given chemical mechanism using the selected integration algorithm. Special purpose sparse linear algebra routines are also generated based on the sparsity pattern of the Jacobian, as outlined in [Sandu et al. \(1996\)](#). The code can be directly compiled using the FORTRAN or the C compiler. The code generated is the same for all target architectures.

Note that the KPP-generated code can become a part of a more comprehensive simulation, for example KPP can be used to build the chemical part of a 3D atmospheric chemistry-transport model. In this situation each grid cell holds its local copies of the

concentrations and of the rate coefficients, therefore, the location dependence of the rate coefficients and fixed species can be easily simulated. The temporal dependencies via variations in temperature, pressure, relative humidity etc. can also be incorporated with minimal user intervention as explained later in the paper (see [Section 4](#) species definition).

3.2.2. Header files

KPP always generates a header file with extension .h appended to the root name of the kinetic description file. This header file contains variable and parameter declarations used by all functions in the code file. If FORTRAN is used as a target language, this header file also contains a common block declaration. The header file is normally included in each FORTRAN function. If C is used as a target language, the header file is included only once at the beginning of the C code file. If the user chooses to have the Jacobian computed by KPP, an additional sparsity header file (*_s.h) that contains the sparse data structures is generated.

3.2.3. Map file

This file contains supplementary information for the user. Several statistics are listed here. Examples are total number equations, total number of species and the number of radical, variable and fixed species. A list of all species including their name, type and numbering from the chemical mechanism follows. In addition, the map file contains a complete list of all functions generated in the code file. A brief description of the input and output parameters along with the computation performed is generated for each of these functions.

3.3. KPP internal modules

In this section we give a brief description of the internal architecture of the KPP preprocessor. A detailed description of the KPP architecture and its implementation is given in [Damian \(1998\)](#). In the following we briefly describe the basic KPP internal modules and their functionalities. An overall view of the KPP architecture is given in [Fig. 5](#).

3.3.1. Scanner and parser

This module is responsible for reading the kinetic description files and extracting the information necessary in the code generation phase. It stores this information in some internal data structures that define the list of atoms, species, equation rates, options and the coefficient matrices.

We use the Flex and Yacc generic tools to implement the scanner and the parser. The scanner and the parser perform error checking on each input line. For any syntax error encountered in any of the input files, the scanner/parser report a detailed error message along

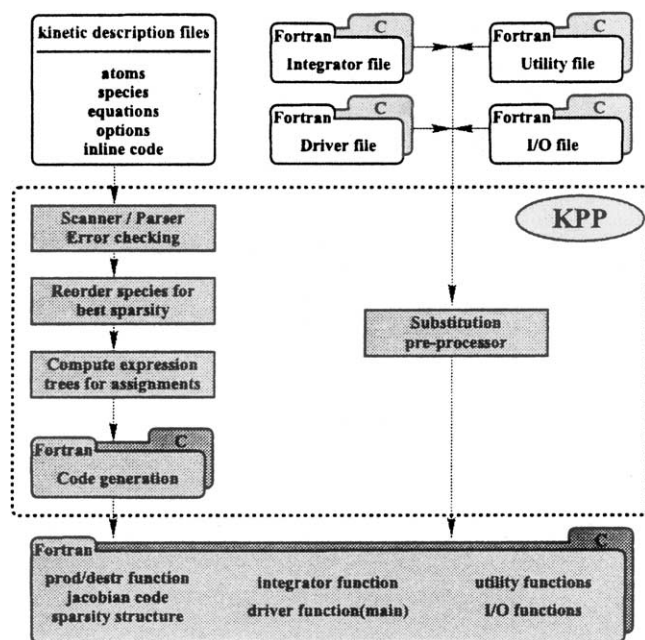


Fig. 5. Overall KPP architecture. KPP is representative for the novel generation of software tools we are developing for solving chemistry kinetics.

with the exact line number in the input file. In most cases the exact cause of the error can be identified, therefore, error messages are very precise. Other errors like mass balance and equation duplicates are checked for at the end of this phase.

3.3.2. Species reordering

Species are reordered using a diagonal Markovitz algorithm to minimize Jacobian fill-in and take maximal advantage of sparsity. This reordering can be turned off if desired.

3.3.3. Expression trees computation

This is the core of the preprocessor. This module generates the production–destruction or aggregate functions along with the Jacobian and all data structures needed by these functions. In doing this efficiently, we build an intermediate structure for each statement in the target code file. Since the vast majority of statements in the target code file are assignments, the intermediate structure is in the form of an expression tree. We build the expression tree incrementally by scanning the coefficient matrices and the rate constant vector, then we simplify this tree to produce a compact output expression. We adopt a similar approach function declarations and prototypes and data declaration and initialization.

3.3.4. FORTRAN and C code generation

KPP contains two code generation modules, each dealing with the syntax particularities of the FORTRAN 77 and the C languages.

3.3.5. Substitution preprocessor

The substitution processor allows us to include any regular C or FORTRAN code in the generated code, without any changes. Two problems appear here; (i) switching from single to double precision involve replacing all single precision declarations to double precision in the input code (ii) if the input code is in a separate file and uses species defined in the header file, the header file must be included in the input file. However, the name of the header file depends on the model in use. To accommodate for this, KPP defines two tokens: KPP_REAL and KPP_CRTFN.

3.3.5.1. KPP_CRTFN: The preprocessor replaces all occurrences of this token in the input file by the root name of the kinetic description file. For instance, if the line `INCLUDE KPP_CRTFN.h` appears in the input file and the model `small_strato.def` is used, then this line will be substituted in the generated code by `INCLUDE small_strato.h`.

3.3.5.2. KPP_REAL: The preprocessor replaces all occurrences of this token by the appropriate single or double precision declaration type. For instance, if the line `KPP_REAL BIG (1000)` appears in the input file and single precision is used, this line will be substituted in the generated code by `REAL BIG (1000)`. If double precision is used, this line will be substituted by `REAL*8 BIG(1000)`.

4. KPP language

In this section we describe the syntax and semantics of the KPP language and the overall structure of a KPP description file. A KPP description file must conform to the following basic rules:

- A KPP description file, contains KPP sections, KPP commands and program fragments.
- Anything enclosed between “{“and”}” is a comment and is ignored by the preprocessor.
- Any name given by the user (to denote an atom or a species, for example) is restricted to be less than 32 characters in length and cannot contain blanks, tabs, new lines, #, +, −, :, ∴. All names are case insensitive.

4.1. KPP sections

A section begins on a new line with a # sign followed by a section name. A section contains a list of items separated by semicolons. The syntax of an item definition is different for each particular section. The sections defined in the KPP language are as follows:

```
#ATOMS (atom_definition_list),
#DEFVAR (species_definition_list),
#DEFRAD (species_definition_list),
#DEFFIX (species_definition_list),
#SETVAR (species_list_plus),
#SETRAD (species_list_plus),
#SETFLX (species_list_plus),
#EQUATIONS (equation_list),
#INTVALUES (initvalues_list),
#CHECK (atom_list),
#LOOKAT (species_list atom_list),
#MQNITOR (species_list atom_list).
```

In the following we give a detailed description of each of these sections.

4.1.1. Atom definition

The atoms used to specify the components of a species must be declared in an #ATOMS section. Usually the names of the atoms are the ones listed in the Periodic Table of Elements. KPP contains a predefined file named atoms that defines all atoms in the Periodic Table of Elements, as in:

```
#ATOMS H; O; N; He; Li; ...
```

To be able to use these definitions in the kinetic description file, we must include the atoms file using the command:

```
#INCLUDE atoms
```

This command must appear in the kinetic description file before any statement that uses an atom defined in the atoms file.

4.1.2. Species definition

Species can be fixed, radical or variable. A species is considered fixed if its concentration does not change during chemical reactions (e.g. the concentration of O₂ in the atmosphere is driven by physical and not chemical processes). As opposed to fixed species, radical species are short life species, i.e. their concentration varies very fast. Variable species are medium life species and their concentration varies in time moderately fast.

Fixed, variable and radical species must be declared in the #DEFFIX, #DEFRAD and #DEFVAR sections, respectively. These KPP sections define all species involved in the chemical mechanism. The type of a

species is implicitly defined by the section that contains the species definition. This division of species into different categories is very useful to integrators that benefit from treating the species differently, based on their type.

Note that the concentration of fixed species can be allowed to change in time with minimal user intervention in the generated code; a function that updates the fixed species can be called before computing the reaction rates. This may be necessary to accommodate, for example, the changes in oxygen concentration due to temporal variations in temperature and pressure.

For each species the user needs to declare the atom composition. This information is used for mass balance checking. If a species is a generic species and its exact composition is not known or not important, it can always be ignored. We must inform KPP of any unknown species composition by declaring the predefined atom IGNORE as part of the species composition, as in the following example:

```
#DEFVAR
NO2 = N + 2O;
RCHO = ignore;
RO2 = 2O + ignore;
#DEFRAD
OH = O + H;
#DEFFIX
CO2 = C + 2O;
```

Here RCHO and RO₂ in photochemical oxidant chemical mechanism in which hydrocarbon reactants are lumped into a relatively small number of species and RCHO represents an aldehyde with a nonspecific organic group R.

We declare species in one category or the other, according to their usual behavior (variable, radical or fixed). Some integrators do not take into account the different types of species involved in the chemical mechanism. In such cases all species should be declared variables. We can redefine the type of an already defined species to be variable by redeclaring it in the #SETVAR section:

```
#SETVAR OH; CO2;
```

For symmetry, KPP also defines the #SETRAD and #SETFIX sections, but they are rarely used. KPP also defines a set of generic species named VAR_SPEC, RAD_SPEC, FIX_SPEC and ALL_SPEC. The first three generic species in this list refer to all variable, radical and fixed species, respectively. The generic species ALL_SPEC refers to all species involved in the chemical mechanism. These generic species can be used in any place where a species name is expected:


```
#DEFVAR ALL_SPEC;
#DEFRAD OH;
```

These lines state that all species are variable species, except for OH, which is a radical species. We can change the types of the radical species to variable species subsequently using:

```
#SETVAR RAD_SPEC;
```

These generic species allow us to easily move all species from one category to another and offer great flexibility in experimenting with different integrators that may or may not take into account the types of the species involved in the chemical mechanism.

4.1.3. Equation specification

The chemical mechanism is specified in the #EQUATIONS section. Each equation is written in the natural way, as shown in the example in [Appendix C](#). Only names of already defined species can be used. The rate coefficient is always placed at the end of each equation, separated by a colon:

```
O + O2 = O3 : 8.018E - 17;
```

The rate coefficient does not necessarily need to be a numerical value. Instead, it can be any valid expression in the target language:

```
OH + NO2 = HNO3 : 2.5 * EXP( (3.0 / TEMP) );
```

For the moment, the expression defining the rate coefficient is restricted to be less than 60 characters. In particular the rate coefficient can be described by a call to a user-defined function, e.g.

```
O + O2 = O3 : USERDEFINED(TIME, TEMPERATURE,
PRESSURE, LIGHT);
```

KPP will insert the function call (the string) into the code; the user has to supply the description of the function and to declare the arguments in the header file.

4.1.4. Initial concentration values

The initial concentration values for all species may be defined in the #INITVALUES section, as shown in [Appendix A](#). For all predefined chemical models, if no value is specified for a particular species, the default value is used. Default values can be set or overridden using the generic species name ALL_SPEC:

```
#INITVALUES
ALL_SPEC = 1.0e - 8;
NO2 = 3.467;
```

These lines state that all species have initial concentration value 1.0×10^{-8} , except for NO₂ whose concentration is 3.467. Similarly, generic species names

VAR_SPEC, RAD_SPEC and FIX_SPEC can be used to set default values for all species in the corresponding category. In order to use coherent units for concentration and rate coefficients, it is sometimes necessary to multiply each concentration value by a constant factor. This factor can be set using the generic name CFAC-TOR (as shown in [Appendix A](#)). Each of the initial concentration values will be multiplied by this factor before being used.

4.1.5. Mass balance checking

KPP performs mass balance checking for each equation. Some chemical equations are not balanced for all atoms, and this may still be chemically correct. To accommodate for this, KPP performs mass balance checking only for the atoms listed in the #CHECK section:

```
#CHECK N; C; O;
```

or to perform balance checking for all atoms, use:

```
#CHECK ALL_SPEC;
```

By default, if this section is missing, no balance checking is performed.

4.1.6. Output data selection

KPP allows us to specify the species whose evolution we are interested in. Such species must be defined in the KPP sections #LOOKAT and/or #MONITOR:

```
#LOOKAT NO2; CO2; O3; N;
#MONITOR O3; N;
```

The LOOKAT section contains a list of species whose evolution is to be saved into a data file by the driver. The drivers built in KPP generate a data file with the same name as the root name as the kinetic description file and extension.dat. For instance, if the name of the kinetic description file is small_strato.def, then a KPP driver saves the evolution of all species in the LOOKAT section into a file called small.stato.dat. To simplify the driver's work, KPP provides three utility functions InitSaveData to open the data file, SaveData to actually save the data and CloseSaveData to close the data file. The driver may simply call these functions. By default, if no LOOKAT section is present, then the evolution of all species involved in the chemical mechanism is saved in the data file. If an atom is specified in the LOOKAT list, then the total mass of that particular atom is reported. This allows us to check whether the total mass of a specific atom has been conserved by the integration method used.

The MONITOR section contains species whose concentrations are displayed at each time step during the integration. This important feature offers instant feed-

back on the evolution in time of the selected species during integration.

4.2. KPP commands

A command begins on a new line with a # sign, followed by the command name and one or more parameters. Here is the list of all commands available in the KPP language:

```
#INCLUDE file_name,
#FUNCTION (AGGREGATE | SPLIT),
#JACOBIAN (OFF | ON | SPARSE),
#SPARSEDATA (OFF | ALL LU_ROW | JAC_ROW | LU_CROW | JAC_CROW),
#DOUBLE (OFF | ON),
#CHECKALL,
#LOOKATALL,
#USE (FORTRAN | C),
#INTEGRATOR file_name,
#DRIVER file_name,
#INTFILE file_name,
#REORDER (ON | OFF),
```

In the following we give a detailed description of each of these commands.

#INCLUDE instructs KPP to look for the file with the name specified as a parameter and parse the content of this file before proceeding to the next line. This allows the atom definitions, the species definitions and even the equation definitions to be shared among several models. Moreover, this allows for custom configuration of KPP to accommodate various classes of users.

#FUNCTION controls the time derivative functions generated to compute the time evolution of the concentration of the chemical species. Two options are available: **AGGREGATE** and **SPLIT**. If the **AGGREGATE** option is used, KPP generates one function that computes the normal derivatives. If the **SPLIT** option is used, then KPP generates two functions for the derivatives in production and destruction forms.

#JACOBIAN controls the computation of the Jacobian for the derivative function. Three options are available: **OFF**, **ON** and **SPARSE**. Parameter **OFF** indicates that the integrator does not need the Jacobian and thus the KPP preprocessor does not generate it. Parameter **ON** indicates that the integrator needs the whole Jacobian for variable and radical species. Parameter **SPARSE** indicates that the integrator needs the whole Jacobian, but in a sparse format. The sparse Jacobian is stored in compressed format in which zeros are not stored explicitly. This is important for applications that require high performance computing, to reduce storage requirements and also to eliminate the need to compute with zeros. The sparse format used is controlled by the **SPARSEDATA** option; the default is

compressed by rows. Note that for very small systems or systems with a low sparsity Jacobian, the use of the **SPARSE** option wastes both time and memory, and therefore, in such cases the use of the **ON** option (full Jacobian) is recommended. The **ON** option is also useful for comparison purposes and for interfacing KPP with off-the-shelf numerical integration software.

#SPARSEDATA controls the sparsity information which is to be generated by KPP. KPP is able to generate sparse data for the Jacobian and for the LU decomposition of the Jacobian. The sparse matrices are stored in compressed format, in which zeros are not stored explicitly. The simplest format, is to store the coordinates (row and column) of the non-zero elements, as shown in the corresponding entry in Table 1. This format can be selected using the options **JAC_ROW** and **LU_ROW** for the Jacobian and the LU decomposition of the Jacobian, respectively. Another commonly used format that permits indexed access to rows (but not columns) is compressed by rows. This format can be selected using options **JAC_CROW** and **LU_CROW** for the Jacobian and the LU decomposition of the Jacobian, respectively. Refer to Table 1 for an example that illustrates the compressed by row format. Three arrays are used in this format: (i) one array data to store the nonzero elements of the matrix row by row (ii) a second array col of the same size used to store the column positions of the elements in data and (iii) a third array crow that has one entry for each row of the matrix, and it stores the position in data of the first non-zero element, of each row of the matrix.

Parameter **OFF** indicates that no sparse data structure is to be generated by KPP and parameter **ALL**

Original matrix

$$\begin{bmatrix} \boxed{a} & \boxed{b} & 0 \\ 0 & 0 & \boxed{c} \\ \boxed{d} & 0 & \boxed{e} \end{bmatrix}$$

Nonzero	data:	a	b	c	d	e
elements		↑ ₁		↑ ₃	↑ ₄	
Store by	row:	1	1	2	3	3
coordinates	col:	1	2	3	1	3
Compressed	crow:	1	3	4		
by row	col:	1	2	3	1	3

indicates that all sparse data structures discussed above are to be generated by KPP.

#DOUBLE selects single or double precision arithmetic for the integrator. ON means double precision. OFF means single precision.

#CHECKALL is a shorthand command for #CHECK ALL_SPEC described before. It indicates that all species are to be involved in the mass balance checking process.

#LOOKATALL is a shorthand command for #LOOKAT ALL_SPEC described before. It indicates that the concentration of all species must be saved in a data file at each time step during integration.

#USE selects the language used to generate the output code file. Available options are FORTRAN and C. Note that the driver and integrator should also be available in the selected language.

#INTFILE selects the file that contains the integrator routine. This command allows the use of different integration techniques on the same model. The parameter of the command is a file name, without extension. Depending on the language used (FORTRAN or C), the appropriate extension (.f or .c) is automatically appended. The file given as argument is copied into the code file in the appropriate place. Clearly, this is only possible if all integrators conform to the same specific calling sequence. An integrator routine takes two arguments: TIN (start time) and TOUT (final time). If C is used as target language, the prototype of the integrator routine is:

```
void INTEGRATE (double TIN, double TOUT)
```

If FORTRAN is used as target language, the prototype of the integrator routine is:

```
SUBROUTINE INTEGRATE(TIN, TOUT)
  REAL*8 TIN
  REAL*8 TOUT
```

Existing general purpose integrators can be embedded in KPP by defining a wrapping function with syntax required by KPP that calls the actual solver with correct arguments.

#INTEGRATOR selects the integrator description file. The parameter is a file name, without extension. The .def extension is automatically appended. The effect of:

```
#INTEGRATOR file
```

is identical to:

```
#INCLUDE file.def
```

The integrator description file for the stratospheric example is shown in [Appendix D](#).

#DRIVER selects the driver, which is the file that contains the main function. The parameter is a file name, without extension. Depending on the language used (FORTRAN or C), the appropriate extension (.f or .c) is automatically appended.

#REORDER specifies whether KPP should reorder the species for best sparsity (option ON) or not (option OFF).

4.3. KPP program fragments

A program fragment begins on a new line with #INLINE followed by a fragment type and ends with #ENDINLINE. In between there is a piece of code written in FORTRAN or C, depending on the fragment type. Fragment types that start with F_ indicate that FORTRAN code is used and fragment types that start with C_ indicate that C code is used. This piece of code is inserted in the output code file in places also determined by the fragment type. [Table 2](#) lists all fragment types defined in KPP and the position in the generated code where these fragments are placed. If two program fragments with the same fragment type are declared, then one of the following happens:

- override: The second program fragment overrides the first one. The fragment types that follow this behavior are marked with override in [Table 2](#).
- append: The second program fragment is appended to the first one. The fragment types that follow this behavior are marked with append in [Table 2](#).

Examples of program fragments are given in the integrator description file in [Appendix D](#).

5. Availability

KPP is publicly available on the web. Both the source code and the documentation can be accessed from <http://www.cs.mtu.edu/~asandu/Kpp> or <http://www.cgrer.uiowa.edu/people/vdamian>.

Several major environmental research and numerical analysis groups from USA, The Netherlands, France, Germany, Italy, Japan and other countries are currently using KPP.

6. Conclusions

KPP is a symbolic preprocessor for solving chemistry kinetics. It provides a simple natural language to describe the chemical mechanism. In a preprocessing step, KPP parses the chemical equations and generates appropriate output code in a high level language (FORTRAN-77 or C). Since this parsing process is not

Table 2
Fragment types

Fragment type	Language	Behavior	Code placement
F_DATA	FORTRAN	Append	In global data declaration and initialization. Allows for declaration of new variables
C_DATA	C	Append	
F_DATA_INT	FORTRAN	Override	
C_DATA_INT	C	Override	
F_DECL	FORTRAN	Append	In global data declaration in the header file. Used for external variables or common block description
C_DECL	C	Append	
F_DECL.INT	FORTRAN	Override	
C_DECL_INT	C	Override	
F_INIT	FORTRAN	Append	At the end of the initialization routine. Used to define integration parameters
C_INIT	C	Append	
F_INIT_INT	FORTRAN	Override	
C_INIT_INT	C	Override	
F_UTIL	FORTRAN	Append	In the code file outside of any routine. Used to insert any user defined functions and sub-routines
C_UTIL	C	Append	
F_UTIL_INT	FORTRAN	Override	
C_UTIL_INT	C	Override	

performed at run time, no overhead is added to the running time to affect the efficiency of the code. Furthermore, the generated code is guaranteed to be correct and adapts easily to any changes to the chemical mechanism. The KPP language used to describe chemical mechanisms is further extended to allow specification of initial values and integration options, including the capability to select the integration method and the integration driver. Special data structures are generated to account for sparsity in the Jacobian.

KPP is written in an expandable way such that new numerical routines can be easily integrated. Moreover, the KPP language itself can be easily extended.

KPP has been proved useful to many researchers for chemical model development. Since the KPP can be easily configured, it can also be used as a learning tool in general chemistry, as well as a broad range of other courses involving chemical simulations.

Future work will expand the KPP capabilities to generate code optimized for vector and parallel architectures; to generate code for the forward and adjoint sensitivity analysis; and to extend the sets of input and output languages (FORTRAN 90, MATLAB, and JAVA).

Acknowledgements

The work of A. Sandu was supported in part by the NSF CAREER award ACI-0093139. The authors wish to thank to anonymous referees for their suggestions which strengthened this paper.

Appendix A: The kinetic description file

The KPP language allows us to specify the chemical equations, the initial values of each of the species

involved and the integration parameters in a file called KPP kinetic description file. For the stratospheric model described by the Chapman system, the KPP kinetic description file is as follows.

```

The Kinetic Description File small_strato.def.

#include small_strato.spc
#include small_strato.eqn

#USE FORTRAN      {Output Language}
#DOUBLE ON        {Double Precision}
#INTEGRATOR rodas3
#DRIVER           general

#LOOKATALL                {File Output}
#MONITOR 03;N;02;0;N0;01D;N02; {Screen Output}

#INITVALUES    {Initial Values}

CFactor = 1.0; {Conversion Factor}
01D = 9.906E+01 ; 0  = 6.624E+08 ;
03  = 5.326E+11 ; 02 = 1.697E+16 ;
N0  = 8.725E+08 ; N02 = 2.240E+08 ;
M   = 8.120E+16 ;

#INLINE F_INIT
    TSTART = (12*3600)
    TEND = TSTART + (3*24*3600)
    DT = 0.25*3600
    TEMP = 270
#ENDINLINE

```

The Kinetic Description File *small_strato.def*:

Appendix B: The Species Description File

The Species Description File *small_strato.spc*:

The Species Description File *small_strato.spc*:

```
#INCLUDE atoms {Use the predefined atom list}

#DEFVAR          { Variable Concentration Species}
O   = 0;          { Oxygen atomic ground state }
O1D = 0;          { Oxygen atomic excited state }
O3  = 0 + 0 + 0;  { Ozone }
NO  = N + 0;      { Nitric oxide }
NO2 = N + 0 + 0;  { Nitrogen dioxide }

#DEFFIX          { Fixed Concentration Species}
O2  = 0 + 0;      { Molecular oxygen }
M   = ignore;     { Its composition unimportant }
```

Appendix C: The Equations Description File

The Equations Description File *small_strato.eqn*:The Equations Description File *small_strato.eqn*:

```
#EQUATIONS { Small Stratospheric Mechanism }

{ 1. } O2 + hv = 2O : 2.643E-10*SUN*SUN*SUN;
{ 2. } O + O2 = O3 : 8.018E-17;
{ 3. } O3 + hv = O + O2 : 6.120E-04*SUN;
{ 4. } O + O3 = 2O2 : 1.576E-15;
{ 5. } O3 + hv = O1D + O2 : 1.070E-03*SUN*SUN;
{ 6. } O1D + M = O + M : 7.110E-11;
{ 7. } O1D + O3 = 2O2 : 1.200E-10;
{ 8. } NO + O3 = NO2 + O2 : 6.062E-15;
{ 9. } NO2 + O = NO + O2 : 1.069E-11;
{10. } NO2 + hv = NO + O : 1.289E-02*SUN;
```

Appendix D: The Integrator Description File

The Integrator Definition File *rodas3.def*:The Integrator Definition File *rodas3.def*:

```
#SETVAR RAD_SPEC { all radicals considered variables }
#FUNCTION AGGREGATE { normal function form }
#JACOBIAN SPARSE { represent Jacobian sparsity ... }
#SPARSEDATA LU_CROW { ... in compressed row format }
#INTFILE rodas3 { integrator is in file rodas3.f }
#INLINE F_DECL_INT
  INTEGER ISNOTAUTONOM
  COMMON /GDATA/ ISNOTAUTONOM
  REAL*8 STEPSTART
  COMMON /GDATA/ STEPSTART
#ENDINLINE
#INLINE F_INIT_INT
  STEPMIN=0.0001
  STEPMAX=60.
  ISNOTAUTONOM=1
  STEPSTART=STEPSMIN
#ENDINLINE
```

References

- Atkinson, R. D., Baulch, D. L., Cox, R. A., Hampson, R. F., Jr, Kerr, J. A. & Troe, J. (1989). Evaluated kinetic and photochemical data for atmospheric chemistry. *Journal of Chemical Kinetics* 21, 115–190.
- Carmichael, G. R., Peters, L. & Kitada, T. (1986). A second generation model for regional-scale transport chemistry deposition. *Atmospheric Environment* 20, 173–188.
- Carter, W.P.L. (1988). Documentation of the SAPRC atmospheric photochemical mechanism preparation and emissions processing programs for implementation in airshed models. *Report to the California Air Resources Board, Contract No. A5-122-32*. <http://pah.cert.ncr.edu/carter/bycarter.htm>.
- Carter, W.P.L. (1999). Documentation of the SAPRC-99 chemical mechanism for VOC reactivity assessment. *Draft report to the California Air Resources Board* <http://cert.ucr.edu/carter-absts.htm>.
- Damian, V. (1998). Computational tools for air quality modeling Ph.D. thesis, Department of Computer Science, University of Iowa.
- Gery, M. W., Whitten, G. Z., Killus, J. P. & Dodge, M. C. (1989). A photochemical kinetics mechanism for urban and regional scale computer modeling. *Journal of Geophysical Research* 94, 12925–12956.
- Jacobson, M. Z. (1998). Improvement of SMVGEAR II on vector and scalar machines through absolute error tolerance control. *Atmospheric Environment* 32, 791–796.
- Jacobson, M. Z. & Turco, R. P. (1994). SMVGEAR: a sparse-matrix, vectorized Gear code for atmospheric models. *Atmospheric Environment* 28A, 273–284.
- Kee, R.J., Rupley, F.M., & Miller, J.A. (1989). CHEMKIN II: a FORTRAN package for the analysis of gas phase chemical kinetics, Technical report, Sandia National Laboratory, Livermore, CA.
- Lurmann, F. W., Loyd, A. C. & Atkinson, R. (1986). A chemical mechanism for use in long-range transport/acid deposition computer modeling. *Journal of Geophysical Research* 91, 10905–10936.
- Nowak, U. (1982). A short user's guide to LARKIN, Technical report, Konrad-Zuse-Zentrum fuer, Informationstechnik Berlin.
- Sandu, A., Potra, F. A., Damian, V. & Carmichael, G. R. (1996). Efficient implementation of fully implicit methods for atmospheric chemistry. *Journal of Computational Physics* 129, 101–110.
- Sandu, A., Blom, J. G., Spee, E., Verwer, J. G., Potra, F. A. & Carmichael, G. R. (1997). Benchmarking stiff ODE solvers for atmospheric chemistry equations II—Rosenbrock Solvers. *Atmospheric Environment* 31, 3459–3472.
- Sandu, A., van Loon, M., Potra, F. A., Carmichael, G. R. & Verwer, J. G. (1997). Benchmarking stiff ODE solvers for atmospheric chemistry equations I—implicit vs. explicit. *Atmospheric Environment* 31, 3151–3166.