PVODE, An ODE Solver for Parallel Computers

George D. Byrne
Illinois Institute of Technology

Alan C. Hindmarsh
Lawrence Livermore National Laboratory

Center for Applied Scientific Computing

UCRL-JC-132361, Rev. 1
May 1999

PVODE, AN ODE SOLVER FOR PARALLEL COMPUTERS

GEORGE D. BYRNE AND ALAN C. HINDMARSH

1. Summary. PVODE is a general purpose solver for ordinary differential equation (ODE) systems, which implements methods for both stiff and non-stiff systems. The code is designed for Single Program Multiple Data environments. It is written in ANSI standard C, with a highly modular structure. The version being distributed uses the MPI (Message-Passing Interface) system for communication. In the stiff case, PVODE uses a Backward Differentiation Formula method combined with preconditioned GMRES iteration. Parallelism is achieved by distributing the ODE solution vector into user-specified segments, and parallelizing a set of vector kernels accordingly. For PDE-based ODE systems, we provide a module which generates a band block-diagonal preconditioner for use with the GMRES iteration. We also provide a set of interfaces to accommodate Fortran applications. The paper includes a stiff example problem, and test results on a Cray-T3D with three different message-passing systems. PVODE is publicly available.

2. Introduction. PVODE is a general purpose ordinary differential equation (ODE) solver for stiff and nonstiff ODEs. It is based on CVODE (Cohen and Hindmarsh, 1994) and (Cohen and Hindmarsh, 1996), which is written in ANSI-standard C. PVODE uses MPI (Message-Passing Interface) (Gropp et al., 1994) and a revised version of the vector module in CVODE to achieve parallelism and portability. PVODE is intended for the SPMD (Single Program Multiple Data) environment with distributed memory, in which all vectors are identically distributed across processors. In particular, the vector module is designed to help the user assign a contiguous segment of a given vector to each of the processors for parallel computation. The idea is for each processor to solve a certain fixed subset of the ODEs.

To better understand PVODE, we first need to understand CVODE and its historical background. The ODE solver CVODE, which was written by Cohen and Hindmarsh, combines features of two earlier Fortran codes, VODE (Brown et al., 1989) and VODPK (Byrne, 1992). Those two codes were written by Brown, Byrne, and Hindmarsh. Both use variable-coefficient multistep integration methods, and address both stiff and nonstiff systems. (Stiffness is characterized by the presence of one or more strong damping modes in which the minimum associated time constant is much less than the time scale of the solution of interest.) In the stiff case, VODE uses direct linear algebra techniques to solve the underlying dense or banded linear systems of equations, in conjunction with a modified Newton method. On the other hand, VODPK uses a preconditioned Krylov iterative method (Brown and Hindmarsh, 1989) to solve the underlying linear system. User-supplied preconditioners directly address the dominant source of stiffness. Consequently, CVODE implements *both* the direct and iterative methods. Currently, with regard to the nonlinear and linear system solution, PVODE has three method options available: functional iteration, Newton iteration with a diagonal approximate Jacobian, and Newton iteration with the iterative method SPGMR (Scaled Preconditioned Generalized Minimal Residual method). Both CVODE and PVODE are written in such a way that other linear algebraic techniques could be easily incorporated, since the code is written with a layer of linear system solver modules that is

isolated, as far as possible, from the rest of the code. Further, the code is structured so that it can readily be converted from double precision to single precision. This precludes the maintenance of two versions of PVODE.

PVODE has been run on an IBM SP2, a Cray-T3D and Cray-T3E, and a cluster of Sun workstations. It is currently being used in a simulation of tokamak edge plasmas at LLNL. Recently, the PVODE solver was incorporated into the PETSc package (Portable Extensible Toolkit for Scientific computation) (Balay et al., 1996) developed at Argonne.

The remainder of this paper is organized as follows: Section 3 sets the mathematical notation and summarizes the basic methods. Section 4 summarizes the organization of the PVODE solver, while Section 5 summarizes its usage. Section 6 describes a preconditioner module, and Section 7 describes a set of Fortran/C interfaces. Section 8 describes an example problem and gives some test results. Following that are comments on availability and conclusions.

3. Mathematical Considerations. PVODE solves initial-value problems (IVPs) for systems of ODEs. Such problems can be stated as

$$\dot{y} = f(t,\, y), \qquad y(t_0) = y_0, \qquad y \in \mathbf{R}^N \ , \tag{1}$$

where $\dot{y} = dy/dt$ and $\mathbf{R}^N$ is the real N-dimensional vector space. That is, (1) represents a system of N coupled ordinary differential equations and their initial conditions at some t_0. The dependent variable is y and the independent variable is t. The independent variable need not appear explicitly in the N-vector-valued function f.

The IVP is solved by one of two numerical methods. These are the backward differentiation formula (BDF) and the Adams-Moulton formula. Both are implemented in a variable-stepsize, variable-order form. The BDF uses a fixed-leading-coefficient form. These formulas can both be represented by a linear multistep formula

$$\sum_{i=0}^{K_1} \alpha_{n,i} y_{n-i} + h_n \sum_{i=0}^{K_2} \beta_{n,i} \dot{y}_{n-i} = 0 \ , \tag{2}$$

where the N-vector y_n is the computed approximation to $y(t_n)$, the exact solution of (1) at t_n. The stepsize is $h_n = t_n - t_{n-1}$. The coefficients $\alpha_{n,i}$ and $\beta_{n,i}$ are uniquely determined by the particular integration formula, the history of the stepsize, and the normalization $\alpha_{n,0} = -1$. The Adams-Moulton formula is recommended for nonstiff ODEs and is represented by (2) with $K_1 = 1$ and $K_2 = q - 1$. The order of this formula is q and its values range from 1 through 12. For stiff ODEs, BDF should be selected and is represented by (2) with $K_1 = q$ and $K_2 = 0$. For BDF, the order q may take on values from 1 through 5. In the case of either formula, the integration begins with $q = 1$, and after that q varies automatically and dynamically.

For either BDF or the Adams formula, $\dot{y}_n$ denotes $f(t_n,\, y_n)$. That is, (2) is an implicit formula, and the nonlinear equation

$$\begin{aligned} G(y_n) &\equiv y_n - h_n \beta_{n,0} f(t_n, y_n) - a_n = 0 \\ \text{where} \quad a_n &\equiv \sum_{i>0} (\alpha_{n,i} y_{n-i} + h_n \beta_{n,i} \dot{y}_{n-i}) \end{aligned}$$

must be solved for y_n at each time step. For nonstiff problems, functional (or fixpoint) iteration is normally used and does not require the solution of a linear system of equations. For stiff problems, a Newton iteration is used and for each iteration an underlying linear system must be solved. This linear system of equations has the form

$$M[y_{n(m+1)} - y_{n(m)}] = -G(y_{n(m)}) \ , \tag{3}$$

where $y_{n(m)}$ is the mth approximation to y_n, and M approximates $\partial G/\partial y$:

$$M \approx I - \gamma J, \quad J = \frac{\partial f}{\partial y}, \quad \gamma = h_n \beta_{n,0} \ .$$

At present, aside from the diagonal Jacobian approximation, the only option implemented in PVODE for solving the linear systems (3) is the iterative method SPGMR (Scaled, Preconditioned GMRES) (Brown and Hindmarsh, 1989), which is a Krylov subspace method. In most cases, performance of SPGMR is improved by a user-supplied preconditioner. The user may precondition the system on the left, on the right, on both the left and right, or use no preconditioner. The SPGMR solver provides for the saving and conditional re-use of Jacobian data that may be involved in the preconditioner.

The integrator computes an estimate E_n of the local error at each time step, and strives to satisfy the following inequality

$$\|E_n\|_{rms,w} < 1 \ . \tag{4}$$

Here the weighted root-mean-square norm is defined by

$$\|E_n\|_{rms,w} = \left[\sum_{i=1}^{N} \frac{1}{N} \left(w_i E_{n,i} \right)^2 \right]^{1/2}$$

where $E_{n,i}$ denotes the ith component of E_n, and the ith component of the weight vector is

$$w_i = \frac{1}{rtol|y_i| + atol_i} \ .$$

This permits an arbitrary combination of relative and absolute error control. The user-specified relative error tolerance is the scalar $rtol$ and the user-specified absolute error tolerance is $atol$ which may be an N-vector (as indicated above) or a scalar. The value for $rtol$ indicates the number of digits of relative accuracy for a single time step. The specified value for $atol_i$ indicates the values of the corresponding component of the solution vector which may be thought of as being at the noise level. In particular, if we set $atol_i = rtol \cdot floor_i$ then $floor_i$ represents the floor value for the ith component of the solution and is that magnitude of the component for which there is a crossover from relative error control to absolute error control. Since these tolerances define the allowed error per step, they should be chosen conservatively. Experience indicates that a conservative choice yields a more economical solution than error tolerances that are too large.

During the course of the integration, PVODE will vary both the stepsize h_n and the order q in an attempt to produce a solution with the minimum number of steps, but always

subject to the local error test (4). Normally, PVODE takes steps until a user-defined output value $t = tout$ is overtaken, and then computes $y(tout)$ by interpolation. See (Brown et al., 1989) for details.

In most PVODE applications of interest, the appropriate method choices will be BDF, Newton iteration, and SPGMR.

4. Code Organization. One can visualize PVODE as being organized in layers, as shown in Fig. 1. The user's main program resides at the top level. This program makes various initialization calls, and calls the core integrator CVode, which carries out the integration steps. Of course, the user's main program also manages input/output. At the next level down, the core integrator CVode manages the time integration, and is independent of the linear system method. CVode calls the user-supplied function `f` and accesses the linear system solver. At the third level, the linear system solver CVSPGMR can be found, along with the approximate diagonal solver CVDIAG. Actually, CVSPGMR calls a generic solver for the SPGMR method, consisting of modules SPGMR and ITERATIV. CVSPGMR also accesses the user-supplied preconditioner solve routine, if specified, and possibly also a user-supplied routine that computes and preprocesses the preconditioner by way of the Jacobian matrix or an approximation to it. Other linear system solvers may be added to the package in the future. Such additions will be independent of the core integrator and CVSPGMR. Several supporting modules reside at the fourth level. The LLNLTYPS module defines types `real`, `integer`, and `boole` (boolean), and facilitates changing the precision of the arithmetic in the package from double to single, or the reverse. The LLNLMATH module specifies power functions and provides a function to compute the machine unit roundoff. Finally, the NVECTOR module is discussed below.

The key to moving from the sequential computing environment to the parallel computing environment lies in the NVECTOR module. The idea is to distribute the system of ODEs over the several processors so that each processor is solving a contiguous subset of the system. This is achieved through the NVECTOR module, which handles all calculations on N-vectors in a distributed manner. For any vector operation, each processor performs the operation on its contiguous elements of the input vectors, of length (say) `Nlocal`, followed by a global reduction operation where needed. In this way, vector calculations can be performed simultaneously with each processor working on its block of the vector. Vector kernels are designed to be used in a straightforward way for various vector operations that require the use of the entire distributed N-vector. These kernels include dot products, weighted root-mean-square norms, linear sums, and so on. The key lies in standardizing the interface to the vector kernels without referring directly to the underlying vector structure. This is accomplished through abstract data types that describe the machine environment data block (type `machEnvType`) and all N-vectors (type `N_Vector`). Functions to define a block of machine-dependent information and to free that block of information are also included in the vector module.

While all N-vectors are distributed across the processors, PVODE itself does not deal with distributed matrices. This is done (if at all) by the user-supplied routines that handle the preconditioner matrix.

The version of PVODE described so far uses the MPI (Message-Passing Interface) system

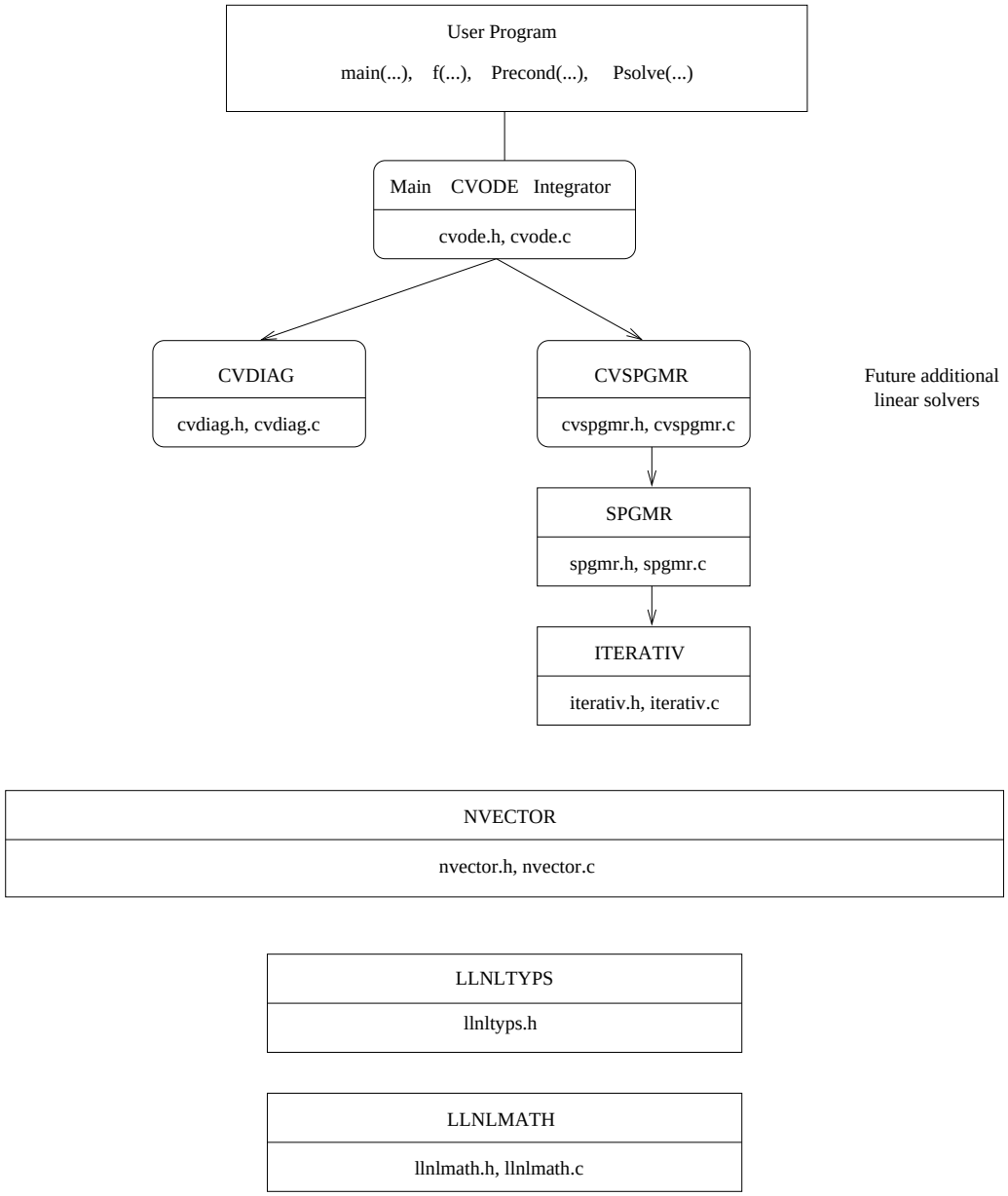

FIG. 1. *Overall structure of the PVODE package. Modules comprising the central solver are distinguished by rounded boxes, while the user program, generic linear solvers, and auxiliary modules are in unrounded boxes.*

(Gropp et al., 1994) for all inter-processor communication. This achieves a high degree of portability, since MPI is becoming widely accepted as a standard for message-passing software. In addition, however, we have prepared a version for the Cray-T3D and -T3E using the Cray Shared Memory (SHMEM) Library. This involves a separate version of the vector module based on SHMEM instead of MPI.

For a different parallel computing environment, some rewriting of the vector module could allow the use of other specific machine-dependent instructions.

5. Using PVODE. This section briefly outlines the use of PVODE, in terms of the header files, the layout of the user's main program, user-supplied functions, and use with C++. For a complete user document, the reader should refer to the report (Byrne and Hindmarsh, 1998). For further details not specific to the parallel extensions, the reader should see the *CVODE User Guide* (Cohen and Hindmarsh, 1994). Also, a sample program is included in the PVODE package and is intended to serve as a template.

The calling program must include several header files so that various macros and data types can be used. Four header files are always required: `llnltyps.h`, `cvode.h`, `nvector.h`, and `mpi.h`. These files define several types and macros, set various constants, and provide function prototypes.

If the user chooses Newton iteration together with the linear system solver SPGMR, then (minimally) CVODE will require a header file `cvspgmr.h` for the Krylov solver SPGMR in the context of PVODE. This in turn includes a header file which enumerates the kind of preconditioning and the choices for the Gram-Schmidt process.

The user's program must have the following steps, in the order indicated:

- Initialize MPI if used by the user's program.
- Set `Nlocal`, the length of the local vector; the global vector length `neq` (the problem size N); and the active set of processors.
- Call `PVecInitMPI` to initialize the NVECTOR module.
- Set the vector `y` of initial values.
- Call `CVodeMalloc` to allocate CVODE's internal memory and initialize CVODE.
- Call `CVSpgmr` if Newton iteration and SPGMR were chosen.
- Call `CVode` for each point at which output is desired. An input parameter specifies whether to overshoot a user-defined point `t = tout` and interpolate, or take a single step and return.
- Deallocate memory for the vector `y` upon completion of the integration.
- Call `CVodeFree` to free the memory allocated for CVODE.
- Call `PVecFreeMPI` to free machine-dependent data.

Several optional inputs and optional outputs are also available through some of these calls.

The user-supplied routines consist of one function defining the ODE, and (optionally) one or two functions that define the preconditioner for use in the SPGMR algorithm. The first of these C functions defines f in (1) and must be of type `RhsFn`, which specifies the call sequence that the user's `f` function must adhere to.

If preconditioning is used, then the user must provide a C function to solve the linear system $Pz = r$, where P may be either a left or a right preconditioner matrix. This C function must be of type `CVSpgmrPSolveFn`. If this function is to use preprocessed Jacobian-

related data, then another routine, of type `CVSpgmrPrecondFn`, must be supplied to do the evaluation and preprocessing involved.

PVODE is written in a manner that permits it to be used by applications written in C++ as well as in C. For this purpose, each PVODE header file is wrapped with conditionally compiled lines reading `extern "C" { ... }`, conditional on the variable `__cplusplus` being defined. This directive causes the C++ compiler to use C-style names when compiling the function prototypes encountered. Users with C++ applications should also be aware that we have defined, in `llnltyps.h`, a boolean variable type, `boole`, since C has no such type.

6. A Band-Block-Diagonal Preconditioner Module. A principal reason for using a parallel ODE solver such as PVODE lies in the solution of partial differential equations (PDEs). Moreover, the use of a Krylov iterative method for the solution of many such problems is motivated by the nature of the underlying linear system of equations (3) that must be solved at each time step. The linear algebraic system is large, sparse, and often structured. However, if a Krylov iterative method is to be effective in this setting, then a nontrivial preconditioner needs to be used. Otherwise, the rate of convergence of the Krylov iterative method is usually unacceptably slow. Unfortunately, effective preconditioners tend to be problem-specific.

However, we have developed one type of preconditioner that treats a rather broad class of PDE-based problems. It has been successfully used for several realistic, large-scale problems (Hindmarsh and Taylor, 1998) and is included in a software module within the PVODE package. The given PDE system may be discretized by any method, as long as the result is an explicit ODE system of the form (1). However, this semi-discrete system is assumed to have predominantly local coupling, and the ordering of the variables on each processor is assumed to reflect that local coupling. This module then generates a preconditioner that is a block-diagonal matrix with each block being a band matrix. The blocks need not have the same number of super- and sub-diagonals and these numbers may vary from block to block. This Band-Block-Diagonal Preconditioner module is called PVBBDPRE.

To envision these preconditioners, think of the domain of the computational PDE problem as being subdivided into M non-overlapping subdomains. Each of these subdomains is then assigned to one of the M processors to be used to solve the ODE system. The basic idea is to isolate the preconditioning so that it is local to each processor, and also to use a (possibly cheaper) approximate right-hand side function in the calculation of the preconditoner. This requires the specification of a new user-defined function $g(t, y)$ which approximates the function $f(t, y)$ in the ODE system (1). Of course, the user may take $g = f$. Corresponding to the domain decomposition, there is a decomposition of the solution vector y into M disjoint blocks y_m, and a decomposition of g into blocks g_m. The block g_m depends on y_m and also on components of blocks $y_{m'}$ associated with neighboring subdomains (so-called ghost-cell data). Let $\bar{y}_m$ denote y_m augmented with those other components on which g_m depends. Then we have

$$g(t, y) = [g_1(t, \bar{y}_1), g_2(t, \bar{y}_2), \ldots, g_M(t, \bar{y}_M)]^T ,$$

and each of the blocks $g_m(t, \bar{y}_m)$ is uncoupled from the others.

The preconditioner associated with this decomposition has the form

$$P = diag[P_1, P_2, \ldots, P_M] \ ,$$

a block-diagonal matrix with blocks

$$P_m \approx I - \gamma J_m \ ,$$

where J_m is a difference quotient approximation to $\partial g_m / \partial y_m$. This matrix is taken to be banded, with upper and lower half-bandwidths `mu` and `ml`, defined as the number of non-zero diagonals above and below the main diagonal, respectively. However, the true band structure of $\partial g_m / \partial y_m$ is typically larger, and for this reason another pair of half-bandwidths, `mudq` and `mldq`, is specified for use in the difference quotient approximation procedure. Thus J_m is computed using `mudq + mldq + 2` evaluations of g_m, but a band matrix of bandwidth `mu + ml + 1` is retained. Neither of these pairs, (`mu, ml`) or (`mudq, mldq`), need be the true values of the half-bandwidths of $\partial g_m / \partial y_m$, if smaller values provide a more efficient preconditioner. Also, they need not be the same on every processor.

The solution of the complete linear system

$$Px = b$$

reduces to solving each of the equations

$$P_m x_m = b_m \ , \tag{5}$$

and this is done by banded LU factorization of P_m followed by a banded backsolve.

To use this PVBBDPRE module, the user must supply two functions which the module calls to construct P. These are in addition to the user-supplied right-hand side function `f`. The first of these functions computes the local version of $g(t, y)$, while the second performs all inter-processor communications necessary for the first. The elements needed in the user's calling program are described in (Byrne and Hindmarsh, 1998).

Similar block-diagonal preconditioners could be considered with different treatment of the blocks P_m. For example, incomplete LU factorization or an iterative method could be used instead of banded LU factorization. Generally, iterative methods become preferable over direct methods for PDE-based systems (5) as the size `Nlocal` grows large. Also, a more scalable linear system method such as Algebraic Multigrid would provide an attractive alternative approach for the preconditioner to Eq. (3). These issues have engendered active areas of research in themselves.

7. The Fortran/C Interface Package. We anticipate that many users of PVODE will work from existing Fortran application programs. To accommodate them, we have provided a set of interface routines that make the required connections to PVODE with a minimum of changes to the application programs. Specifically, a Fortran/C interface package called FPVODE is a collection of C language functions and header files which enables the user to write a main program and all user-supplied subroutines in Fortran and to use the C language PVODE package. This package entails some compromises in portability, but we

have kept these to a minimum by requiring fixed names for user-supplied routines, and by using a name-mapping scheme to set the names of externals in the Fortran/C linkages. The latter depends on two parameters, set in a small header file, which determine whether the Fortran external names are to be in upper case and whether they are to have an underscore character prefix.

Details on the use of these routines can be found in the header file `fpvode.h` and in the user documentation (Byrne and Hindmarsh, 1998). The functions which are callable from the user's Fortran program interface with `PVecInitMPI, CVodeMalloc, CVSpgmr, CVode, CVodeFree`, and `PVecFreeMPI`.

The user-supplied Fortran subroutines have fixed names, which are case-sensitive. These subroutines define the function f, the right-hand side function of the system of ODEs, solve the preconditioner equation (required if preconditioning is used), and compute the preconditioner (required if preconditioning involves pre-computed matrix data).

A similar interface package, called FPVBBD, has been written for the block banded preconditioner module and works in conjunction with the FPVODE interface package. There are three additional user-callable functions in the package and the user must supply two additional Fortran subroutines.

8. Example Problem - A Stiff PDE System. We illustrate the capabilities of PVODE with a test problem that involves the method of lines solution of a system of stiff partial differential equations (PDEs) wherein finite differences are used to reduce the PDEs to a system of ODEs. The problem is based on a two-dimensional system of two PDEs involving diurnal kinetics, advection, and diffusion. The equations represent a simplified model for the transport, production, and loss of the oxygen singlet and ozone in the upper atmosphere. The PDEs can be written as follows:

$$\frac{\partial c^i}{\partial t} = K_h \frac{\partial^2 c^i}{\partial x^2} + V\frac{\partial c^i}{\partial x} + \frac{\partial}{\partial y}\left[K_v(y)\frac{\partial c^i}{\partial y}\right] + R^i(c^1, c^2, t) \quad (i = 1, 2)\ , \tag{6}$$

where the superscript i distinguishes the chemical species, and the reaction terms are given by

$$\begin{aligned} R^1(c^1, c^2, t) &= -q_1 c^1 c^3 - q_2 c^1 c^2 + 2q_3(t)c^3 + q_4(t)c^2 \\ R^2(c^1, c^2, t) &= q_1 c^1 c^3 - q_2 c^1 c^2 - q_4(t)c^2\ . \end{aligned}$$

The spatial domain is the square $0 \le x \le 20,\ 30 \le y \le 50$. The various coefficients for this problem have the following values: $K_h = 4.0 \cdot 10^{-6},\ V = 10^{-3},\ K_v = 10^{-8}\exp(y/5),\ q_1 = 1.63 \cdot 10^{-16},\ q_2 = 4.66 \cdot 10^{-16},\ c^3 = 3.7 \cdot 10^{16}$, and the diurnal rate constants are defined as follows:

$$q_i(t) = \left\{ \begin{array}{l} \exp[-a_i/\sin\omega t], \ \text{ for } \sin\omega t > 0 \\ 0, \ \text{ for } \sin\omega t \le 0 \end{array} \right\} \quad (i = 3, 4)\ ,$$

where $\omega = \pi/43200,\ a_3 = 22.62,\ a_4 = 7.601$. The time interval of integration is $[0, 86400]$, representing 24 hours measured in seconds.

Homogeneous Neumann boundary conditions are imposed on each boundary and the initial conditions are

$$\begin{aligned} c^1(x,y,0) &= 10^6\alpha(x)\beta(y), \quad c^2(x,y,0) = 10^{12}\alpha(x)\beta(y) \\ \alpha(x) &= 1-(0.1x-1)^2+(0.1x-1)^4/2 \\ \beta(y) &= .75+.25\tanh(10y-400)\ . \end{aligned}$$

The function $\beta(y)$ adds some realistic complexity to the solution by introducing a fairly steep front in the vertical direction of the initial profile.This function rises from about 0.5 to about 1.0 over a length of about 0.2 on the y axis (i.e. 1/100 of the total span in y). This vertical variation, together with the horizonatal advection and diffusion in the problem, demands a fairly fine spatial mesh to achieve acceptable resolution.

We discretize the PDE system as follows. The domain of the PDE is discretized with MX and MY uniformly spaced grid points on the x and y axes, respectively. The semi-discrete form for the two PDEs (6) constitute a pair of ODEs at each spatial node, giving an ODE system $\dot{u} = f(t,u)$ of size $N = 2 \cdot MX \cdot MY$. The differencing scheme is central except for the advection terms $V\partial c^i/\partial x$, where it is biased-upwind. More specifically, the ODEs in the system are

$$\begin{aligned} \frac{dc^i_{jk}}{dt} &= K_h\left[\frac{c^i_{j+1,k}-2c^i_{jk}+c^i_{j-1,k}}{\Delta x^2}\right]+V\left[\frac{(3/2)c^i_{j+1,k}-c^i_{jk}-(1/2)c^i_{j-1,k}}{2\Delta x}\right] \\ &\quad +\frac{K_{v,k+1/2}\left(c^i_{j,k+1}-c^i_{jk}\right)-K_{v,k-1/2}\left(c^i_{jk}-c^i_{j,k-1}\right)}{\Delta x^2}+R^i(c^1_{jk},c^2_{jk},t) \\ &\quad (i=1,2;\ 1\le j\le MX;\ 1\le k\le MY)\ , \end{aligned}$$

where $c^i_{jk} \simeq c^i(x_j,y_k,t)$, $\Delta x = 20/(MX-1)$, $\Delta y = 20/(MY-1)$, $x_j = (j-1)\Delta x$, $y_k = 30+(k-1)\Delta y$, and $K_{v,k\pm1/2} = [K_v(y_k)+K_v(y_{k\pm1})]/2$. It is to be understood that the boundary conditions are applied in discrete form, by defining $c^i_{0,k} = c^i_{2,k}$ for the left edge, and likewise for the other three edges.

For this example, we regard the processors as being laid out in a rectangle, and each processor being assigned a subgrid of size `MXSUB` $\times$ `MYSUB` of the $x-y$ grid. If there are `NPEX` processors in the x direction and `NPEY` processors in the y direction, then the overall grid size is `MX` $\times$ `MY` with `MX = NPEX`·`MXSUB` and `MY = NPEY`·`MYSUB`. To compute f in this setting, the processors pass and receive information as follows. The solution components for the bottom row of grid points in the current processor are sent to the processor below it, and the solution components for the top row of grid points in the current processor are sent to the processor above it. Similarly, the components for the leftmost and rightmost columns of grid points are sent from the current processor to the processors to its left and right. If this is the first or last processor in either direction, then message-passing is bypassed for that direction. In MPI terminology, the message-passing is done with blocking sends, non-blocking receives, and receive-waiting. For clarity of illustration, we have used a simple, uniform discretization and domain decomposition, but any other methods for these steps could be applied, including the use of automatic domain decomposition packages, as long as the discretization results in an explicit ODE system of the form (1).

The solution method is BDF with Newton iteration and SPGMR. The left preconditioner is the block-diagonal part of the Newton matrix, with 2×2 blocks, and the corresponding diagonal blocks of the Jacobian are saved each time the preconditioner is generated, for reuse later as directed by the SPGMR solver. (The choice of 2 for the block size is based on the number of species. Larger values may also be effective, but would involve a tradeoff of cost vs speed of convergence.)

A series of tests was performed using this problem. This work was largely carried out by M. Wittman and reported in (Wittman, 1996). To test the scalability of PVODE, we fix the subgrid dimensions at `MXSUB = MYSUB =` 50, so that the local (per-processor) problem size is 5000, while the processor array dimensions, `NPEX` and `NPEY`, are varied. In one (typical) sequence of tests, we fixed `NPEY` = 8 (for a vertical mesh size of `MY` = 400), and set `NPEX` = 8 (`MX` = 400), `NPEX` = 16 (`MX` = 800), and `NPEX` = 32 (`MX` =1600). Thus the largest problem size N is $2 \cdot 400 \cdot 1600 = 1,280,000$. For these tests, we also raise the maximum Krylov dimension, `maxl`, to 10 (from its default value of 5).

For each of the three test cases, the test program was run on a Cray-T3D (256 processors) with each of three different message-passing libraries:

- MPICH: an implemenation of MPI on top of the Chameleon library
- EPCC: an implemenation of MPI by the Edinburgh Parallel Computer Centre
- SHMEM: Cray's Shared Memory Library

The following table gives the run time and selected performance counters for these 9 runs. In all cases, the solutions agreed well with each other, showing expected small variations with grid size. In the table, M-P denotes the message-passing library, RT is the reported run time in CPU seconds, `nst` is the number of time steps, `nfe` is the number of f evaluations, `nni` is the number of nonlinear (Newton) iterations, `nli` is the number of linear (Krylov) iterations, and `npe` is the number of evaluations of the preconditioner.

NPEX	M-P	RT	nst	nfe	nni	nli	npe
8	MPICH	436.	1391	9907	1512	8392	24
8	EPCC	355.	1391	9907	1512	8392	24
8	SHMEM	349.	1999	10,326	2096	8227	34
16	MPICH	676.	2513	14,159	2583	11,573	42
16	EPCC	494.	2513	14,159	2583	11,573	42
16	SHMEM	471.	2513	14,160	2581	11,576	42
32	MPICH	1367.	2536	20,153	2696	17,454	43
32	EPCC	737.	2536	20,153	2696	17,454	43
32	SHMEM	695.	2536	20,121	2694	17,424	43

TABLE 1
PVODE test results vs problem size and message-passing library

Some of the results were as expected, and some were surprising. For a given mesh size, variations in performance counts were small or absent, except for moderate (but still acceptable) variations for SHMEM in the smallest case. The increase in costs with mesh size can be attributed to a decline in the quality of the preconditioner, which neglects most of the

spatial coupling. The preconditioner quality can be inferred from the ratio `nli/nni`, which is the average number of Krylov iterations per Newton iteration. The most interesting (and unexpected) result is the variation of run time with library: SHMEM is the most efficient, but EPCC is a very close second, and MPICH loses considerable efficiency by comparison, as the problem size grows. This means that the highly portable MPI version of PVODE, with an appropriate choice of MPI implementation, is fully competitive with the Cray-specific version using the SHMEM library. While the overall costs do not represent a well-scaled parallel algorithm (because of the preconditioner choice), the cost per function evaluation is quite flat for EPCC and SHMEM, at .033 to .037 (for MPICH it ranges from .044 to .068).

For tests that demonstrate speedup from parallelism, we consider runs with fixed problem size: `MX` = 800, `MY` = 400. Here we also fix the vertical subgrid dimension at `MYSUB` = 50 and the vertical processor array dimension at `NPEY` = 8, but vary the corresponding horizontal sizes. We take `NPEX` = 8, 16, and 32, with `MXSUB` = 100, 50, and 25, respectively. The runs for the three cases and three message-passing libraries all show very good agreement in solution values and performance counts. The run times for EPCC are 947, 494, and 278, showing speedups of 1.92 and 1.78 as the number of processors is doubled (twice). For the SHMEM runs, the times were slightly lower, and the ratios were 1.98 and 1.91. For MPICH, consistent with the earlier runs, the run times were considerably higher, and in fact show speedup ratios of only 1.54 and 1.03.

9. Availability. The PVODE package has been released for general distribution. Interested potential users should contact Alan Hindmarsh, alanh@llnl.gov. The CVODE package, on which PVODE is based, is freely available from the Netlib collection. See for example the listing `cvode.tar.qz` at the web site

`http://www.netlib.org/ode/index.html`

The latter file includes the CVODE User Guide (Cohen and Hindmarsh, 1994) in the form of a PostScript file.

10. Conclusion. We have presented PVODE, a general purpose solver for stiff and non-stiff ODE systems, written in C. Its modular design places all operations on N-vectors into a separate module, where the parallelism is achieved using MPI. Included in the package is a preconditioner module aimed at PDE-based problems, which generates a band block-diagonal preconditioner for use with the GMRES iteration. The package also includes a set of interfaces for Fortran applications. We have illustrated the use and performance of PVODE with a stiff system of size 1.28 million based on a coupled pair of PDEs in two space dimensions. PVODE is available to the public and is currently being applied to problems in plasma physics.

11. Acknowledgments. Research for this paper was performed under the auspices of the U.S. Department of Energy, by Lawrence Livermore National Laboratory under contract W-7405-ENG-48. The work was supported by LDRD, Project 95-ER-036. We are grateful to Dr. Michael Minkoff at Argonne National Laboratory for assistance in the use of the IBM SP2 there.

12. Biographies. *George D. Byrne* is an IIT Professor Emeritus at Illinois Institute of Technology. From 1994 until 1998, he held various positions at IIT, including Chairman of the Department of Mathematics. He was a Visiting Professor at Southern Methodist University during the 1993-1994 academic year and was with Exxon Research and Engineering Company from 1980 until 1993. During the period 1963-1980, he was a member of the faculty at the University of Pittsburgh. He earned his Ph.D. in applied mathematics at Iowa State University in 1963. Dr. Byrne began his collaboration with Dr. Hindmarsh in 1973 at Lawrence Livermore National Laboratory. His interests include numerical methods and software for differential equations, their application, and golf.

Alan C. Hindmarsh is a Mathematician at Lawrence Livermore National Laboratory's Center for Applied Scientific Computing. He received a Ph.D. in mathematics from Stanford University in 1968, and has been at LLNL (Livermore, California) since then. He has published extensively in numerical analysis and mathematical software, and has authored or co-authored numerous widely used software packages. His particular special interests are in methods and software for stiff systems of ordinary differential equations, the application of ODE software to partial differential equations, and related topics.

REFERENCES

Balay, S., Gropp, W., McInnes, L., and Smith, B. 1996. PETSc 2.0 Users Manual. Technical report ANL-95/11, Argonne National Laboratory.

Brown, P. N., Byrne, G. D., and Hindmarsh, A. C. 1989. VODE, a Variable-Coefficient ODE Solver, *SIAM J. Sci. Stat. Comput.* 10(5):1038-1051.

Brown, P. N., and Hindmarsh. A. C. 1989. Reduced Storage Matrix Methods in Stiff ODE Systems, *J. Appl. Math. & Comp.* 31(1):40-91.

Byrne. G. D. 1992. Pragmatic Experiments with Krylov Methods in the Stiff ODE Setting. In *Computational Ordinary Differential Equations*, edited by J. R. Cash and I. Gladwell. Oxford: Oxford University Press, pp. 323-356.

Byrne, G. D., and Hindmarsh, A. C. 1998. User Documentation for PVODE, An ODE Solver for Parallel Computers. Technical report UCRL-ID-130884, Lawrence Livermore National Laboratory.

Cohen, S. D., and Hindmarsh, A. C. 1994. CVODE User Guide. Technical report UCRL-MA-118618: Lawrence Livermore National Laboratory.

Cohen, S. D., and Hindmarsh, A. C. 1996. CVODE, a Stiff/Nonstiff ODE Solver in C. *Computers in Physics*, 10(2):138-143.

Gropp, W., Lusk, E., and Skjellum, A. 1994. *Using MPI: Portable Parallel Programming with the Message-Passing Interface.* Cambridge: MIT Press.

Hindmarsh, A. C., and Taylor, A. G. 1998. PVODE and KINSOL: Parallel Software for Differential and Nonlinear Systems. Technical report UCRL-ID-129739: Lawrence Livermore National Laboratory.

Wittman, M. R. 1996. Testing of PVODE, a Parallel ODE Solver. Technical report UCRL-ID-125562: Lawrence Livermore National Laboratory.