Pergamon

Computers Math. Applic. Vol. 31, No. 9, pp. 43–55, 1996

0898-1221/96 $15.00 + 0.00

S0898-1221(96)00041-7

Experiences with Sparse Matrix Solvers in Parallel ODE Software

J. J. B. DE SWART AND J. G. BLOM
Department of Numerical Mathematics, CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

(Received October 1995; accepted November 1995)

Abstract—The use of implicit methods for numerically solving stiff systems of differential equations requires the solution of systems of nonlinear equations. Normally these are solved by a Newton-type process, in which we have to solve systems of linear equations. The Jacobian of the derivative function determines the structure of the matrices of these linear systems. Since it often occurs that the components of the derivative function only depend on a small number of variables, the system can be considerably sparse. Hence, it can be worth the effort to use a sparse matrix solver instead of a dense LU-decomposition. This paper reports on experiences with the direct sparse matrix solvers MA28 by Duff [1], Y12M by Zlatev *et al.* [2] and one special-purpose matrix solver, all embedded in the parallel ODE solver PSODE by Sommeijer [3].

Keywords—Numerical analysis, Sparse matrices, Newton iteration, Runge-Kutta methods, Parallelism.

1. INTRODUCTION

For solving the stiff initial value problem (IVP)

$$y'(t) = f(t, y(t)), \quad y(t_0) = y_0, \qquad y, f \in \mathbb{R}^d, \quad t_0 \leq t \leq t_e, \tag{1}$$

one of the most powerful methods is an implicit Runge-Kutta (RK) scheme. However, in such a method we have to solve a system of nonlinear equations of dimension sd in every time step. Here, s is the number of stages. This may require a lot of computational effort and for this reason implicit RKs have not been very popular on sequential computers. On parallel computer architectures the costs can be reduced significantly. Several ways of doing this are described in [4–8]. In this paper we will focus on PSODE (Parallel Software for ODEs), described in [3]. PSODE is an implementation of a Radau IIA method with a Newton-type iteration process, which reduces the nonlinear systems to sequences of linear systems of dimension d. Most time in PSODE is spent on the solution of these linear systems, using a dense LU-decomposition, followed by forward-back substitutions. For problems arising in practice it often occurs that the components of the derivative function only depend on a small number of variables. Such problems lead to linear systems of which the matrix is sparse. The goal of this paper is to reduce the linear algebra costs for such problems by using a sparse matrix solver.

The outline of the paper is as follows. Section 2 briefly describes PSODE. Section 3 presents the off-the-shelf sparse matrix solvers MA28 [1] and Y12M [2] and a sparse matrix solver that is designed especially for ODE solvers. An analysis of the influence of the errors made in the

The authors are grateful to B. P. Sommeijer for his careful reading of the manuscript and for suggesting several improvements. The research reported in this paper was supported by STW (Dutch Foundation for Technical Sciences).

numerical solution of the linear systems is given in Section 4. Finally, in Section 5, numerical experiments give insight into the behavior of these matrix solvers.

2. THE PARALLEL ODE SOLVER PSODE

In the following, the m^{th} canonical basis vector of $\mathbb{R}^s$ will be denoted by e_m, and the $m \times m$ identity matrix by I_m.

PSODE is a code based on a parallel method for numerically solving problems of type (1). It is based on the implicit RK method Radau IIA, for which the number of stages s equals 4. Denoting the Radau IIA matrix by A, we have to solve in every time step the sd-dimensional system of nonlinear equations

$$Y_n = (e \otimes I_d)y_{n-1} + h_n(A \otimes I_d)F(Y_n). \tag{2}$$

Here, Y_n is the sd-dimensional stage vector $(Y_{n,i})$ containing approximations to $y(t_{n-1} + c_i h_n)$, $i = 1, \ldots, s$, in which $c = (c_1, \ldots, c_s)^\top$ is the abscissa vector, e is the s-dimensional unit vector $(1, \ldots, 1)^\top$, y_{n-1} is the approximation to the step point value $y(t_{n-1})$, h_n is the stepsize $t_n - t_{n-1}$ and $F(Y_n)$ contains the derivative values $f(t_{n-1}+c_i h_n, Y_{n,i})$. Since $c_s = 1$ for Radau IIA methods, Y_n contains y_n:

$$y_n = \left(e_s^\top \otimes I_d\right) Y_n.$$

Solving (2) by a modified Newton process would yield a sequence of iterates $Y_n^{(0)}, Y_n^{(1)}, Y_n^{(2)}, \ldots$ defined by

$$Y_n^{(0)} = \text{given by some predictor formula,} \tag{3}$$

$$(I_{sd} - h_n(A \otimes J_n))\Delta Y_n^{(j+1)} = -R\left(Y_n^{(j)}\right), \qquad j = 0, 1, 2, \ldots, \tag{4}$$

where J_n is an approximation to the Jacobian of f in (t_{n-1}, y_{n-1}), $\Delta Y_n^{(j+1)} = Y_n^{(j+1)} - Y_n^{(j)}$ and $R(Y_n^{(j)})$ denotes the residual of $Y_n^{(j)}$ with respect to (2); i.e.,

$$R\left(Y_n^{(j)}\right) = Y_n^{(j)} - (e \otimes I_d)y_{n-1} - h_n(A \otimes I_d)F\left(Y_n^{(j)}\right).$$

The process (3),(4) requires the solution of linear systems of dimension sd. Since the approximation J_n does not vary during a time step, we are dealing in every time step with a sequence of linear systems with the same matrix. In practice we often keep the approximation of the Jacobian and the stepsize h_n constant over a number of time steps. Hence, the number of linear systems with the same matrix is frequently a multiple of the number of Newton iterations per time step. This explains our bias to use a direct matrix solver instead of, e.g., an iterative Krylov subspace method; in the latter case we would have to rebuild the Krylov subspace for every new right-hand side, whereas with an LU-decomposition, for every new right-hand side, only the forward-back substitutions have to be performed. However, in an iterative approach, it may be possible to exploit the fact that the linear systems are related; this will be the subject of future research.

The costs of an LU-decomposition of the matrix in (4) are $\mathcal{O}(s^3d^3)$. These costs can be reduced by replacing A with a matrix $D = \operatorname{diag}\{d_1, \ldots, d_s\}$. The linear system of dimension sd is now decoupled into s systems of dimension d:

$$(I_d - h_n d_i J_n))\Delta Y_{n,i}^{(j+1)} = -\left(e_i^\top \otimes I_d\right) R\left(Y_n^{(j)}\right); \qquad i = 1, 2, \ldots, s; \quad j = 0, 1, 2, \ldots. \tag{5}$$

These systems can be solved in parallel if s processors are available: every processor makes an LU-decomposition of $I_d - h_n d_i J_n$ and performs the forward-back substitutions on the right-hand side $-(e_i^\top \otimes I_d)R(Y_n^{(j)})$. The sequential costs on s processors are thus $\mathcal{O}(d^3)$. Notice that it is possible to compute the components of $R(Y_n^{(j)})$ in parallel too: every processor computes $f(Y_{n,i}^{(j)})$,

broadcasts the result to the other $s-1$ processors and receives the remaining part of $F(Y_n^{(j)})$ from the other processors.

In PSODE the matrix D is chosen such that the stiff components of the iteration errors are strongly damped (see [9,10]). We shall refer to (3),(5) as a *simplified Newton process*. PSODE uses this simplified Newton process to solve (2) together with strategies for determining when the Jacobian should be reevaluated, when a new LU-decomposition should be made and how many iterations should be performed in (5). A stepsize control is also included. For details on implementation, we refer the reader to [3].

3. SPARSE MATRIX SOLVERS

A general direct solver for nonsymmetric, sparse linear systems is in most cases based on the Gaussian Elimination process (GE). The main feature that characterizes a direct sparse matrix solver is the pivoting strategy. A balance has to be found between *stability pivoting* and *sparsity pivoting*. Stability pivoting means reordering the matrix to obtain pivots that are relatively large, so that GE becomes numerically stable. The aim of sparsity pivoting is to find a reordering of the matrix such that the number of operations and the fill-in of the reordered matrix are kept small. Here, the fill-in of a matrix M after reordering is defined as follows. Suppose P_1 and P_2 are permutation matrices such that P_1MP_2 denotes the reordered matrix. If

$$P_1MP_2 = LU,$$

where L is lower triangular with $\mathrm{diag}(L) = e$ and U is upper triangular, then the fill-in of P_1MP_2 is the number of nonzeros in the strictly lower triangular part of L + the number of nonzeros in U − the number of nonzeros in M. A well-known strategy to keep fill-in small is to use the *Markowitz criterion* [11]. Suppose that the first k steps of GE have been performed. Let $r_i^{(k)}$ and $c_j^{(k)}$ be the number of nonzero entries in the i^{th} row and j^{th} column of the remaining $(n-k)\times(n-k)$ submatrix, respectively. Then the Markowitz criterion selects as pivot the nonzero element

$$m_{ij}^{(k)} \text{ with } i,j \text{ such that } \left(r_i^{(k)}-1\right)\left(c_j^{(k)}-1\right) \text{ is minimal.}$$

Notice that by this definition, a row or column with only one nonzero entry automatically delivers this entry as pivot. Unfortunately, the Markowitz ordering does not always produce the best ordering. On the other hand, the problem of finding the reordering that really minimizes fill-in is NP-complete [12]. In the sequel we will refer to $(r_i^{(k)}-1)(c_j^{(k)}-1)$ as the *Markowitz number of* $m_{ij}^{(k)}$, where $M^{(k)} = (m_{ij}^{(k)})$ denotes the matrix M after $k-1$ steps of GE.

Another categorization of pivoting strategies distinguishes between regions of the matrix in which the pivot is to be found. The case where the pivot of the k^{th} step in GE is determined in the last $n-k+1$ entries of the k^{th} column is referred to as *partial* pivoting. If the pivot is chosen on the main diagonal, we speak of *diagonal* pivoting. *Complete* pivoting means scanning the whole submatrix $M(k:n,k:n)$ to select the pivot. In a straightforward way one derives combinations of several pivoting strategies. For example, diagonal Markowitz pivoting means selecting in the k^{th} step of GE the nonzero element $m_{ii}^{(k)}$, of which the Markowitz number is minimal.

3.1. MA28

MA28 is a set of Fortran subroutines for sparse unsymmetric linear equations. This code by Duff [1] is part of the Harwell Subroutine Library, which is licensed. However, one may use this code for research purposes. The user can set a parameter u to control bias towards stability pivoting or sparsity pivoting: $u = 1.0$ gives partial stability pivoting, while $u = 0.0$ minimizes fill-in without checking the magnitude of the pivots at all. The sparsity pivoting is performed

by means of the Markowitz criterion. For values of $u \in (0,1)$ a stability control is added. The user supplies the sparse matrix in a one-dimensional array containing the nonzeros. The row and column indices in the sparse matrix are stored in two one-dimensional integer arrays. Since MA28 performs the LU-decomposition and the forward-back substitutions in separate subroutines, it is easy to solve sequences of linear systems with the same matrix by performing only one decomposition.

In the version of MA28 dated 1 January 1984, which we used, common blocks that communicate data between the 17 internal subroutines of the package complicate parallel implementation of the code. One may get around this problem by using more up-to-date versions.

3.2. Y12M

Y12M is a package of Fortran subroutines for the same purpose as MA28. It was developed at the Regional Computing Centre at the University of Copenhagen (RECKU) by Zlatev *et al.* One can obtain the code from Netlib [13]. The complete documentation is in [2]. Although Y12M is similar to MA28, the influence of the user on the choice of the pivoting strategy is different. Although the code selects by itself the mixture between sparsity and stability pivoting, the user can decide where the pivot is to be selected. He can restrict the pivots to the diagonal and it is also possible to choose in how many rows that have least number of nonzero elements the pivotal search is carried out. Again the underlying sparsity pivoting strategy is based on the Markowitz criterion.

3.3. Special Purpose Solver

Our first experiments with Y12M and MA28 suggested that for our test problems the use of stability pivoting had hardly any influence. The experience that stability pivoting is seldom required for solving stiff ODEs was also reported by others; see [12,14–16]. In the next section we give an heuristic explanation of this phenomenon. Therefore, we also implemented a special purpose matrix solver without stability pivoting. In this solver, we use the following strategy:

1. Use only diagonal sparsity pivoting in order to reduce the fill-in of the matrix $I_d - h_n d_i J_n$.
2. Compute the fill-in of the reordered matrix and add the elements that will be made nonzero to the sparse data structure.
3. Perform an Incomplete LU-decomposition (ILU) on the augmented reordered matrix of Step 2.
4. Perform the forward-back substitutions with the reordered right-hand sides.

We reorder the matrix with the diagonal Markowitz strategy. Remember that the sparsity structure of $I_d - h_n d_i J_n$ remains constant over the integration interval, so that Steps 1 and 2 have to be done only once. Since the magnitude of the entries is not involved, this is a symbolic operation. We programmed Steps 1 and 2 in Maple [17].

Step 3 and 4 are performed by modified versions of subroutines of the Sparse Linear Algebra Package (SLAP) that perform ILU as preconditioner. SLAP is written by Greenbaum and Seager (with contributions of several other authors) and is available from Netlib [13]. It uses the compressed row/column format. Notice that the input for these subroutines does not only contain the nonzero elements of the matrix $I_d - h_n d_i J_n$, but also the zero elements that will be made nonzero by the GE process. Consequently, the ILU performed in Step 3 is algebraically equivalent with a complete LU-decomposition. In the sequel, we will refer to this sparse matrix solver as Special Purpose Solver (SPS). Remark that both MA28 and Y12M do not allow the pivoting strategy followed in SPS.

4. ERROR ANALYSIS

In this section, we investigate how the omission of stability pivoting in solving linear systems arising from ODEs, that are solved numerically with the use of a Newton-type process, influences the numerical solution to the ODE.

In the sequel, we use the shorthand notations M for $I_d - h_n d_i J_n$, the short form x_j for $Y_{n,i}^{(j)}$ and $r(x_j)$ for $-(e_i^\top \otimes I_d)R(Y_n^{(j)})$. Denoting the update $x_{j+1} - x_j$ by u_{j+1}, the linear system (5) takes the form

$$M u_{j+1} = r(x_j). \tag{6}$$

Other implicit ODE solvers, e.g., codes based on Backward Differentiation Formulas, lead to linear systems for which the remainder of this section holds as well.

First note that neither the matrix M (if nonsingular) nor its decomposition have influence on the *solution* of the Newton process; they can only affect the rate of convergence to this solution.

Assume that (6) can be solved using GE with some pivoting strategy. Hence, the inverse matrix M^{-1} exists. The omission of stability pivoting may lead to an accidental breakdown, if a diagonal element of M becomes zero. However, a change of stepsize will cure this breakdown. If the convergence of the Newton process stagnates because of pivots which are too small, then the integrator would detect this and restrict the stepsize. The matrix M becomes more diagonal dominant and the pivots using no pivoting or diagonal pivoting would now be relatively larger. Hence, GE becomes more stable. On the other hand, for efficiency reasons, we want the stiff solver to use large steps. Experiments suggest that the increase of step rejections due to the omission of stability pivoting is very modest.

Let us now look in more detail to the error propagation in the Newton process. Assuming that x is such that $r(x) = 0$, that no error is made when solving the linear systems and defining the Newton error by $\delta_j := x_j - x$, we arrive at

$$\begin{aligned} \delta_{j+1} &= \delta_j + M^{-1}(r(x_j) - r(x)) \\ &= \left(I + M^{-1}Q(x)\right)\delta_j + \text{ higher order terms} \\ &= P^{j+1}\delta_0 + \text{ higher order terms.} \end{aligned} \tag{7}$$

Here, $Q(x)$ is the Jacobian of $r(x)$ and $P := I + M^{-1}Q(x)$. However, due to the omission of stability pivoting, we compute instead of $x_1, x_2, \ldots$ the sequence $\widetilde{x}_1, \widetilde{x}_2, \ldots$, that satisfies

$$(M + E)\widetilde{u}_{j+1} = r\left(\widetilde{x}_j\right), \tag{8}$$

where $\widetilde{u}_{j+1} := \widetilde{x}_{j+1} - \widetilde{x}_j$. Notice that we neglected here other rounding errors than those arising in the LU-factorization of M. Defining the linear system error in the j^{th} Newton update by $\epsilon_j := \widetilde{u}_j - \widehat{u}_j$, where $\widehat{u}_j$ is the solution of $M\widehat{u}_j = r(\widetilde{x}_{j-1})$, we arrive at

$$\epsilon_j = -M^{-1}E\,\widetilde{u}_j. \tag{9}$$

Combining the formulas (8),(9),(7) and $\widetilde{\delta}_0 = \delta_0$ yields a formula for the Newton errors in which the linear system errors are taken into account too, described by $\widetilde{\delta}_j := \widetilde{x}_j - x$:

$$\widetilde{\delta}_j = \delta_j + \sum_{k=1}^{j} P^{j-k}\epsilon_k + \text{ higher order terms.} \tag{10}$$

In the sequel, we denote the spectrum and spectral radius of any matrix X by $\sigma(X)$ and $\rho(X)$.

The formulas above lead us to several indications why omitting the stability pivoting works well. First of all, for dissipative systems, it holds that

$$\sigma(J_n) \in \{z \in \mathbb{C} \mid \mathrm{Re}(z) \leq 0\},$$

and consequently, since the diagonal entries $d_i > 0$,

$$\sigma(M) \in \{z \in \mathbb{C} \mid \mathrm{Re}(z) \geq 1\} \quad \text{and} \quad \rho\left(M^{-1}\right) \leq 1.$$

This is a necessary (although not sufficient) condition for M to damp the error matrix E in (9).

Second, for $h \to 0$, the matrix M becomes increasingly diagonal dominant. This means $\|E\| \to 0$. On the other hand, the situation $h \to \infty$ is usually initiated by an ODE-solution that tends to a steady state. Here we expect that $\forall j$, $\widetilde{u}_j \to 0$. In both situations, $\forall k$, $\epsilon_k \to 0$, so $\widetilde{\delta}_j \to \delta_j$.

Our last argument is based on formulas (9) and (10). Normally, Newton iterates are monotonically decreasing:

$$\|\widetilde{u}_k\| < \|\widetilde{u}_j\| \qquad \text{for } k > j.$$

Together with formula (9) this tells us that it is likely that the error matrix E has less influence on ϵ_k as k increases. However, the contribution of ϵ_k in $\widetilde{\delta}_j$ is by means of (10) multiplied by P^{j-k} and thus more damped for small k, since P is a contracting operator if the Newton process converges.

Although we did not give a rigorous proof that the omission of stability pivoting in the solution process of the nonlinear systems arising from ODEs is harmless, we showed in the above that at least a number of necessary conditions are fulfilled.

5. NUMERICAL EXPERIMENTS

5.1. Test Problems

To test how the sparse matrix solvers perform in PSODE we consider three stiff test problems. The first one comes from circuit analysis and describes a ring modulator. It is of dimension 15. Our second test problem has dimension 20 and is the chemical part of an air pollution model. The last problem is the EMEP MSC-W ozone chemistry model of dimension 66. For a more detailed description of these problems we refer the reader to the Appendix of this paper.

In order to see the effect of an increasing problem size on the performance of the sparse matrix solvers we 'cascade' the problems m times as follows. If the original test problems are of the form

$$\widehat{y}'(t) = \widehat{f}(t, \widehat{y}), \quad \widehat{y}(0) = \widehat{y}_0, \qquad \widehat{y} \in \mathbb{R}^{\widehat{d}}, \quad t_0 \leq t \leq t_e,$$

then the resulting 'cascaded' problems are of form (1), where f is defined by

$$f(t,y) = \begin{pmatrix} \widehat{f}(t,\widehat{y}) \\ \vdots \\ \widehat{f}(t,\widehat{y}) \end{pmatrix}, \qquad y = \begin{pmatrix} \widehat{y} \\ \vdots \\ \widehat{y} \end{pmatrix}, \qquad y_0 = \begin{pmatrix} \widehat{y}_0 \\ \vdots \\ \widehat{y}_0 \end{pmatrix},$$

and $d = m\widehat{d}$.

Information on the sparsity of the matrix $I_d - h_n d_i J_n$ is listed in Table 1. As usual, we define the *nonzero ratio* of a matrix to be the number of nonzero entries divided by the total number of entries. In the table, the fill-in before and after reordering with the diagonal Markowitz strategy is specified for $m \in \{1, 2, 3, 10\}$. For the pollution and EMEP problem we see that the fill-in after reordering does not depend linearly on m. The reason for this is that from the elements with the same Markowitz number the last one is chosen as pivot. Consequently, the diagonal blocks of the Jacobian of the 'm times cascaded' problem are not necessarily treated identically by the reordering algorithm.

Table 1. Sparsity characteristics for the three test problems.

Problem ('m times cascaded')	Ring modulator	Pollution problem	EMEP problem
Nonzero ratio	$55/(225m)$	$86/(400m)$	$496/(66^2m)$
Fill-in before reordering	$58m$	$176m$	$2166m$
Fill-in after diagonal Markowitz	$12, 24, 36, 120$	$9, 17, 24, 82$	$87, 171, 260, 861$
Sparsity structure symmetric?	yes	no	no

5.2. Numerical Results

First, we experimented with PSODE using Y12M, MA28 and SPS on a one-processor machine in order to compare the matrix solvers mutually and to see the influence of an increasing nonzero ratio of the Jacobian on the performance. We also listed the results of the dense matrix solver of LINPACK (the routines `dgefa` and `dgesl`, for computing the LU factors and forward-back substitutions, respectively). Tables 2–4 show the results of solving the three test problems. A few remarks with respect to these tables should be made.

- The numerical experiments were done on a Silicon Graphics *Indy* workstation (100 MHz R4000SC), using 64-bit arithmetic. 'CPU' refers to the CPU time in seconds to solve a test problem on this machine.
- m denotes the number of times that the problem is cascaded.
- For MA28, the pivoting parameter u was set equal to 0.5. Other settings of u did not yield significantly better results.
- For Y12M most parameters were set to their default values. Only the drop tolerance was valued 10^{-14} (using the default value 10^{-12}, a breakdown occurred in the pollution problem). Other choices for the parameters that determine the pivoting strategy did not improve the results considerably.
- The integration statistics of PSODE are given by the following:

  scd denoting the minimum number of significant correct digits in the numerical solution in the end point, that is:

$$\text{scd} := \min_{i\in\{1,\ldots,d\}} \left(-\log_{10} \left| \frac{\widetilde{y}_i(t_e) - y_i(t_e)}{y_i(t_e)} \right| \right),$$

  where $\widetilde{y}_i(t_e)$ and $y_i(t_e)$ denote the i^{th} component of the numerical and true solution in the end point, respectively, and d is the dimension of the problem. Since the exact solution to the problems is not known, the numerical solution was compared with the output of a very accurate run.

  Steps refers to the number of time steps (including rejected steps).

  J-eval is the number of evaluations of the analytical Jacobian of the function f.

  LU-dec denotes the number of 'sequential' LU-decompositions of matrices of the form $I_d - h_n d_i J_n$.

  f-eval is the number of 'sequential' evaluations of the function f.

  Here, 'sequential' means that operations on the four stages at the same time step, which can be done in parallel, are counted as one.
- The user of PSODE has to supply the error tolerance, a safe upperbound for $\|y(t)\|_\infty$ on the whole integration interval, and the initial stepsize. For the ring modulator and the pollution problem we set them equal to 10^{-4}, 1 and 10^{-7}, respectively, for the EMEP problem to 10^{-2}, 10^{16} and 1.
- Blank entries in the tables are identical to corresponding entries in the adjacent upper row.

Table 2. Results on ring modulator.

	m	CPU	scd	Steps	J-eval	LU-dec	f-eval
LINPACK	1	24.40	4.20	3129	537	2393	19247
	2	67.13					
	3	130.52					
	10	1145.44					
MA28	1	30.57	4.46	3188	568	2462	19784
	2	59.41	4.06	3155	554	2434	19488
	3	96.21					
	10	306.12					
Y12M	1	24.82	4.78	3173	552	2437	19540
	2	50.13	4.04	3165	552	2452	19486
	3	75.86	4.64	3146	548	2443	19408
	10	258.69	4.31	3159	557	2458	19446
SPS	1	14.35	4.49	3159	550	2428	19390
	2	28.16	4.73	3160	554	2424	19481
	3	42.81	4.62	3131	541	2434	19337
	10	153.38	4.53	3150	551	2418	19422

Table 3. Results on pollution problem.

	m	CPU	scd	Steps	J-eval	LU-dec	f-eval
LINPACK	1	0.59	6.11	46	10	44	299
	2	1.80					
	3	3.68					
	10	37.80					
MA28	1	0.67	6.11	46	10	44	299
	2	1.32					
	3	1.92					
	10	6.42					
Y12M	1	0.54	6.11	46	10	44	299
	2	1.14					
	3	1.62					
	10	5.53					
SPS	1	0.29	6.11	46	10	44	299
	2	0.55					
	3	0.85					
	10	2.98					

We nicely see that the CPU-timings for the sparse matrix solvers are $\mathcal{O}(m)$, whereas for the dense solver of LINPACK they are superlinear. For these three test problems it begins to pay off to use MA28 or Y12M instead of LINPACK for nonzero ratios of less than about 20–25%. For SPS this maximum nonzero ratio is even somewhat larger. SPS performs about 1.75 times more efficiently than Y12M and is about twice as fast as MA28. For the ring modulator and EMEP problem, the four solvers show roughly the same statistics. They slightly depend on m, since the order in which pivots are selected within diagonal blocks of $I_d - h_n d_i J_n$ may differ. For the pollution problem all integration statistics were identical to the LINPACK statistics.

Second, we investigate how the parallel performance of PSODE depends on the solver and on m. We implemented the codes on the Cray C98/4256 at SARA. Since the integration statistics were roughly the same as in Tables 2–4, we only listed the speed-up factors of the runs on four processors compared to the runs in one-processor mode. The Cray C98/4256 is a shared memory computer with four processors. Since we did not have the machine in dedicated mode during

Table 4. Results on EMEP problem.

	m	CPU	scd	Steps	J-eval	LU-dec	f-eval
LINPACK	1	107.27	3.75	594	243	716	4422
	2	404.32	3.84	614	273	722	4516
	3	861.06	3.77	567	260	676	4251
	10	9951.05	3.91	621	260	719	4507
MA28	1	80.36	3.81	645	267	772	4723
	2	142.15	3.56	549	251	660	4132
	3	228.44	3.73	635	250	749	4592
	10	690.34	3.81	530	238	626	3973
Y12M	1	56.66	3.95	574	250	685	4992
	2	116.46	3.75	618	262	738	4553
	3	180.10	3.97	654	262	773	4705
	10	581.59	3.63	589	260	689	4369
SPS	1	33.09	3.11	606	264	722	4499
	2	57.78	3.64	581	246	681	4270
	3	89.77	3.55	605	256	699	4390
	10	352.61	3.89	509	229	643	3924

our experiments (on the average we used 2.5 processors concurrently), we used a tool called ATExpert [18] to predict the speed-up factors on four processors. Table 5 gives these results. Denoting the fraction of the code that can be done in parallel by f_P, the optimal speed-up on N processors according to Amdahl's law is given by the formula $1/(1 - f_P + f_P/N)$. ATExpert produces these optimal speed-up values, based on estimates of the parallel fraction f_P. These values are also listed in Table 5.

Table 5. Speed-up factors for the three test problems.

		Ring modulator		Pollution problem		EMEP problem	
		Predicted speed-up	Optimal speed-up	Predicted speed-up	Optimal speed-up	Predicted speed-up	Optimal speed-up
LINPACK	1	2.2	2.7	2.2	2.5	3.4	3.5
	2	2.6	3.0	2.7	3.0	3.6	3.8
	3	3.0	3.2	3.1	3.3	3.6	3.9
	10	3.6	3.8	3.5	3.8	3.5	3.9
Y12M	1	3.1	3.4	2.8	3.4	3.8	3.8
	2	3.3	3.5	2.8	3.5	3.8	3.8
	3	3.4	3.6	3.5	3.6	3.8	3.8
	10	3.5	3.7	3.6	3.7	3.8	3.9
SPS	1	2.4	2.8	2.4	2.7	3.6	3.7
	2	2.6	2.9	2.7	2.9	3.7	3.8
	3	2.8	3.1	2.5	3.0	3.8	3.8
	10	3.3	3.4	3.0	3.3	3.7	3.8

We compiled the codes using the flags `-dp`, `-ZP` and `-Wu"-p"`. The environment variables `NCPUS` and `MP_DEDICATED` were valued 4 and 1, respectively. We refer to the Cray C90 documentation [19] for the specification of these settings. We did not include results for MA28, since for a parallel implementation of this code, one would have to get rid of the common blocks. Table 5 confirms the expectation that the speed-up factors grow for increasing problem sizes. The predicted speed-up factors do not always increase monotonically with m. This can be explained by the fact that the results of ATExpert are based on a varying number of processors. For PSODE with

LINPACK the optimized routines `sgefa` and `sgesl` of the Cray library were used. The relatively fast performance of the resulting code leads to smaller speed-up factors. The speed-up of SPS is somewhat withdrawn with respect to Y12M for large m. We explain this by the smaller amount of computations that have to be done in SPS before communicating.

6. CONCLUSIONS

In this paper, we tested the direct sparse matrix solvers MA28, Y12M and one special purpose solver in the parallel ODE solver PSODE. If the number of nonzeros in the Jacobian of the derivative function is less than about 20–25% of the total number of entries, then it begins to pay off to use a sparse matrix solver. The costs can be further reduced by a factor varying from 1.75 to 2 by using a special purpose solver based on Gaussian Elimination and diagonal Markowitz pivoting without stability check. Up to a certain extent we can explain theoretically why numerically solving stiff ODEs with the use of a Newton-type process for the solution of the systems of nonlinear equations leads to linear systems that can be solved without stability pivoting.

Experiments on a Cray C98/4256 show speed-up factors of PSODE on four processors in the region 2.2–3.8, depending on the problem size, but not much on the (sparse) matrix solver.

APPENDIX

In this Appendix, we describe the test problems that we used in Section 5. All problems are contained in the test set for IVP solvers, which is available on the WWW page with URL

`http://www.cwi.nl/cwi/projects/IVPtestset.html`.

A more elaborate description, references to the literature and Fortran 77 codes of these (and other) problems can be found there.

A.1. Ring Modulator

The problem is of form (1), where $t_0 = 0$, $t_e = 10^{-3}$, $d = 15$ and f is defined by

$$f(t,y) = \begin{pmatrix} C^{-1}\big(y_8 - 0.5y_{10} + 0.5y_{11} + y_{14} - R^{-1}y_1\big) \\ C^{-1}\big(y_9 - 0.5y_{12} + 0.5y_{13} + y_{15} - R^{-1}y_2\big) \\ C_s^{-1}\big(y_{10} - q(U_{D1}) + q(U_{D4})\big) \\ C_s^{-1}\big(-y_{11} + q(U_{D2}) - q(U_{D3})\big) \\ C_s^{-1}\big(y_{12} + q(U_{D1}) - q(U_{D3})\big) \\ C_s^{-1}\big(-y_{13} - q(U_{D2}) + q(U_{D4})\big) \\ C_p^{-1}\big(-R_p^{-1}y_7 + q(U_{D1}) + q(U_{D2}) - q(U_{D3}) - q(U_{D4})\big) \\ -L_h^{-1}y_1 \\ -L_h^{-1}y_2 \\ L_{s2}^{-1}(0.5y_1 - y_3 - R_{g2}y_{10}) \\ L_{s3}^{-1}(-0.5y_1 + y_4 - R_{g3}y_{11}) \\ L_{s2}^{-1}(0.5y_2 - y_5 - R_{g2}y_{12}) \\ L_{s3}^{-1}(-0.5y_2 + y_6 - R_{g3}y_{13}) \\ L_{s1}^{-1}(-y_1 + U_{in1}(t) - (R_i + R_{g1})y_{14}) \\ L_{s1}^{-1}(-y_2 - (R_c + R_{g1})y_{15}) \end{pmatrix}. \tag{11}$$

The auxiliary functions $U_{D1}, U_{D2}, U_{D3}, U_{D4}, q, U_{in1}(t)$ and $U_{in2}(t)$ are given by

$$\begin{aligned} U_{D1} &= y_3 - y_5 - y_7 - U_{in2}(t), \\ U_{D2} &= -y_4 + y_6 - y_7 - U_{in2}(t), \end{aligned}$$

$$
\begin{aligned}
U_{D3} &= y_4 + y_5 + y_7 + U_{in2}(t),\\
U_{D4} &= -y_3 - y_6 + y_7 + U_{in2}(t),\\
q(U) &= \gamma(e^{\delta U} - 1),\\
U_{in1}(t) &= 0.5\sin(2000\pi t),\\
U_{in2}(t) &= 2\sin(20000\pi t).
\end{aligned}
$$

The values of the parameters are

$$
\begin{aligned}
C &= 1.6\cdot 10^{-8}, & L_{s3} &= 5\cdot 10^{-4},\\
C_s &= 10^{-9}, & R_{g1} &= 36.3,\\
C_p &= 10^{-8}, & R_{g2} &= 17.3,\\
R &= 25000, & R_{g3} &= 17.3,\\
R_p &= 50, & R_i &= 50,\\
L_h &= 4.45, & R_c &= 600,\\
L_{s1} &= 0.002, & \gamma &= 40.67286402\cdot 10^{-9},\\
L_{s2} &= 5\cdot 10^{-4}, & \delta &= 17.7493332.
\end{aligned}
$$

Initially, all components are zero; i.e., $y_0 = (0,\ldots,0)^\top$.

A.2. Pollution Problem

The problem is of form (1), where $t_0 = 0$, $t_e = 60$, $d = 20$ and f is defined by

$$
f = \begin{pmatrix}
-\sum_{j\in\{1,10,14,23,24\}} r_j + \sum_{j\in\{2,3,9,11,12,22,25\}} r_j\\
-r_2 - r_3 - r_9 - r_{12} + r_1 + r_{21}\\
-r_{15} + r_1 + r_{17} + r_{19} + r_{22}\\
-r_2 - r_{16} - r_{17} - r_{23} + r_{15}\\
-r_3 + 2r_4 + r_6 + r_7 + r_{13} + r_{20}\\
-r_6 - r_8 - r_{14} - r_{20} + r_3 + 2r_{18}\\
-r_4 - r_5 - r_6 + r_{13}\\
r_4 + r_5 + r_6 + r_7\\
-r_7 - r_8\\
-r_{12} + r_7 + r_9\\
-r_9 - r_{10} + r_8 + r_{11}\\
r_9\\
-r_{11} + r_{10}\\
-r_{13} + r_{12}\\
r_{14}\\
-r_{18} - r_{19} + r_{16}\\
-r_{20}\\
r_{20}\\
-r_{21} - r_{22} - r_{24} + r_{23} + r_{25}\\
-r_{25} + r_{24}
\end{pmatrix},
$$

where the r_i are auxiliary variables, given by

$$
\begin{aligned}
r_1 &= k_1\cdot y_1, & r_2 &= k_2\cdot y_2\cdot y_4,\\
r_3 &= k_3\cdot y_5\cdot y_2, & r_4 &= k_4\cdot y_7,\\
r_5 &= k_5\cdot y_7, & r_6 &= k_6\cdot y_7\cdot y_6,\\
r_7 &= k_7\cdot y_9, & r_8 &= k_8\cdot y_9\cdot y_6,
\end{aligned}
$$

$$
\begin{aligned}
r_9 &= k_9 \cdot y_{11} \cdot y_2, & r_{10} &= k_{10} \cdot y_{11} \cdot y_1,\\
r_{11} &= k_{11} \cdot y_{13}, & r_{12} &= k_{12} \cdot y_{10} \cdot y_2,\\
r_{13} &= k_{13} \cdot y_{14}, & r_{14} &= k_{14} \cdot y_1 \cdot y_6,\\
r_{15} &= k_{15} \cdot y_3, & r_{16} &= k_{16} \cdot y_4,\\
r_{17} &= k_{17} \cdot y_4, & r_{18} &= k_{18} \cdot y_{16},\\
r_{19} &= k_{19} \cdot y_{16}, & r_{20} &= k_{20} \cdot y_{17} \cdot y_6,\\
r_{21} &= k_{21} \cdot y_{19}, & r_{22} &= k_{22} \cdot y_{19},\\
r_{23} &= k_{23} \cdot y_1 \cdot y_4, & r_{24} &= k_{24} \cdot y_{19} \cdot y_1,\\
r_{25} &= k_{25} \cdot y_{20}.
\end{aligned}
$$

The values of the parameters k_j are

$$
\begin{aligned}
k_1 &= .350\text{E+}00, & k_2 &= .266\text{E+}02,\\
k_3 &= .123\text{E+}05, & k_4 &= .860\text{E}{-}03,\\
k_5 &= .820\text{E}{-}03, & k_6 &= .150\text{E+}05,\\
k_7 &= .130\text{E}{-}03, & k_8 &= .240\text{E+}05,\\
k_9 &= .165\text{E+}05, & k_{10} &= .900\text{E+}04,\\
k_{11} &= .220\text{E}{-}01, & k_{12} &= .120\text{E+}05,\\
k_{13} &= .188\text{E+}01, & k_{14} &= .163\text{E+}05,\\
k_{15} &= .480\text{E+}07, & k_{16} &= .350\text{E}{-}03,\\
k_{17} &= .175\text{E}{-}01, & k_{18} &= .100\text{E+}09,\\
k_{19} &= .444\text{E+}12, & k_{20} &= .124\text{E+}04,\\
k_{21} &= .210\text{E+}01, & k_{22} &= .578\text{E+}01,\\
k_{23} &= .474\text{E}{-}01, & k_{24} &= .178\text{E+}04,\\
k_{25} &= .312\text{E+}01.
\end{aligned}
$$

The initial vector y_0 is given by

$$y_0 = (0, 0.2, 0, 0.04, 0, 0, 0.1, 0.3, 0.01, 0, 0, 0, 0, 0, 0, 0, 0.007, 0, 0, 0)^{\mathsf{T}}.$$

A.3. EMEP Problem

The problem is of form (1), where $t_0 = 14400$, $t_e = 417600$, $d = 66$. Since the function f is too voluminous to be described here, we refer to the Web page mentioned previously for more details. The initial vector is given by

$$
y_{0,i} = \begin{cases}
1.0\text{E+}09 & \text{for } i = 1,\\
5.0\text{E+}09 & \text{for } i \in \{2, 3\},\\
3.8\text{E+}12 & \text{for } i = 4,\\
3.5\text{E+}13 & \text{for } i = 5,\\
1.0\text{E+}07 & \text{for } i \in \{6, 7, \ldots, 13\},\\
5.0\text{E+}11 & \text{for } i = 14,\\
1.0\text{E+}02 & \text{for } i \in \{15, 16, \ldots, 37\},\\
1.0\text{E}{-}03 & \text{for } i = 38,\\
1.0\text{E+}02 & \text{for } i \in \{39, 40, \ldots, 66\}.
\end{cases}
$$

REFERENCES

1. I.S. Duff, MA28—A set of Fortran subroutines for sparse unsymmetric linear equations, Technical Report AERE R8730, HMSO, London, (1977).
2. Z. Zlatev, J. Wasniewski and K. Schaumburg, Y12M, solution of large and sparse systems of linear algebraic equations, In *Lecture Notes in Computer Science*, No. 121, Springer-Verlag, Berlin, (1981).
3. B.P. Sommeijer, Parallelism in the numerical integration of initial value problems, Ph.D. Thesis, University of Amsterdam, (1992).

4. A. Bellen, Parallelism across the steps for difference and differential equations, In *Lecture Notes in Mathematics*, p. 1386, Springer-Verlag, (1987).
5. A. Bellen, R. Vermiglio and M. Zennaro, Parallel ODE-solvers with stepsize control, *JCAM* **31**, 277–293 (1990).
6. K. Burrage, Parallel methods for initial value problems, *Applied Numerical Mathematics* **11**, 5–26 (1993).
7. P. Chartier, Parallelism in the numerical solutions of initial value problems for ODEs and DAEs, Ph.D. Thesis, Université de Rennes I, France, (1993).
8. B. Orel, Parallel Runge-Kutta methods with real eigenvalues, *Applied Numerical Mathematics* **11**, 241–250 (1993).
9. P.J. van der Houwen and B.P. Sommeijer, Iterated Runge-Kutta methods on parallel computers, *SIAM J. Sci. Stat. Comput.* **12**, 1000–1028 (1991).
10. P.J. van der Houwen and B.P. Sommeijer, Analysis of parallel diagonally implicit iteration of Runge-Kutta methods, *APNUM* **11**, 169–188 (1993).
11. H.M. Markowitz, The elimination form of the inverse and its application to linear programming, *Management Sci.* **3**, 255–269 (1957).
12. I.S. Duff, A.M. Erisman and J.K. Reid, *Direct Methods for Sparse Matrices, Monographs on Numerical Analysis*, Oxford Science Publications, (1986).
13. J.J. Dongarra and E. Grosse, Distribution of software via electronic mail, Technical Report 30, *Commun. ACM*, 403–407 (1987).
14. M.Z. Jacobson and R.P. Turco, SMVGEAR: A sparse-matrix, vectorized gear code for atmospheric models, *Atmospheric Environment* **28**, 273–284 (1994).
15. A.C. Hindmarsh, ODEPACK, a systemized collection of ODE solvers, In *Scientific Computing*, (Edited by R. Stepleman *et al.*), pp. 55–64, IMACS, North-Holland, Amsterdam, (1983).
16. J.G. Verwer, J.G. Blom, M. van Loon and E.J. Spee, A comparison of stiff ODE solvers for atmospheric chemistry problems, *Atmospheric Environment* **29** (1995).
17. B.W. Char, K.O. Geddes, G.H. Gonnet, B.L. Leong, M.B. Monagan and S.M. Watt, *Maple V Language Reference Manual*, Springer-Verlag, New York, (1991).
18. Cray Research, Inc., *UNICOS Performance Utilities Reference Manual*, SR-2040 8.0 edition, (1994).
19. Cray Research, Inc., *CF77 Commands and Directives*, SR-3771 6.0 edition, (1994).