ELSEVIER

Journal of Computational and Applied Mathematics 78 (1997) 197–211

JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS

# Parallel implementation of block boundary value methods for ODEs[1]

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

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

[b]*Dipartimento di Energetica, Via C. Lombroso 6/17, 50134 Firenze, Italy*

Received 22 March 1996; revised 28 August 1996

**Abstract**

The parallel solution of initial value problems for ODEs has been the subject of much research in the last thirty years, and different approaches to the problem have been devised. In this paper we examine the parallel methods derived by block *boundary value methods* (*BVMs*), recently introduced for approximating Hamiltonian problems. Here we restrict the analysis of the methods when applied to linear problems, since their nonlinear parallel implementation deserves further study. However, for linear problems, the methods can reach a high parallel efficiency.

Some of these solvers can also be adapted for approximating continuous two-point boundary value problems. Numerical tests carried out on a distributed memory parallel computer are reported.

*Keywords*: Numerical methods for ODEs; Boundary value methods; Parallel computers

*AMS classification*: 65L05; 65L10; 65L06; 65Y05

## 1. Introduction

The solution of initial value problems (IVPs) for ODEs,

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

is often approximated by using appropriate discrete numerical schemes. In particular, we are interested in methods which are suitable for an efficient implementation on parallel computers. This has been subject of investigation in the last thirty years [17, 18], and different approaches to such problems have been considered (see, for example, [6, 13, 19, 21]). The most straightforward is that of exploiting possible parallelism existing in the continuous problem (*parallelism across the problem*). However, this approach is usually considered when explicit methods are used. As a consequence, a

* Corresponding author. e-mail: na.brugnano@na-net.ornl.gov.

[1] Work supported by CNR, contract n. 96.00243.CT01.


*PII* S0377-0427(96)00112-4

more general way of devising parallel ODE solvers was that of considering methods whose work per step can be split over a certain number of processors. The so-called solvers with *parallelism across the method* are then obtained. Such methods are essentially Runge–Kutta schemes and, in general, they have not a high degree of parallelism. For this reason, another possible approach consists in the simultaneous approximation of the continuous solution at several grid points. The obtained solvers are then characterized by a *parallelism across the steps.*

In the last few years, this scenario has been enriched because of the introduction of a new class of methods, namely the *boundary value methods* (*BVMs*), which presents a natural parallelism in time [3, 8, 16]. As a consequence, their parallel implementation leads to the definition of parallel ODE solvers falling in the intersection of the last two classes of methods.

Recently, a block version of such methods has been introduced [11]. It has made possible the definition of a very efficient parallel ODE solver, which is the subject of the present paper. Here, however, we restrict our analysis to the simpler case where problem (1) is linear. This because the nonlinear implementation of the methods is still subject of research.

In Section 2 the main facts about BVMs are recalled, along with their block implementation. In Section 3 we introduce the parallel solver. A sketch of the generalization of the parallel algorithm for solving continuous two-point boundary value problems (BVPs) is given in Section 4. In Section 5 we analyze the complexity of the algorithm, in order to derive a model for the expected speedup. Finally, in Section 6 some numerical examples, carried out on a distributed memory parallel computer, are reported.

## 2. Boundary value methods

Suppose that the integration interval of problem (1) is discretized by using a uniform mesh with stepsize $h = (T - t_0)/s$. The simplest way to define BVMs is to consider the application of a $k$-step linear multistep formula (LMF)

$$\sum_{i=0}^{k} \alpha_i y_{n+i} = h \sum_{i=0}^{k} \beta_i f_{n+i} \tag{2}$$

over this mesh, relaxing the usual request of assigning all the $k$ conditions needed by the discrete problem at the initial points $t_0, t_1, \ldots, t_{k-1}$. The latter choice originates all the known methods based on LMF, which approximate the given continuous IVP by means of a discrete IVP.

Let now $k_1$ and $k_2$ be two natural numbers, with $k_1 + k_2 = k$. Then, in principle one could fix the first $k_1$ values of the discrete solution,

$$y_0, \; y_1, \ldots, y_{k_1-1}, \tag{3}$$

and the final $k_2$ ones,

$$y_{s-k_2+1}, \ldots, y_s. \tag{4}$$

In this way, the continuous IVP is approximated by means of a discrete BVP with $(k_1, k_2)$-*boundary conditions*. The obtained methods have been called boundary value methods or, shortly, BVMs. Even if earlier references on this approach exists [5, 14], these methods have been systematically

examined only in the last years, starting from [15]. In particular, the usual notions of 0-stability and $A$-stability have been generalized, leading to the definitions of $0_{k_1k_2}$-stability and $A_{k_1k_2}$-stability [10]. It has also been shown that no more order barrier exists for such methods. In fact, several families of BVMs have been found, each containing $0_{k_1k_2}$-stable, $A_{k_1k_2}$-stable methods of arbitrary high order [2, 10], including methods of order $p = 2k$, which is the highest possible order for a $k$-step LMF [1].

The problem of finding the values in (3) and (4) not provided by the continuous problem can be easily overcome. In fact, let us rewrite formula (2), used with $(k_1, k_2)$-boundary conditions, as follows:

$$\sum_{i=-k_1}^{k_2} \alpha_{i+k_1} y_{n+i} = h \sum_{i=-k_1}^{k_2} \beta_{i+k_1} f_{n+i}, \quad n = k_1, \ldots, s - k_2. \tag{5}$$

We have then a set of $s - k + 1$ equations which use the values

$$y_0, y_1, \ldots, y_s \tag{6}$$

of the discrete solution. Of such values, only $y_0$ is provided by the continuous problem. The remaining $k - 1$ additional values in (3) and (4) can also be considered as unknowns, by introducing an equal number of equations independent of those in (5). This is usually done by means of a set of $k_1 - 1$ *additional initial equations*,

$$\sum_{i=0}^{k} \alpha_i^{(j)} y_i = h \sum_{i=0}^{k} \beta_i^{(j)} f_i, \quad j = 1, \ldots, k_1 - 1, \tag{7}$$

and $k_2$ *final* ones,

$$\sum_{i=0}^{k} \alpha_{k-i}^{(j)} y_{s-i} = h \sum_{i=0}^{k} \beta_{k-i}^{(j)} f_{s-i}, \quad j = s - k_2 + 1, \ldots, s. \tag{8}$$

The above equations are conveniently obtained by a set of *additional methods* having the same order of the main formula (5). For simplicity, we have assumed such formulae to have the same number of steps as the main one.

One then obtains a set of $s$ equations in the $s + 1$ unknowns (6). As a consequence, the discrete solution is obtained by providing only one more condition, which is that given by the continuous problem. In this way, by choosing appropriate BVMs, it is also possible to approximate continuous BVPs [9].

The previous use of BVMs has another important implication. In fact, let us recast the discrete problem (5)–(8) in matrix form. Supposing, for simplicity, that the problem (1) is scalar, one then obtains

$$\begin{pmatrix} 1 & \mathbf{0}_s^{\mathrm{T}} \\ \boldsymbol{a} & A \end{pmatrix} \begin{pmatrix} y_0 \\ \boldsymbol{y} \end{pmatrix} - h \begin{pmatrix} 0 & \mathbf{0}_s^{\mathrm{T}} \\ \boldsymbol{b} & B \end{pmatrix} \begin{pmatrix} f_0 \\ \boldsymbol{f} \end{pmatrix} = \begin{pmatrix} \eta \\ \mathbf{0}_s \end{pmatrix}, \tag{9}$$

where, for any integer $r$, we denote with $\mathbf{0}_r$ the null vector in $\mathbb{R}^r$,

$$\boldsymbol{y} = (\, y_1, \ldots, y_s \,)^{\mathrm{T}}, \qquad \boldsymbol{f} = (\, f_1, \ldots, f_s \,)^{\mathrm{T}},$$

and

$$[\boldsymbol{a}\,|\,A] = \left(\begin{array}{c|ccc} \alpha_0^{(1)} & \alpha_1^{(1)} & \dots & \alpha_k^{(1)} \\ \vdots & \vdots & & \vdots \\ \alpha_0^{(k_1-1)} & \alpha_1^{(k_1-1)} & \dots & \alpha_k^{(k_1-1)} \\ \alpha_0 & \alpha_1 & \dots & \alpha_k \end{array}\right.$$
$$\left.\begin{array}{cccc} & & & \\ & & & \\ & & & \\ \alpha_0 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & \ddots \\ & \alpha_0 & \dots & \alpha_k \\ & \alpha_0^{(s-k_2+1)} & \dots & \alpha_k^{(s-k_2+1)} \\ & \vdots & & \vdots \\ & \alpha_0^{(s)} & \dots & \alpha_k^{(s)} \end{array}\right)_{s\times(s+1)},$$

$$[\boldsymbol{b}\,|\,B] = \left(\begin{array}{c|ccc} \beta_0^{(1)} & \beta_1^{(1)} & \dots & \beta_k^{(1)} \\ \vdots & \vdots & & \vdots \\ \beta_0^{(k_1-1)} & \beta_1^{(k_1-1)} & \dots & \beta_k^{(k_1-1)} \\ \beta_0 & \beta_1 & \dots & \beta_k \end{array}\right.$$
$$\left.\begin{array}{cccc} & & & \\ & & & \\ & & & \\ \beta_0 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & \ddots \\ & \beta_0 & \dots & \beta_k \\ & \beta_0^{(s-k_2+1)} & \dots & \beta_k^{(s-k_2+1)} \\ & \vdots & & \vdots \\ & \beta_0^{(s)} & \dots & \beta_k^{(s)} \end{array}\right)_{s\times(s+1)}.$$

One then concludes that (9) can also be regarded as a one-step composite method. This feature has led to define a block version of BVMs [11], which has been successfully used for approximating Hamiltonian problems.

In few words, such a block version amounts to discretizing the interval $[t_0, T]$ by using two different meshes: a coarser one and a finer one. Let the coarser mesh contain the $p+1$ points

$$\tau_i = \tau_{i-1} + \hat{h}_i, \qquad i = 1,\dots,p, \quad \tau_0 \equiv t_0, \quad \tau_p \equiv T.$$

Then, on each subinterval $[\tau_{i-1}, \tau_i]$, $i = 1,\dots,p$, we apply the same (composite) BVM, as described above, by using the finer stepsize $h_i = \hat{h}_i/s$.

As a consequence, the points in the finer mesh belonging to the subinterval $(\tau_{i-1}, \tau_i]$, which we call *internal steps*, are given by

$$t_{ji} = \tau_{i-1} + jh_i, \qquad j = 1,\dots,s, \quad i = 1,\dots,p,$$

where the rightmost lower index of $t_{ji}$ identifies the $i$th subinterval. In this case, we speak about a *block BVM with s internal steps.*

## 3. The parallel algorithm

From the previous arguments, it is evident that BVMs present a natural parallelism in time. In fact, when solving problem (9), one looks for simultaneous approximations of the discrete solution at several grid points. An efficient parallel solution of such equation would then lead us to define a parallel ODE solver which could be regarded as having both a parallelism across the method, and a parallelism across the steps. Parallel ODE solvers of this kind have already been considered in the past years [3, 8, 16]. Instead, we shall here consider a different approach, which will gain parallelism from the block version of BVMs.

In this paper we shall restrict our analysis to the application of the methods to linear problems, because in such a case the discrete problem obtained is linear. In the more general case of nonlinear problems, the discrete problem is nonlinear, and some iterative procedure (e.g., Newton's method) should be considered for its solution. As a consequence, disregarding for the moment the convergence of the iterative procedure, the following arguments could be applied to the linearized problem obtained at each step.

Let us then consider the linear problem in $\mathbb{R}^m$,

$$y' = L(t)y + g(t), \qquad t \in (t_0, T], \quad y(t_0) = \eta. \tag{10}$$

Moreover, in order to avoid unnecessary complications, we shall also suppose the points in the coarser mesh to be equally spaced. Let us introduce, for $i = 1, \ldots, p$, the block vectors (see (9)),

$$\boldsymbol{v}_i = \boldsymbol{a} \otimes I_m - h\boldsymbol{b} \otimes L_{0i},$$

where, for any integer $r$, $I_r$ denotes the identity matrix of size $r$, and $L_{ji} \equiv L(t_{ji})$. Moreover, we define the $sm \times sm$ matrices

$$V_i = [\hat{O}_{s,s-1} \mid \boldsymbol{v}_i],$$

$$M_i = A \otimes I_m - h(B \otimes I_m) \begin{pmatrix} L_{1i} & & \\ & \ddots & \\ & & L_{si} \end{pmatrix},$$

where, by denoting by $O_{s,s-1}$ the $s \times (s-1)$ zero matrix and, for any integer $r$, with $O_r$ the $r \times r$ zero matrix,

$$\hat{O}_{s,s-1} = O_{s,s-1} \otimes O_m.$$

Then, the application of a block BVM to problem (10) leads to the following discrete problem:

$$M^{(p)}\boldsymbol{y}^{(p)} = \boldsymbol{g}^{(p)}, \tag{11}$$

where

$$M^{(p)} = \begin{pmatrix} I_m & & & \\ v_1 & M_1 & & \\ & V_2 & M_2 & \\ & & \ddots & \ddots \\ & & & V_p \; M_p \end{pmatrix}, \qquad y^{(p)} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_p \end{pmatrix}, \qquad g^{(p)} = \begin{pmatrix} \eta \\ hg_1 \\ hg_2 \\ \vdots \\ hg_p \end{pmatrix},$$

$y_i$ contains the approximations at the internal steps in the $i$th subinterval, and $g_i$ is a block vector containing suitable combinations of the inhomogeneity in (10) at the points in the same subinterval.

The parallel implementation of block BVMs will exploit the particular structure of the coefficient matrix $M^{(p)}$. In fact, in order the discrete solution be defined (as we obviously assume), all the square diagonal blocks $M_i$ must be nonsingular. As a consequence, we may consider the following factorization:

$$M^{(p)} = T^{(p)} S^{(p)}, \tag{12}$$

where

$$T^{(p)} = \begin{pmatrix} I_m & & & \\ & M_1 & & \\ & & \ddots & \\ & & & M_p \end{pmatrix}, \qquad S^{(p)} = \begin{pmatrix} I_m & & & \\ w_1 & \hat{I}_s & & \\ & W_2 & \hat{I}_s & \\ & & \ddots & \ddots \\ & & & W_p \; \hat{I}_s \end{pmatrix},$$

having denoted, for any integer $r$, with $\hat{I}_r = I_r \otimes I_m$ and, for all allowed $i$, $W_i = [\hat{O}_{s,s-1} \mid w_i]$, while $w_i$ is the solution of the linear system

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

We here assume that such systems are solved by means of the $LU$ factorization with partial pivoting algorithm.

As a consequence, Eq. (11) is equivalent to solving the following ones:

$$T^{(p)} x^{(p)} = g^{(p)}, \qquad S^{(p)} y^{(p)} = x^{(p)} \equiv \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_p \end{pmatrix},$$

i.e.,

$$x_0 = \eta, \qquad M_i x_i = hg_i, \quad i = 1, \ldots, p, \tag{14}$$

$$y_0 = x_0, \qquad y_1 = x_1 - w_1 y_0, \qquad y_i = x_i - W_i y_{i-1}, \quad i = 2, \ldots, p. \tag{15}$$

By the way, we observe that Eq. (14) corresponds to the parallel approximation of the $p$ IVPs

$$x' = L(t)x + g(t), \qquad t \in (\tau_{i-1}, \tau_i], \qquad x(\tau_{i-1}) = 0, \qquad i = 1, \ldots, p.$$

In order to examine Eq. (15), we need to further partition the vectors $x_i$, $y_i$ and $w_i$ as follows:

$$x_i = \begin{pmatrix} \hat{x}_i \\ x_{si} \end{pmatrix}, \quad y_i = \begin{pmatrix} \hat{y}_i \\ y_{si} \end{pmatrix}, \quad w_i = \begin{pmatrix} \hat{w}_i \\ w_{si} \end{pmatrix}, \qquad i = 1, \ldots, p,$$

where $x_{si}, y_{si} \in \mathbb{R}^m$ and $w_{si} \in \mathbb{R}^{m \times m}$. The existing parallelism is then emphasized by considering the following $(ps+1)m \times (ps+1)m$ permutation matrix:

$$P = \begin{pmatrix}
\hat{\mathbf{0}}_{s-1} & \hat{I}_{s-1} & \hat{\mathbf{0}}_{s-1} & \cdots & & & & \\
\hat{\mathbf{0}}_{s-1} & \hat{O}_{s-1} & \hat{\mathbf{0}}_{s-1} & \hat{I}_{s-1} & \hat{\mathbf{0}}_{s-1} & \cdots & & \\
\hat{\mathbf{0}}_{s-1} & \hat{O}_{s-1} & \hat{\mathbf{0}}_{s-1} & \hat{O}_{s-1} & \hat{\mathbf{0}}_{s-1} & \hat{I}_{s-1} & \hat{\mathbf{0}}_{s-1} & \cdots \\
 & & \cdots & \cdots & & & & \\
I_m & \hat{\mathbf{0}}_{s-1}^{\mathrm{T}} & \cdots & & & & & \\
O_m & \hat{\mathbf{0}}_{s-1}^{\mathrm{T}} & I_m & \hat{\mathbf{0}}_{s-1}^{\mathrm{T}} & \cdots & & & \\
O_m & \hat{\mathbf{0}}_{s-1}^{\mathrm{T}} & O_m & \hat{\mathbf{0}}_{s-1}^{\mathrm{T}} & I_m & \hat{\mathbf{0}}_{s-1}^{\mathrm{T}} & \cdots & \\
 & & \cdots & \cdots & & & &
\end{pmatrix}, \tag{16}$$

where $\hat{\mathbf{0}}_{s-1} = \mathbf{0}_{s-1} \otimes O_m$, and $\hat{O}_{s-1} = O_{s-1} \otimes O_m$. In fact, the solution of (15) is equivalent to the permuted system

$$P S^{(p)} P^{\mathrm{T}} P y^{(p)} = P x^{(p)},$$

i.e.,

$$\left(\begin{array}{cccc|cccc}
\hat{I}_{s-1} & & & & \hat{w}_1 & O_m & & \\
 & \hat{I}_{s-1} & & & & \hat{w}_2 & O_m & \\
 & & \ddots & & & & \ddots & \ddots \\
 & & & \hat{I}_{s-1} & & & & \hat{w}_p \; O_m \\
\hline
 & & & & I_m & & & \\
 & & & & w_{s1} & I_m & & \\
 & & & & & w_{s2} & I_m & \\
 & O & & & & & \ddots & \ddots \\
 & & & & & & & w_{sp} \; I_m
\end{array}\right)
\left(\begin{array}{c} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_p \\ \hline y_0 \\ y_{s1} \\ \vdots \\ y_{sp} \end{array}\right)
=
\left(\begin{array}{c} \hat{x}_1 \\ \hat{x}_2 \\ \vdots \\ \hat{x}_p \\ \hline x_0 \\ x_{s1} \\ \vdots \\ x_{sp} \end{array}\right). \tag{17}$$

By considering the matrix

$$R_p = \begin{pmatrix}
I_m & & & \\
w_{s1} & I_m & & \\
 & w_{s2} & I_m & \\
 & & \ddots & \ddots \\
 & & & w_{sp} \; I_m
\end{pmatrix}, \tag{18}$$

the solution of (17) is obtained by first solving the *reduced system*

$$R_p \begin{pmatrix} y_0 \\ y_{s1} \\ \vdots \\ y_{sp} \end{pmatrix} = \begin{pmatrix} x_0 \\ x_{s1} \\ \vdots \\ x_{sp} \end{pmatrix}, \tag{19}$$

and then updating in parallel the right-hand side (let $y_{s0} \equiv y_0$),

$$\hat{y}_i = \hat{x}_i - \hat{w}_i y_{s,i-1}, \qquad i = 1, \ldots, p. \tag{20}$$

As a consequence, one has that only the reduced system (19) cannot be directly solved in parallel. The parallel solution of such system, however, can be obtained by using a block cyclic reduction, even if also in this case the degree of parallelism is not constant.

Finally, we observe that the dimension of the matrix $R_p$ is $(p+1)m \times (p+1)m$, where $p$ is the number of the parallel processors, and $m$ is the dimension of the continuous problem. It is then independent of both the number $s$ of internal steps, and the number $k$ of steps of the main formula of the block BVM.

## 4. Parallel solution of two-point BVPs

Block BVMs can be also used for approximating continuous two-point BVPs. By considering the following linear problem:

$$y' = L(t)y + g(t), \qquad B_a y(a) + B_b y(b) = \eta,$$

where $B_a$ and $B_b$ are $m \times m$ matrices, the obtained discrete problem is still given by (11), with the only difference that, now,

$$M^{(p)} = \begin{pmatrix} B_a & & & (\hat{\mathbf{0}}_{s-1}^{\mathrm{T}} \; B_b) \\ v_1 & M_1 & & \\ & V_2 & M_2 & \\ & & \ddots & \ddots \\ & & V_p & M_p \end{pmatrix}. \tag{21}$$

However, in this case a factorization similar to (12) cannot be used, because of stability reasons, since the diagonal blocks $M_i$ may be very ill-conditioned or even singular.

For this reason, let us consider the following partitioning of the coefficient matrix (21):

$$M^{(p)} = \begin{pmatrix} B_a & & & & B_b \\ v_1 & N_1 & z_1 & & \\ & v_2 & N_2 & z_2 & \\ & & & \ddots & \\ & & & v_p & N_p \; z_p \end{pmatrix}, \tag{22}$$

where each block $N_i$ has size $sm \times (s-1)m$ and must have full column rank, in order the matrix $M^{(p)}$ is nonsingular. As a consequence, $N_i$ can be factored as

$$N_i = Q_i L_i \begin{pmatrix} \hat{U}_i \\ \hat{\mathbf{0}}_{s-1}^{\mathrm{T}} \end{pmatrix}, \qquad i = 1, \ldots, p, \tag{23}$$

where $Q_i$ is an $sm \times sm$ permutation matrix, $L_i$ is lower triangular, and $\hat{U}_i$ is $(s-1)m \times (s-1)m$ upper triangular. We observe that the factorizations (23) are independent of each other and, therefore, they can be computed in parallel.

The subsequent step is the solution of the linear systems defining the block vectors $\boldsymbol{u}_i$ and $\boldsymbol{w}_i$:

$$Q_i L_i \boldsymbol{u}_i = \boldsymbol{z}_i, \quad Q_i L_i \boldsymbol{w}_i = \boldsymbol{v}_i, \qquad i = 1, \ldots, p,$$

which can be also computed in parallel. For convenience, we partition these vectors as follows:

$$\boldsymbol{u}_i = \begin{pmatrix} \hat{\boldsymbol{u}}_i \\ u_{si} \end{pmatrix}, \qquad \boldsymbol{w}_i = \begin{pmatrix} \hat{\boldsymbol{w}}_i \\ w_{si} \end{pmatrix}, \quad u_{si}, w_{si} \in \mathbb{R}^{m \times m}.$$

The above steps then lead to the following factorization of matrix (22):

$$M^{(p)} = L^{(p)} D^{(p)} U^{(p)}, \tag{24}$$

where

$$L^{(p)} = \begin{pmatrix} I_m & & & \\ & Q_1 L_1 & & \\ & & \ddots & \\ & & & Q_p L_p \end{pmatrix}, \qquad U^{(p)} = \begin{pmatrix} I_m & & & \\ & U_1 & & \\ & & \ddots & \\ & & & U_p \end{pmatrix},$$

$$U_i = \begin{pmatrix} \hat{U}_i & \hat{\boldsymbol{u}}_i \\ & I_m \end{pmatrix}, \qquad D^{(p)} = \begin{pmatrix} B_a & & & & & B_b \\ \hat{\boldsymbol{w}}_1 & \hat{I}_{s-1} & & & & \\ w_{s1} & & u_{s1} & & & \\ & & \hat{\boldsymbol{w}}_2 & \hat{I}_{s-1} & & \\ & & w_{s2} & & u_{s2} & \\ & & & \ddots & \ddots & \\ & & & & \hat{\boldsymbol{w}}_p & \hat{I}_{s-1} \\ & & & & w_{sp} & u_{sp} \end{pmatrix}.$$

At this point the solution of the linear systems with the matrices $L^{(p)}$ and $U^{(p)}$ can be easily performed in parallel on the $p$ processors. The existing parallelism in the solution of the system with the matrix $D^{(p)}$ requires the use of the permutation matrix (16), thus showing that the only sequential section amounts to the solution of a reduced system, with the reduced matrix

$$R_p = \begin{pmatrix} B_a & & B_b \\ w_{s1} & u_{s1} & \\ & \ddots & \ddots \\ & & w_{sp} \quad u_{sp} \end{pmatrix}. \tag{25}$$

However, we observe that matrix (25) has a structure similar to that of the original matrix (21), but smaller size. As a consequence, the same steps considered above can be repeated recursively, thus obtaining a cyclic reduction-like solution of the reduced system, as already proposed by Wright [22] for the solution of ABD systems (see also [4]).

## 5. Expected speedup

Let us now examine the parallel complexity of the algorithms described in the previous sections. For this purpose, observe that when a distributed memory parallel computer is used and its processors are numbered from 1 to $p$, the rightmost lower index in each mentioned quantity denotes the index of the processor where this quantity has to be stored. The only exception is for the vectors and the matrices involved in the initial or boundary conditions. In fact, they are all stored on the first processor, when the reduced system is sequentially solved. Conversely, they are stored on the processor performing the last step in the reduction of the reduced system, when the parallel approach is used.

The following estimates are obtained in the case where $p = 2^r$. In the general case, they are slightly modified.

The parallel algorithm derived by factorization (12) is summarized by the following steps, where we have assumed that $k_1 \approx k/2$, since this is the case for the most effective BVMs [1, 2, 10, 12]. Then, we derive a simplified model for the expected speedup on $p$ processors. The simplifying assumption consists in neglecting data communication time, thus considering only the number of *flops* (floating point operations). This is because data communication originates terms with a much smaller complexity.

*Step 1* (*Computation of the factorization* (12)): For all $i = 1, \ldots, p$, processor $i$ computes the $LU$ factorization of $M_i$ and solves the $i$th linear system in (13); $n_1 \approx psk^2m^3$ flops, equally redistributed on the $p$ parallel processors, are required.

*Step 2* (*Parallel solution of the system with the matrix* $T^{(p)}$): For all $i = 1, \ldots, p$, processor $i$ solves the $i$th linear system in (14); $n_2 \approx 2pskm^2$ flops, equally redistributed on the $p$ parallel processors, are required.

*Step 3* (*Solution of the reduced system* (18)–(19)): $n_3 \approx 2pm^2$ flops are required if a sequential solution of the reduced system is considered. Conversely, when it is solved by using block cyclic reduction, the parallel complexity is $n_3' \approx 2m^3 \log_2 p$ flops.

*Step 4* (*Parallel updates required by system* (17)): For all $i = 1, \ldots, p$, processor $i$ computes the $i$th update (20); $n_4 \approx 2psm^2$ flops, equally redistributed on the $p$ processors, are required.

Observe that step 3, that is the solution of the reduced system, represents the only synchronization point among the processors. In particular, when the reduced system is sequentially solved, $p - 1$ data communications of length $m$ are needed. Conversely, when a block cyclic reduction is used, one needs approximately $\log_2 p$ data communications of length $m^2$.

The standard $LU$ factorization with partial pivoting algorithm applied to problem (3) represents the scalar algorithm of comparison. It requires $n_s \approx psk^2m^3$ flops.

The following expression for the expected speedup of the parallel algorithm, over the sequential implementation of the same method, is then obtained,

$$S_p = \begin{cases} \dfrac{n_s}{\dfrac{n_1+n_2+n_4}{p}+n_3} \approx \dfrac{p}{1+2(sk^2m)^{-1}p} \\ \text{when the reduced system is solved sequentially,} \\ \dfrac{n_s}{\dfrac{n_1+n_2+n_4}{p}+n_3'} \approx \dfrac{p}{1+2(sk^2)^{-1}\log_2 p}, \\ \text{otherwise.} \end{cases} \tag{26}$$

From the above expressions, it is evident that the parallel solution of the reduced system via block cyclic reduction is conveniently used only if $n_3 > n_3'$, i.e., when

$$m < p/\log_2 p.$$

Consequently, it is convenient when the number of parallel processors $p$ is suitably large, and/or the size $m$ of the continuous problem is suitably small. In this case, the expected speedup is almost independent of $m$. Moreover, in both cases, the parallel efficiency grows with the number $k$ of steps of the main formula, and the number $s$ of internal steps of the block BVM.

We now briefly examine the case of two-point BVPs, thus analyzing the algorithm corresponding to the factorization (24). It can be shown that the complexity of the sequential $LU$ factorization algorithm applied to the system (11)–(21) is still given by $n_s \approx psk^2m^3$ flops. Moreover, concerning the parallel solver, it can be shown that the complexity of the operations with perfect degree of parallelism is essentially the same as in the previous case. The only significant difference consists in the solution of the reduced system with matrix (25). In fact, it requires approximately $12pm^3$ flops, if sequentially solved, or a parallel complexity of approximately $20/3\,m^3\log_2 p$ flops, when the cyclic reduction-like approach is used.

One easily verifies that, in this case, it is always convenient to consider the parallel solution of the reduced system, thus obtaining the following expression for the expected speedup on $p$ processors:

$$S_p \approx \frac{p}{1+20/3(sk^2)^{-1}\log_2 p}.$$

## 6. Numerical examples

We now consider some numerical examples, in order to show the effectiveness of the presented parallel implementation of block BVMs. We first consider two initial value problems, then we examine the solution of a boundary value problem.

In all the cases, the parallel solution of the reduced system has been considered. Moreover, the chosen BVMs are extended trapezoidal rules (ETRs) [2], i.e., methods having the following form:

$$y_n - y_{n-1} = h\sum_{i=-v}^{v-1}\beta_{i+v}f_{n+i}, \qquad n = v,\ldots,s-v+1, \quad v \geqslant 1.$$

In this case, $k = 2v-1$ ($k_1 = v$, $k_2 = v-1$), and the coefficients $\{\beta_i\}$ are uniquely determined by imposing a $O(h^{k+2})$ truncation error. The above formula is conveniently used with the following

additional initial equations:

$$y_j - y_{j-1} = h\sum_{i=0}^{k} \beta_i^{(j)} f_i, \qquad j = 1,\dots,v-1,$$

and the following additional final ones:

$$y_j - y_{j-1} = h\sum_{i=0}^{k} \beta_{k-i}^{(j)} f_{s-i}, \qquad j = s-v+2,\dots,s.$$

The coefficients of the additional methods are uniquely determined by imposing the same order $k+1$ of the main formula.

Such methods are suitably used either for approximating Hamiltonian problems [7, 11, 20], or continuous BVPs [9].

The parallel computer used is a transputer based machine, which is a distributed memory parallel computer. Its nodes communicate through four physical channels called *links*.

The first problem is

$$y' = \begin{pmatrix} -21 & 19 & -20 \\ 19 & -21 & 20 \\ 40 & -40 & -40 \end{pmatrix} y, \qquad y(0) = \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}, \quad t \in [0,5]. \tag{27}$$

The second problem is a Hamiltonian one

$$y' = \begin{pmatrix} & -I_5 \\ I_5 & \end{pmatrix} Sy, \qquad y(0) = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \quad t \in [0,50], \tag{28}$$

where $S = 8I_{10} + Q$, and the $(i,j)$th entry of $Q$ is $i+j$, $i,j = 1,\dots,10$.

In both cases, a uniform mesh has been considered, and the speedup over the sequential implementation of the methods is computed. In this case, the time for the parallel execution includes both computations and communications.

In Table 1 we report the measured speedups on $p = 1,2,4,8,16$ processors for problem (27), while Table 2 summarizes the results for problem (28). For both problems, we have considered ETRs of different order ($k = 3,5,7,9$) and two values for the number of internal steps ($s = 20,40$) in order to observe the predicted growth of the speedup with $k$ and $s$. In this case, since we are considering linear problems, there is no difference (up to machine precision) between the solution computed on one processor, and those computed in parallel. Consequently, the accuracy of the computed solutions completely reflects the order of the methods.

From Tables 1 and 2, one can also observe a moderate growth of the speedup with the dimension $m$ of the continuous problem, not predicted by our simplified model (26). However, the most interesting feature is that the proposed parallel solver is very effective, since all the speedups are very close to $p$. Moreover, the parallel solution of the reduced system allows the use of parallel computers with a large number $p$ of processors, since the complexity for its solution grows as $\log_2 p$.

Table 1
Measured speedups for problem (27)

| $s$ | 20 | | | | 40 | | | |
|---|---|---|---|---|---|---|---|---|
| $p \setminus k$ | 3 | 5 | 7 | 9 | 3 | 5 | 7 | 9 |
| 1 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| 2 | 1.89 | 1.93 | 1.94 | 1.95 | 1.94 | 1.96 | 1.97 | 1.98 |
| 4 | 3.55 | 3.68 | 3.77 | 3.83 | 3.74 | 3.84 | 3.88 | 3.92 |
| 8 | 6.42 | 6.93 | 7.20 | 7.44 | 7.11 | 7.43 | 7.58 | 7.72 |
| 16 | 12.46 | 13.52 | 14.13 | 14.52 | 14.03 | 14.70 | 14.98 | 15.24 |

Table 2
Measured speedups for problem (28)

| $s$ | 20 | | | | 40 | | | |
|---|---|---|---|---|---|---|---|---|
| $p \setminus k$ | 3 | 5 | 7 | 9 | 3 | 5 | 7 | 9 |
| 1 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| 2 | 1.95 | 1.99 | 1.99 | 1.99 | 1.99 | 1.99 | 1.99 | 1.99 |
| 4 | 3.77 | 3.91 | 3.94 | 3.96 | 3.94 | 3.94 | 3.97 | 3.97 |
| 8 | 7.34 | 7.67 | 7.77 | 7.85 | 7.67 | 7.78 | 7.88 | 7.90 |
| 16 | 14.16 | 15.05 | 15.34 | 15.55 | 15.10 | 15.41 | 15.67 | 15.72 |

Finally, let us consider the following second-order boundary value problem ($\varepsilon = 10^{-3}$):

$$\begin{aligned} &\varepsilon u'' = u, \\ &\varepsilon y'' = t + \tfrac{1}{2}tu' - \tfrac{1}{2}ty' - \varepsilon\pi^2\cos(\pi t) - \tfrac{1}{2}t\pi\sin(\pi t), \\ &u(-1) = -y(-1) = 1, \qquad u(1) = y(1) = e^{-2/\sqrt{\varepsilon}}. \end{aligned} \tag{29}$$

It is solved after recasting as a first-order system. In this case, we fix the number $s = 40$ of internal steps for the considered BVMs. Then, we solve the above problem on $p$ processors, by using a constant stepsize $h = (ps)^{-1}$, $p = 1, 2, 4, 8, 16$. This means that when the stepsize is halved, and consequently the size of the discrete problem is doubled, the number of the parallel processors used is also doubled. As a consequence, we expect the execution time to remain approximately constant for increasing number of processors, even if the accuracy of the solution improves, due to the use of a smaller stepsize.

In Table 3 we report the measured execution times, expressed in units of time (*ticks*), each corresponding to 64 $\mu$s. Finally, in Table 4 the maximum absolute errors are reported. As predicted, as the number $p$ of processors increases, the maximum error decreases.

Concerning Table 3, it is worth mentioning that the execution times on multiple processors are often smaller than the execution times on one processor. This is due to the fact that, as the stepsize is decreased (i.e., $p$ increases), the discrete problem changes, and the $LU$ factorization algorithm requires less permutations for pivoting.

Table 3
Measured execution times for problem (29)

| $p \setminus k$ | 3 | 5 | 7 | 9 |
|---|---|---|---|---|
| 1 | 4217 | 6137 | 8482 | 10899 |
| 2 | 3955 | 5822 | 8571 | 10993 |
| 4 | 4110 | 5987 | 8007 | 10259 |
| 8 | 4176 | 6043 | 8116 | 10404 |
| 16 | 4265 | 6030 | 8151 | 10551 |

Table 4
Measured maximum absolute errors for problem (29)

| $p \setminus k$ | 3 | 5 | 7 | 9 |
|---|---|---|---|---|
| 1 | 3.6e−2 | 1.2e−2 | 7.5e−03 | 5.5e−03 |
| 2 | 3.8e−3 | 9.3e−4 | 1.2e−04 | 5.0e−04 |
| 4 | 2.4e−4 | 1.4e−5 | 3.5e−06 | 1.6e−06 |
| 8 | 1.1e−5 | 1.9e−7 | 3.9e−09 | 4.5e−10 |
| 16 | 8.3e−7 | 3.1e−9 | 2.1e−11 | 5.6e−13 |

## Acknowledgements

The authors are very indebted to Professor Donato Trigiante for the help in the preparation of the manuscript.

## References

[1] P. Amodio, $A$-stable $k$-step linear multistep formulae of order $2k$ for the solution of stiff ODEs, submitted.
[2] P. Amodio and F. Mazzia, A boundary value approach to the numerical solution of ODEs by multistep methods, *J. Difference Equations Appl.* **1** (1995) 353–367.
[3] P. Amodio and F. Mazzia, Parallel block preconditioning for the solution of boundary value methods, *J. Comput. Appl. Math.* **69** (1996) 191–206.
[4] P. Amodio and M. Paprzycki, A cyclic reduction approach to the numerical solution of boundary value ODEs, *SIAM J. Sci. Comput.*, (to appear).
[5] A.O.H. Axelsson and J.G. Verwer, Boundary value techniques for initial value problems in ordinary differential equations, *Math. Comput.* **45** (1985) 153–171.
[6] A. Bellen and M. Zennaro, Parallel algorithms for initial-value problems for difference and differential equations, *J. Comput. Appl. Math.* **25** (1989) 341–350.
[7] L. Brugnano, Essentially symplectic boundary value methods for linear Hamiltonian systems, *J. Comput. Math.*, to appear.
[8] L. Brugnano and D. Trigiante, A parallel preconditioning technique for boundary value methods, *Appl. Numer. Math.* **13** (1993) 277–290.
[9] L. Brugnano and D. Trigiante, High order multistep methods for boundary value problems, *Appl. Numer. Math.* **18** (1995) 79–94.
[10] L. Brugnano and D. Trigiante, Convergence and stability of boundary value methods for ordinary differential equations, *J. Comput. Appl. Math.* **66** (1996) 97–109.

[11] L. Brugnano and D. Trigiante, Block boundary value methods for linear Hamiltonian systems, *J. Appl. Math.*, to appear.
[12] L. Brugnano and D. Trigiante, *Solving Differential Problems by Multistep Initial and Boundary Value Methods*, in preparation.
[13] K. Burrage, *Parallel and Sequential Methods for Ordinary Differential Equations* (Clarendon Press, Oxford, 1995).
[14] J.R. Cash, *Stable Recursions* (Academic Press, London, 1979).
[15] L. Lopez and D. Trigiante, Boundary value methods and BV-stability in the solution of initial value problems, *Appl. Numer. Math.* **11** (1993) 225–239.
[16] F. Mazzia, Boundary value methods for the initial value problems: parallel implementation. *Ann. Numer. Math.* **1** (1994) 439-450.
[17] W.L. Miranker and W. Liniger, Parallel methods for the numerical integration of ordinary differential equations, *Math. Comput.* **21** (1967) 303–320.
[18] J. Nievergelt, Parallel methods for integrating ordinary differential equations, *Comm. ACM* **7** (1964) 731–733.
[19] B.P. Sommeijer, W. Couzy and P.J. van der Houwen, *A*-stable parallel block methods for ordinary and integro-differential equations, *Appl. Numer. Math.* **9** (1992) 267–281.
[20] D. Trigiante, Multipoint methods for linear Hamiltonian systems, in: *Advances in Nonlinear Dynamics*, series on Stability and Control: Theory Methods and Applications, (Gordon and Breach, Reading, UK, 1996).
[21] P.J. van der Houwen, B.P. Sommeijer and W.A. van der Veen, Parallel Iteration across the steps of high-order Runge–Kutta methods for nonstiff initial value problems, *J. Comput. Appl. Math.* **60** (1995) 309–329.
[22] S.J. Wright, Stable parallel elimination for boundary value ODEs, *Numer. Math.* **67** (1994) 521–535.