ELSEVIER

Applied Numerical Mathematics 28 (1998) 95–106

APPLIED NUMERICAL MATHEMATICS

ParalleloGAM: a parallel code for ODEs ☆

Pierluigi Amodio [a,*], Luigi Brugnano [b,1]

[a] *Dipartimento di Matematica, Via Orabona 4, I-70125 Bari, Italy*

[b] *Dipartimento di Matematica "U. Dini", Viale Morgagni 67/A, I-50134 Firenze, Italy*

Received 1 September 1997; received in revised form 17 February 1998; accepted 2 April 1998

Abstract

A new parallel solver for ODEs implementing a "parallelism across the steps" has been recently proposed in (Amodio and Brugnano, 1997; Brugnano and Trigiante, 1998), where it is shown that it is able to obtain an almost perfect speed-up on linear problems, and given mesh. A possible way to adapt this algorithm to efficiently handle nonlinear initial value problems has been studied in the companion paper (Brugnano and Trigiante, this issue). The corresponding algorithm is here analyzed in details, in order to show its parallel efficiency. Numerical tests on a distributed memory parallel computer are also included.

Keywords: Parallel methods for ODEs; Parallel computers; Parallelism across the steps; Boundary Value Methods

1. Introduction

Finding methods and strategies for solving the problem

$$y' = f(t, y), \quad t \in (t_0, T], \qquad y(t_0) = \eta \in \mathbb{R}^m, \tag{1}$$

on parallel computers has been the matter of several researches in the last thirty years (see [8] for a more complete overview). Such methods have been classified in different classes, according to the way they exploit the potentialities of a parallel machine. In particular, when the gain from parallelism consists in looking for simultaneous approximations to the solution at different grid points, one speaks about *parallelism across the steps*. A new method of this kind has been recently proposed [1,2] (see also [6]). In the above references the proposed algorithm is able to reach an almost perfect speed-up when problem (1) is linear and the mesh is fixed. This is achieved by constructing the whole discrete problem, consisting in a lower block bidiagonal linear system, whose solution can be efficiently obtained on a parallel computer.

☆ Work supported by CNR (contract no. 96.00243.CT01) and MURST.

* Corresponding author. E-mail: na.pamodio@na-net.ornl.gov.

[1] E-mail: na.brugnano@na-net.ornl.gov.

0168-9274/98/$19.00
PII: S0168-9274(98)00038-5

The extension of this approach to general nonlinear problems, requiring a variable mesh, needs additional efforts. In particular, the convergence of the iterative procedure used for solving the discrete problem must be handled. This matter has been recently studied in [7] where, essentially, a two-step procedure has been proposed. Suppose then that the vectors $\boldsymbol{y}$ and $\boldsymbol{h}$ contain the discrete solution and the stepsizes defining the optimal mesh to be used, respectively. Both vectors are in general unknown and are obtained by solving a discrete problem of the form

$$G(\boldsymbol{y}, \boldsymbol{h}) = \boldsymbol{0}, \tag{2}$$

where G depends on the chosen method and on the function f in (1). Problem (2), however, is not easily, nor cheaply solvable. Consequently, an efficient parallel procedure cannot in general rely on the direct solution of such equation. For this reason, in [7] it has been proposed to first solve a much simpler (and cheaper) problem,

$$G_1(\widetilde{\boldsymbol{y}}, \widetilde{\boldsymbol{h}}) = \boldsymbol{0}, \tag{3}$$

in order to derive a suitable mesh $\widetilde{\boldsymbol{h}}$ and an initial solution $\widetilde{\boldsymbol{y}}$. Then, one solves iteratively the problem

$$G_2(\boldsymbol{y}) = \boldsymbol{0}, \qquad G_2(\boldsymbol{y}) \equiv G(\boldsymbol{y}, \widetilde{\boldsymbol{h}}), \tag{4}$$

starting from the vector $\boldsymbol{y}^{(0)} = \widetilde{\boldsymbol{y}}$.

The solution of problem (3) is obtained by using an almost sequential procedure, called DLGSM [7]. On the other hand, problem (4) can be efficiently handled in parallel by using the modified Newton procedure, since the same approach described in [1,2] can be used at each iteration of such method.

The structure of the paper is the following: Sections 2 and 3 are devoted to study the complexity of the two steps of the above procedure. Since this paper is strictly related to the companion paper [7], we use the same notation defined there. In Section 4 a model for the expected speed-up of the whole algorithm is derived. Finally, Section 5 contains some numerical tests carried out on a distributed memory parallel computer, along with some concluding remarks.

2. Analysis of the DLGSM step

In this section, we analyze the implementation of the first step of the procedure described above. The goal of this starting procedure is twofold:

(1) determine a suitable *coarse* mesh,

$$t_0 \equiv \tau_0 < \tau_1 < \cdots < \tau_p \leqslant T, \tag{5}$$

and a corresponding *fine* mesh,

$$t_{ni} = \tau_{i-1} + nh_i, \quad n = 1, \ldots, s, \qquad h_i = (\tau_i - \tau_{i-1})/s, \quad i = 1, \ldots, p;$$

(2) determine an approximate solution

$$y_{ni} \approx y(t_{ni}), \quad n = 1, \ldots, s, \ i = 1, \ldots, p,$$

such that the vectors

$$\widetilde{\boldsymbol{h}} = (h_1, \ h_2, \ \ldots, \ h_p)^{\mathrm{T}} \otimes E_s, \quad E_s = (1, \ldots, 1)^{\mathrm{T}} \in \mathbb{R}^s, \tag{6}$$

and

$$\tilde{\mathbf{y}} = (\eta, y_{11}, \ldots, y_{s1}, \ldots, y_{1p}, \ldots, y_{sp})^{\mathrm{T}} \tag{7}$$

satisfy Eq. (3). This is achieved by performing, over each subinterval $[\tau_{i-1}, \tau_i]$, $i = 1, \ldots, p$, s steps of the trapezoidal rule with stepsize h_i. Such operations are performed by the ith processor, assuming that p processors are available. The choice of the trapezoidal rule is due both to its relatively low computational cost, and to its stability properties, which are similar to those of GAMs [5,6], the methods used in the second step.

It is remarkable to observe that the *window* $[\tau_0, \tau_p]$ in (5) is dynamically determined in order to guarantee the convergence of the modified Newton iteration, used for solving (4), within a given number of steps [7]. This implies that in general more than one window may be required to cover the interval $[t_0, T]$ of problem (1). However, the control of the convergence of the nonlinear iteration for solving (4) requires an additional computational effort, with respect to that needed to solve (3). The latter equation is solved by using an iterative procedure, called *Diagonally Linearized Gauss–Seidel Method* or, shortly, DLGSM. On the ith subinterval it is defined by the following iteration where, hereafter, I_r is the identity matrix of size r,

$$\left(I_m - \frac{h_i}{2} J_{0i}\right) y_{ni}^{(j)} = y_{n-1,i}^{(j)} + \frac{h_i}{2} f_{n-1,i}^{(j)} + \frac{h_i}{2}\left(f_{ni}^{(j-1)} - J_{0i} y_{ni}^{(j-1)}\right),$$
$$n = 1, \ldots, s, \quad j = 1, 2, 3. \tag{8}$$

In (8) $J_{0i} = J(\tau_{i-1}, y_{0i})$, where $J(t, y)$ denotes the Jacobian of the function f evaluated at (t, y), j is the index of the DLGSM iteration, and $f_{ni}^{(j)} = f(t_{ni}, y_{ni}^{(j)})$. Finally, the starting vectors are given by $y_{ni}^{(0)} = y_{s,i-1}$, $n = 0, \ldots, s$, where for $i = 1$, $y_{s0} = \eta$ is the initial condition of problem (1), and, in general, $y_{s,i-1} \equiv y_{0i} \approx y(\tau_{i-1})$ is known from the previous subinterval.

In [7] a complete convergence analysis for such iteration is done, resulting in a mesh selection strategy which determines the new stepsize h_{i+1} in order to have the error at the third DLGSM iteration adequately small. Such stepsize selection, done by controlling the convergence of DLGSM, is also able to control the truncation error (or the local error), with the exception of some pathological cases. The latter are avoided by switching to the control of the truncation error when either one of the following two conditions holds true (the norm used is the infinity norm):

$$\frac{\|f_{1i} - f_{0i}\|}{h} > 1.1\,\|J_{0i} f_{0i}\|, \qquad \frac{\|f_{0i}''\|}{(\|f_{0i}'' - J_{0i}^2 f_{0i}\|/\|f_{0i}\|)^{3/2}} > \nu_1, \tag{9}$$

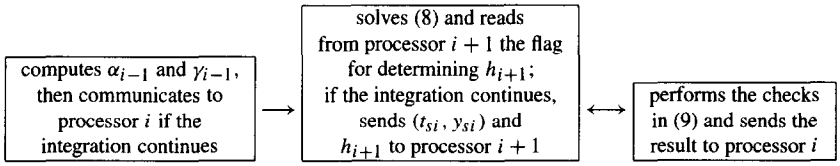
where f_{0i}'' is an approximation to the second derivative of f at (t_{0i}, y_{0i}), and ν_1 is a fixed tolerance. In such a case, the new stepsize is determined by taking into account suitable error estimates (see [7, Section 3.3]).

The arithmetic complexity of the step is summarized in the following list, where we count as 1 *flop* one of the four basic binary *fl*oating *op*erations:

(i) 1 Jacobian evaluation;
(ii) s function evaluation per iteration;
(iii) $\frac{2}{3}m^3 + \mathrm{O}(m^2)$ flops for the factorization of the matrix $I_m - 0.5h_i J_{0i}$;
(iv) $4sm^2 + \mathrm{O}(sm)$ flops per iteration for both evaluating the right-hand side and solving the linear system (8).

Consequently, if three iterations are performed, we obtain a total of $\approx \frac{2}{3}m^3 + 12sm^2$ flops, 1 Jacobian and $3s$ function evaluations. After that, the checks (9) are performed, requiring approximately $4m^2$ flops, and, when needed, the computation of the truncation error (or the local error), that requires an additional cost of $O(sm)$ flops (respectively, $O(sm^2)$ flops). However, on a parallel computer the quantities needed for checking (9) can be computed by the $(i+1)$st processor, while processor i performs the computations in (i)–(iv).

After the above operations, by considering that the ith subinterval is handled by the ith processor, the final point (t_{si}, y_{si}) and the new stepsize h_{i+1} are sent to processor $i+1$, which starts the same procedure for the next subinterval. In the meantime, the ith processor carries out the procedure for the dynamic determination of the window of integration, where the convergence of the modified Newton method for solving (4) should be assured. This is done by computing approximations to the two parameters α_i and γ_i (see [7, Section 4.1] for further details), used to control the convergence of the nonlinear iteration. It requires an additional Jacobian evaluation (at the final point t_{si}) plus the solution of $2s$ linear systems with the same coefficient matrix factored in step (iii) ($\approx 4sm^2$ flops) and the transmission of two auxiliary vectors. However, if implemented on a sequential computer, the new Jacobian evaluation does not introduce additional overhead, since it should be evaluated anyway to carry out the iteration over the next subinterval. Similarly, the whole additional cost is not effective in the implementation on a parallel computer, since all the operations are executed when the subsequent processor performs the operations (i)–(iv) previously listed (obviously, corresponding to the next subinterval). The following diagram summarizes the operations which can be executed concurrently on processors $i-1, i, i+1$, $i = 1, \ldots, p$, where, for simplicity, processors 0 and p, as well as processors 1 and $p+1$, coincide.



The DLGSM procedure is stopped as soon as the product $\alpha_i\gamma_i$ exceeds a given upper bound, chosen in order to have the modified Newton iteration converging in a given number of steps. In such a case, the window is chosen as the interval $[\tau_0, \tau_{i+1}]$.

We conclude summarizing that the first step of the procedure has a sequential complexity of 1 Jacobian evaluation, $3s$ function evaluations and $\frac{2}{3}m^3 + 16sm^2 + O(sm) + O(m^2)$ flops, whereas the parallel complexity (three processors are active at each time) amounts to 1 Jacobian evaluation, $3s$ function evaluations and $\frac{2}{3}m^3 + 12sm^2 + O(sm) + O(m^2)$ flops.

For what concerns communications, processor i reads one scalar flag from processor $i+1$ (i.e., the flag for the stepsize control, containing the sum of the Boolean values of (9)). From processor $i-1$, it reads the starting point $(t_{s,i-1}, y_{s,i-1})$, the stepsize h_i, the two scalars α_{i-1} and γ_{i-1} plus the two auxiliary vectors that are needed to compute α_i and γ_i. After that, it sends to processor $i+1$ the final point (t_{si}, y_{si}), the new stepsize h_{i+1}, and the two parameters α_i, γ_i, along with two auxiliary vectors. Consequently, assuming a ring topology connecting the p processors, we have a total complexity of $(3m+4)p$ data transmitted among adjacent processors (which reduce to $(m+4)p$ if we consider that the transmission of $2mp$ data overlaps the remaining workload).

3. Analysis of the second step

The second step of the procedure, as sketched in the introduction, consists in applying the modified Newton procedure for solving Eq. (4), starting from the vector $y^{(0)} = \widetilde{y}$ obtained by the DLGSM procedure. This amounts to solve the linear systems of equations (see [7, Section 3]),

$$Mz^{(j)} = g^{(j)}, \quad j = 0, 1, \ldots, \tag{10}$$

then updating

$$y^{(j+1)} = y^{(j)} - z^{(j)}, \tag{11}$$

until a suitable stopping criterion is satisfied. The matrix M turns out to be lower block bidiagonal,

$$M = \begin{pmatrix} I_m & & & & \\ v_1 & M_1 & & & \\ & V_2 & M_2 & & \\ & & \ddots & \ddots & \\ & & & V_p & M_p \end{pmatrix}, \tag{12}$$

where, for any allowed index i, the matrices M_i and V_i are $sm \times sm$,

$$M_i = A_s \otimes I_m - h_i(B_s \otimes I_m) \begin{pmatrix} J_{1i} & & \\ & \ddots & \\ & & J_{si} \end{pmatrix},$$

$$V_i = [O \mid v_i], \quad v_i = a_s \otimes I_m - h_i b_s \otimes J_{s,i-1},$$

$[a_s \ A_s]$ and $[b_s \ B_s]$ contain the coefficients of the used k-step GAM (having order $k+1$), and $J_{ni} = J(t_{ni}, y_{ni})$. The rightmost lower index of each quantity usually denotes the subinterval in the coarse mesh. When using a distributed memory parallel computer, it also identifies the processor where the given quantity has to be stored. Indeed, processor i deals with the computational load corresponding to the subinterval $[\tau_{i-1}, \tau_i]$ in the coarse mesh (5), when p processors are available.

In order to efficiently solve the linear system (10) on a parallel computer with p processors, according to [1,2] we consider the factorization $M = DT$,

$$D = \begin{pmatrix} I_m & & & \\ & M_1 & & \\ & & \ddots & \\ & & & M_p \end{pmatrix}, \qquad T = \begin{pmatrix} I_m & & & & \\ w_1 & I_{sm} & & & \\ & W_2 & I_{sm} & & \\ & & \ddots & \ddots & \\ & & & W_p & I_{sm} \end{pmatrix},$$

where the blocks W_i are $sm \times sm$, $W_i = [O \mid w_i]$, and w_i is the solution of

$$M_i w_i = v_i, \quad i = 1, \ldots, p. \tag{13}$$

The above factorization is indeed able to produce a very efficient parallel algorithm on p processors. In fact, the two factors can be easily computed in parallel. Moreover, since the matrix D is block diagonal,

the p subsystems corresponding to its diagonal blocks can be solved independently. Finally, the matrix T can be permuted to get

$$\left(\begin{array}{ccc|ccc} I_{(s-1)m} & & & \widehat{\boldsymbol{w}}_1 & O_m & \\ & \ddots & & & \ddots & \ddots \\ & & I_{(s-1)m} & & & \widehat{\boldsymbol{w}}_p \quad O_m \\ \hline & & & I_m & & \\ & & & w_{s1} & I_m & \\ & O & & & \ddots & \ddots \\ & & & & & w_{sp} \quad I_m \end{array}\right),$$

where O_m is the $m \times m$ zero matrix, and (see (13))

$$\boldsymbol{w}_i \equiv \begin{pmatrix} \widehat{\boldsymbol{w}}_i \\ w_{si} \end{pmatrix}, \quad w_{si} \in \mathbb{R}^{m\times m}.$$

Consequently, we still have perfect parallelism on p processors, once the *reduced system* with the *reduced matrix*

$$\begin{pmatrix} I_m & & & \\ w_{s1} & I_m & & \\ & \ddots & \ddots & \\ & & w_{sp} & I_m \end{pmatrix}$$

has been solved. We observe that such matrix has dimension $(p+1)m \times (p+1)m$, and then the (sequential) solution of the reduced system only requires $2pm^2 + pm$ flops plus p data transmissions of length m, which are the only transmissions needed in the second step of the procedure. It is worth noting that, in practice, the whole computational load for solving the reduced system is negligible. This also because it is hidden by the asynchronous workload of the processors, due to the sequential nature of DLGSM.

For what concerns the check of the exit condition for the Newton iteration, since the matrix M in (10) is block bidiagonal (see (12)), on processor i we only consider the norm of the first i blocks, say $z_1^{(j)}, \ldots, z_i^{(j)}$, of the vector $z^{(j)}$. This means that, in general, processor i may require a number of Newton iterations smaller than that required by processors $i+1, \ldots, p$. Consequently, the size of the bidiagonal system depends on the the number of processors for which the Newton iteration is not yet completed.

The following list summarizes the sequential computational cost of this section where k is the number of steps of the GAM chosen and, for brevity, we neglect the smaller terms:

(1) $p(s-1)$ additional Jacobian evaluations for constructing the matrix M;
(2) $p(s-k/3)k^2m^3$ flops for the factorization of the blocks M_i;
(3) $3pskm^3$ flops for constructing the block vectors $\boldsymbol{w}_i$ (see (13)); they are not required if the code is run sequentially;
(4) ps function evaluations plus $3pksm^2$ flops for each iteration (10)–(11);
(5) $\approx 2pm^2$ flops plus p transmissions of length m for the solution of the reduced system in each Newton iteration (sequential section).

After the new, more accurate, discrete solution has been obtained, an approximation to the global error over the whole window is also computed. This requires a computational cost equal to that needed for one iteration (10)–(11). In fact, the vector $\boldsymbol{e}$ whose entries are the (estimate) global errors at the grid points,

is obtainable through deferred correction by solving a linear system with the same coefficient matrix in (12), whose right-hand side is obtained by using the block GAM of order $k+2$ [3,6].

4. Expected speed-up

We are now in the position of deriving a simplified model for the expected speed-up of the whole procedure (first and second step) when executed on p ($p > 1$) processors. In the expression, we use the following ratios:

$$\widehat{J} = \frac{\text{time for 1 Jacobian evaluation}}{\text{time to execute 1 flop}}, \qquad \widehat{f} = \frac{\text{time for 1 function evaluation}}{\text{time to execute 1 flop}},$$

and

$$\widehat{d} = \frac{\text{time to send/receive 1 datum}}{\text{time to execute 1 flop}}.$$

We neglect the smaller terms, in order to avoid a huge expression. Moreover, we assume to get convergence for the modified Newton iteration (10)–(11) in at most three iterations. Consequently, by considering all the arguments in the previous sections, we obtain that the speed-up S_p is given by

$$S_p(k,s,m) = \frac{(ps+1)\widehat{J} + 7ps\widehat{f} + (12k+16)spm^2 + \left((s-\frac{k}{3})k^2 + \frac{2}{3}\right)pm^3}{(p+s)\widehat{J} + (3p+4)s\widehat{f} + 5pm\widehat{d} + (12ks+12ps+8p)m^2 + \left((s-\frac{k}{3})k^2 + 3sk + \frac{2}{3}p\right)m^3}, \tag{14}$$

where we recall that:

- p is the number of subintervals in the window, which equals the number of the processors used;
- k is the number of steps of the GAM used;
- s is the number of points in each subinterval;
- m is the size of the ODE.

In order to make some prediction, we can consider the case where each entry of the Jacobian matrix, as well as each entry of the function f, requires O(1) flops, so that $\widehat{J} = \mathrm{O}(m^2)$ and $\widehat{f} = \mathrm{O}(m)$. Consequently, for m sufficiently large, we have that

$$S_p(k,s,m) \approx \frac{p}{1 + \frac{3sk+2(p-1)/3}{(s-k/3)k^2+2/3}}. \tag{15}$$

Considering that $s > k$, we obtain that the right-hand side in (15), for $p < sk$, is greater than $p(1+5.5/k)^{-1}$, which may become significantly close to p as k grows. Anyway, this is welcome, since the higher k the higher the order of the method (which is equal to $k+1$). Moreover, also the assumption $p < sk$ is reasonable since, for example, for $k = 8$ and $s = 10$ we obtain $p < 80$. In addition to this, the right-hand side (15) is able to provide us with the following information about the speed-up. Namely,

(1) it increases with k;

(2) it increases with s, provided that $k \geqslant 2$ (which is trivially satisfied);

(3) it increases with p, even though the parallel efficiency, i.e., $E_p = S_p/p$, decreases monotonically.

Concerning the last point, it is an easy matter to verify that the values of S_p accumulate towards $1 + 1.5(s - k/3)k^2$, as $p \to \infty$.

4.1. Expected speed-up for fixed number of processors

The previous analysis applies to the case where:
(a) one window is required for solving problem (1);
(b) the number of parallel processors equals the number of subintervals in the window.
The above assumptions, however, may not hold in general since, as we have said, the width of the window (and then the number of subintervals needed to cover it) is dynamically determined in the first step of the procedure, whereas the number of processors is generally fixed. Moreover, the problem may require more than one window, and the number of subintervals inside each window may differ. As a consequence, it is appropriate to slightly modify the previous analysis to handle the more general situation. Nevertheless, in order to keep the analysis as simple as possible, we shall consider the simpler case of one window, containing ℓ subintervals, when p processors are available, even though the arguments can be readily generalized to the case of multiple windows.

By slightly modifying the analysis in the previous sections, we then obtain that the sequential execution time is given by

$$T_1 = (1 + s\ell)\hat{J} + 7s\ell\hat{f} + (12k + 16)s\ell m^2 + ((s - k/3)k^2 + \tfrac{2}{3})\ell m^3. \tag{16}$$

Similarly, the parallel execution time on p processors turns out to be

$$T_p = \left(\ell + \left\lceil\frac{\ell}{p}\right\rceil s\right)\hat{J} + \left(3\ell + 4\left\lceil\frac{\ell}{p}\right\rceil\right)s\hat{f} + 5\ell m\hat{d} + \left(12\left\lceil\frac{\ell}{p}\right\rceil ks + 12\ell s + 8\ell\right)m^2 + \left(\left(sk^2 - \frac{k^3}{3} + 3sk\right)\left\lceil\frac{\ell}{p}\right\rceil + \frac{2}{3}\ell\right)m^3. \tag{17}$$

Evidently, when $\ell = p$, (16) and (17) reduce to the numerator and the denominator of (14), respectively.

Proceeding as done in the previous section, when $m \gg 1$ the speed-up on p processors (i.e., the ratio T_1/T_p) is approximately given by

$$S_p \approx \frac{((s - k/3)k^2 + 2/3)\ell}{(sk^2 - k^3/3 + 3sk)\lceil\ell/p\rceil + (2/3)\ell}. \tag{18}$$

The right-hand side in (18) provides the same value as the right-hand side in (15) if the ratio ℓ/p is an integer, or very close to it if $\ell \gg p$. Conversely, it may be much smaller.

We also mention that the conclusions at the end of the previous section, derived from the right-hand side in (15), can be extended to the right-hand side in (18), provided that the term "increases" at the third point is replaced by "does not decrease".

5. Numerical tests

We now report numerical results obtained on a few test problems taken from the literature. They have been obtained by using a first release of the code, written in Fortran with the Express parallel library [9]. The parallel platform used is an *N-cube* with 128 nodes connected with a hypercube topology. The numerical tests, however, have been carried out on subcubes of maximum dimension 5 (namely, at most 32 parallel processors have been used). Because of stability reasons [6], we use block GAMs up to order 9 ($k = 8$). For all methods, the blocksize used is $s = 10$. We here report results concerning the

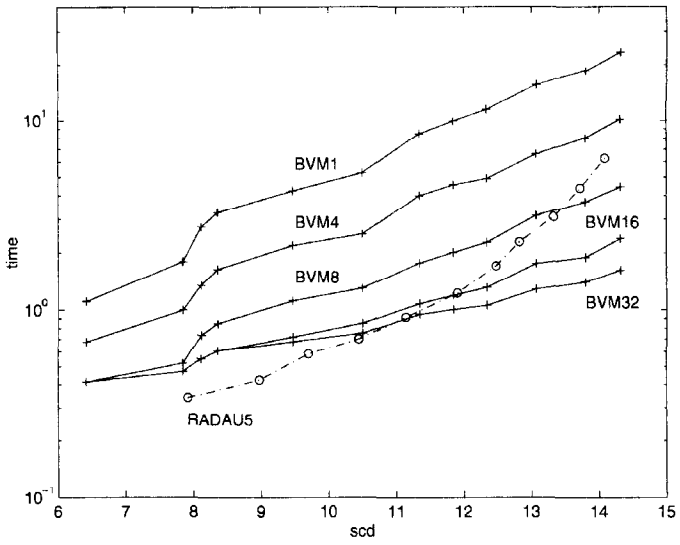


Fig. 1. Work-precision diagram for the Chemical Akzo Nobel problem.

parallel execution of the algorithm and comparing it with the code RADAU5 [11]. We plan to port the final version of the code to PVM [10] in order to improve its portability.

We want to stress that the results are obtained by using a very preliminary version of the code, and that its tuning is far from to be complete. In particular, the choice of the most appropriate tolerance parameters deserves further investigation, as well as the study of cheaper implementations for the second step of the procedure. Another open problem is, for a given tolerance, the optimal choice of the order of the method to be used in the second step. For this reason, for each tolerance we plot the first one or two best obtained results, among the considered methods, in order to get the work-precision diagrams. For all tests, the tolerance used for DLGSM (which equals the initial stepsize) is 10^3 times the tolerance, say tol, used for the modified Newton iteration (10)–(11) in the second step of the procedure.

The first test problem is the Chemical Akzo Nobel problem from the CWI testset [12], an ODE of dimension 6. We have run the parallel code with tolerance $\text{tol} = 10^{-i}$, $i = 5, \ldots, 10$, and orders $5, \ldots, 9$ (i.e., with $k = 4, \ldots, 8$), on 1, 4, 8, 16, 32 processors. In Fig. 1 we plot the work-precision diagram relative to the parallel code executed on p processors (BVMp, hereafter) and RADAU5 (the execution times are in seconds). As usual, on the horizontal axis we have the number of significant computed digits (scd) in the obtained solution. The data for RADAU5 have been obtained by using the parameters $\text{atol} = \text{rtol} = h0 = 10^{-i}$, $i = 7, \ldots, 17$. For the parallel code (BVM), since the problem has required only one window, the code provides the estimate of the global error, which well matches the actual error at the last point, where the solution is known. We observe that the poor parallel performances for large tolerances are due to the fact that too few mesh points are required. As a consequence, the size of the corresponding discrete problem is too small. For the parallel code, the measured speed-ups (with respect to its sequential execution) range from 2.7 to 14.4.

The second test problem is the van der Pol problem [7] with parameter $\mu = 10^3$. The tolerances used for the parallel code are $\text{tol} = 10^{-i}$, $i = 7, \ldots, 13$, with orders $4, \ldots, 9$, while those used for RADAU5 are $\text{atol} = \text{rtol} = h0 = 10^{-i}$, $i = 5, \ldots, 17$. In Fig. 2 there is the work-precision diagram for this problem. For clarity, we have dropped the plot for BVM32, which essentially overlaps that of BVM16 (i.e., for

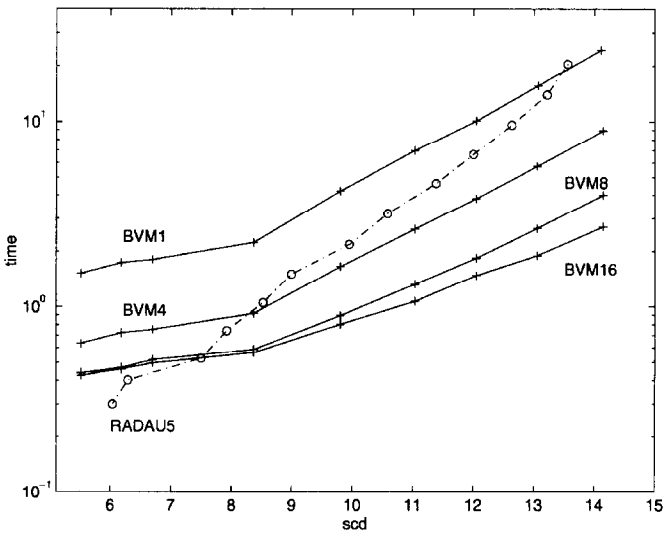


Fig. 2. Work-precision diagram for the van der Pol problem.

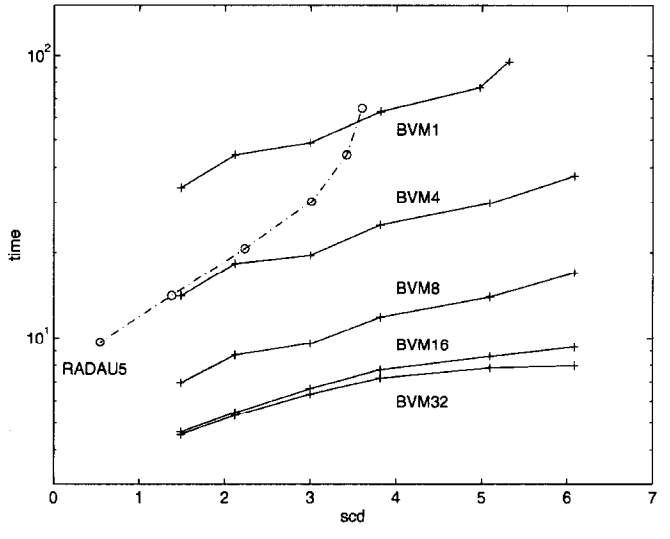


Fig. 3. Work-precision diagram for the Kepler problem.

this problem, there is no practical speed-up, when passing from 16 to 32 processors). Considerations similar to those made for the previous test problem also hold true now, when large tolerances are used. The obtained speed-ups for the parallel code range from 3.6 to 9.0.

Finally, we consider the Kepler problem [13, pp. 6–8], with eccentricity $e = 0.999$. With this value of e, the problem is very nasty. We evaluate the accuracy at $T = 4\pi$, namely after two periods. The tolerances used for the parallel code are $\text{tol} = 10^{-i}$, $i = 12, \ldots, 15$, with the methods of order 7, 8, 9, while those used for RADAU5 are $\text{atol} = \text{rtol} = h0 = 10^{-i}$, $i = 11, \ldots, 16$. In Fig. 3 there is the plot of the work-precision diagram for this problem. The obtained speed-ups range from 7.5 to 11.9.

5.1. Conclusions and final remarks

From the results in the numerical tests, one may infer that the present version of the parallel code is quite well scalable. As matter of fact, we get speed-ups up to ≈ 15, when the problem allows the use of large windows. Moreover, the code compares favorably with existing sequential codes, at least when a high accuracy is required, and allows to obtain estimates of the global error. At the moment, however, for larger tolerances RADAU5 performs generally better. This because, in such a case, too few mesh points are required, thus resulting in a small sized discrete problem. On the other hand, the good performances for high accuracies are due to the availability of high order stable methods.

Another drawback for the parallel code consists in its memory requirements. In fact, even though the storage is distributed over all the parallel processors used, each processor deals at least with one subinterval in the coarse mesh. That is, a storage requirements of at least $s(k+2)m^2$ data is required, which is cumbersome for large problems. Moreover, to construct and factor M_i one also needs s Jacobian evaluation plus $\approx (s - k/3)k^2m^3$ flops, which is a relatively high cost. In order to overcome both problems, we are currently studying a new implementation of BVMs [4], which should require only one Jacobian evaluation, plus the storage and factorization of one matrix of size m, for each subinterval in the coarse mesh. This would dramatically reduce the computational cost of the parallel solver, both for storage requirement and operations count. We plan to upgrade the current version of the code, after some additional investigation will be carried out.

Acknowledgements

The authors wish to thank Professor Donato Trigiante for the helpful discussions and the reviewer for his comments.

References

[1] P. Amodio and L. Brugnano, Parallel implementation of block boundary value methods for ODEs, *J. Comput. Appl. Math.* 78 (1997) 197–211.

[2] P. Amodio and L. Brugnano, Parallel ODE solvers based on block BVMs, *Adv. Comput. Math.* 7 (1–2) (1997) 5–26.

[3] L. Brugnano, Boundary value methods for the numerical approximation of ordinary differential equations, in: *Lecture Notes in Computer Science* 1196 (1997) 78–89.

[4] L. Brugnano, Blended Block BVMs (B_3VMs): a family of economical implicit methods for ODEs (in progress).

[5] L. Brugnano and D. Trigiante, Boundary value methods: the third way between linear multistep and Runge–Kutta methods, *Comput. Math. Appl.* (to appear).

[6] L. Brugnano and D. Trigiante, *Solving Differential Problems by Multistep Initial and Boundary Value Methods* (Gordon and Breach, Amsterdam, 1998).

[7] L. Brugnano and D. Trigiante, Parallel implementation of block boundary value methods on nonlinear problems: theoretical results, *Appl. Numer. Math.* 28 (1998) 127–141.

[8] K. Burrage, *Parallel and Sequential Methods for Ordinary Differential Equations* (Clarendon Press, Oxford, 1995).

[9] *Express: a Communication Environment for Parallel Computers* (ParaSoft Corp., 1988).

[10] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek and V. Sunderam, PVM 3 User's Guide and Reference Manual, Report ORNL/TM-12187 (May 1993).
[11] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II*, Springer Series in Computational Mathematics 14 (Springer, Berlin, 1991).
[12] W.M. Lioen, J.J.B. de Swart and W.A. van der Veen, Test set for IVP solvers, CWI Report NM-R9615 (August 1996).
[13] J.M. Sanz-Serna and M.P. Calvo, *Numerical Hamiltonian Problems* (Chapman & Hall, London, 1994).