Journal of Computational and Applied Mathematics 25 (1989) 341–350
North-Holland

Parallel algorithms for initial-value problems for difference and differential equations *

Alfredo BELLEN
Dipartimento di Scienze Matematiche, Università di Trieste, I-34100 Trieste, Italy

Marino ZENNARO
Dipartimento di Matematica e Informatica, Università di Udine, I-33100 Udine, Italy

Received 3 May 1988

Abstract: Let $\{y_n\}$ be a trajectory defined sequentially by a nonlinear vector difference equation $y_{n+1} = F_{n+1}(y_n)$ with y_0 known. A Steffensen-like iterative method is proposed which starts from a guessed sequence $\{u_n^{(0)}\}$ for the approximation of $\{y_n\}$ and allows certain computations to be performed in parallel. The sequence $\{u_n^{(k+1)}\}$ is obtained from $\{u_n^{(k)}\}$ by means of a formula of the form $u_{n+1}^{(k+1)} = v_{n+1}^{(k+1)} + \Lambda_{n+1}^{(k+1)}(u_n^{(k+1)} - u_n^{(k)})$. Here the vectors $v_{n+1}^{(k+1)}$ and the matrices $\Lambda_{n+1}^{(k+1)}$ each require a function evaluation but can be computed in parallel with respect to n in a suitable interval $[N_k, M_k]$, the length of which depends on the number of processors available. Therefore, for the algorithm to serve effectively, the Steffensen iteration must converge quickly and the machine used must possess a large number of processors. Finally, note that the theory includes the case that the sequence $\{y_n\}$ is given by a one-step ODE solver.

Keywords: Parallel computing, difference equations, differential equations, IVPs.

1. Introduction

In this paper we are concerned with the potential for parallelism in an initial-value problem (IVP) for difference equations of the form

$$y_{n+1} = F_{n+1}(y_n), \qquad y_0 \text{ known}, \tag{1.1}$$

where $y_n \in \mathbb{R}^m$ and $F_{n+1}: \mathbb{R}^m \to \mathbb{R}^m$ for every $n \geqslant 0$. Note that the problem includes as an important particular case the solution of an IVP for ordinary differential equations (ODEs) by means of a one-step numerical method. For example, if $y: [t_0, T] \to \mathbb{R}^m$ and $f: [t_0, T] \times \mathbb{R}^m \to \mathbb{R}^m$ satisfy

$$y'(t) = f(t, y(t)), \qquad y(t_0) = y_0, \tag{1.2}$$

an s-stage Runge–Kutta (RK) method

$$K_i^{(n+1)} = f\left(t_n + c_i h_{n+1},\ y_n + h_{n+1} \sum_{j=1}^{s} a_{ij} K_j^{(n+1)}\right), \quad i = 1, \ldots, s, \tag{1.3a}$$

$$y_{n+1} = y_n + h_{n+1} \sum_{i=1}^{s} b_i K_i^{(n+1)} \tag{1.3b}$$

* This work was supported by the Italian C.N.R. within the Project "Calcolo parallelo".

can be employed to get a sequence $\{ y_n \}$ of approximations to the solution $y(t)$ at the points t_n of a given grid. It is clear that the RK-method (1.3) defines a sequence of functions F_{n+1} which map y_n to y_{n+1}.

In spite of the recursive definition of the sequence $\{ y_n \}$, there are some types of parallelism which can be introduced for (1.1). For example, considering ODE solvers, Gear [4] has detected two types of parallelism:

(a) across the method;

(b) across the problem.

Specifically, parallelism across the method can be achieved by computing the RK stages in (1.3a) on different processors. On the other hand, parallelism across the problem can be introduced by assigning each equation in (1.2) to a different processor. For additional information on these ideas, the reader is referred to the works of El-Tarazi [3], Vinokurov [11], Lie [7], Karakashian [6] and to the references in [4].

In addition to the two types of parallelism mentioned above, we wish to isolate a third which is analogous to what Gear [5] has more recently called parallelism *across the time.* Here it is more appropriately called parallelism

(c) across the steps.

In fact, the algorithm we propose is a realization of this kind of parallelism. Without discussing it in detail here, we want to point out that the idea is indeed that of multiple shooting and parallelism is introduced at the cost of redundancy of computation. This strategy was already considered by Bellen and Zennaro [1] for linear ordinary and delay differential equations and by Nievergelt [8] for nonlinear ODEs (1.2), although the latter developed it in a different direction. We replace the nonlinear recursive formula (1.1) by a sequence of affine recursive formulae, obtained by the application of the Steffensen iterative method, until the approximate values are sufficiently close to the y_n's. The ingredients of the affine formulae can be computed in parallel. Moreover, well-known parallel techniques are applied to link conveniently these ingredients (see, for example, [10]). These two basic facts lead, in the case of fast convergence of the iterative method, to achieve significantly high speedup factors. Of course, the Steffensen method could be replaced by other iterative methods like Newton's or some modifications of it, according to the availability and complexity of the Jacobians $\partial F_n / \partial y$.

For a first and merely indicative estimate of the effectiveness of our algorithm, we do not consider possible complications and time delays arising from the communication problems among the various processors and memories. We make the naive assumption that computations carried out concurrently take the time of just one of them. This is the basis for our estimate of the speedup.

Finally, note that in this paper we are concerned with the general problem (1.1) and we do not treat in detail the particular case of ODEs. This will be done in a forecoming paper, where we shall produce some parallel algorithms including also step-size control strategies. In any case, if one is satisfied with fixed step ODE solvers, then the contents of this paper are sufficient.

2. The Steffensen iteration

Throughout the paper, we assume that all the functions F_n in the difference equation (1.1) are twice continuously differentiable. In order to take advantage of the parallelism across the steps,

we consider (1.1) as a fixed-point problem in the space of the m-vector sequences. Specifically, if $\{y_n\} := \{y_0, y_1, \ldots, y_n, \ldots\}$ is an m-vector sequence, then (1.1) is equivalent to

$$\{y_n\} = \Phi(\{y_n\}),$$

where the function Φ is defined by

$$\Phi(\{y_n\}) := \{y_0, F_1(y_0), \ldots, F_n(y_{n-1}), \ldots\}.$$

Then we apply the well-known Steffensen iterative method starting from a guessed sequence $\{u_n^{(0)}\}$, where we naturally choose $u_0^{(0)} := y_0$. The $(k+1)$th iteration yields the sequence $\{u_n^{(k+1)}\}$ from the sequence $\{u_n^{(k)}\}$ by means of the formula

$$\{u_n^{(k+1)}\} = \Phi(\{u_n^{(k)}\}) + \Delta\Phi(\{u_n^{(k)}\})(\{u_n^{(k+1)}\} - \{u_n^{(k)}\}),$$

where $\Delta\Phi(\{u_n^{(k)}\})$ is an approximation to the differential $D\Phi(\{u_n^{(k)}\})$. More precisely, the iteration looks as follows:

$$v_0^{(k+1)} := u_0^{(k)} \quad \text{and} \quad v_n^{(k+1)} := F_n(u_{n-1}^{(k)}), \quad n \geqslant 1; \tag{2.1a}$$

$$v_{j,n}^{(k+1)} := \left((u_n^{(k)})_1, \ldots, (v_n^{(k+1)})_j, \ldots, (u_n^{(k)})_m\right)^{\mathrm{T}}, \quad j = 1, \ldots, m \quad \text{and} \quad n \geqslant 0 \tag{2.1b}$$

(note that $v_{j,n}^{(k+1)}$ differs from $u_n^{(k)}$ only in the jth component);

$$w_{j,n+1}^{(k+1)} := F_{n+1}(v_{j,n}^{(k+1)}), \quad j = 1, \ldots, m \text{ and } n \geqslant 0; \tag{2.1c}$$

$$(A_{n+1}^{(k+1)})_{ij} := \begin{cases} \dfrac{(w_{j,n+1}^{(k+1)} - v_{n+1}^{(k+1)})_i}{(v_n^{(k+1)} - u_n^{(k)})_j} & \text{if } (v_n^{(k+1)} - u_n^{(k)})_j \neq 0, \\ \dfrac{\partial (F_{n+1})_i}{\partial y_j}(u_n^{(k)}) & \text{otherwise,} \end{cases} \tag{2.1d}$$

$$i, j = 1, \ldots, m \text{ and } n \geqslant 0;$$

$$u_0^{(k+1)} := u_0^{(k)} \quad \text{and} \quad u_{n+1}^{(k+1)} := v_{n+1}^{(k+1)} + A_{n+1}^{(k+1)}(u_n^{(k+1)} - u_n^{(k)}), \quad n \geqslant 0. \tag{2.1e}$$

In (2.1) $(x)_j$ stands for the jth component of the m-vector x and $(A)_{ij}$ for the element of the $m \times m$-matrix A in the ith row and jth column.

Remark 2.1. It is easily seen that, since $u_0^{(0)} = y_0$, at the first iteration ($k = 0$) we get $u_1^{(1)} = v_1^{(1)} = y_1$. Similarly, at the $(k+1)$th iteration we get $u_{k+1}^{(k+1)} = v_{k+1}^{(k+1)} = y_{k+1}$. In other words, we get an exact value each iteration and the sequences $\{u_n^{(k)}\}$ are expected to converge to the exact solution $\{y_n\}$ of (1.1) as $k \to \infty$.

Remark 2.2. Note that (2.1e) is a recursive formula just as (1.1). Also, each of the four stages (2.1a–d) must preceed (2.1e) in a sequential fashion. However, within each of these stages, computations can be performed in parallel with respect to all the indices n, i, j. Hence if the ratio between the time taken to compute one step of (1.1) and the time taken to compute one step of (2.1e) is large, then there is a potential for computing a Steffensen iteration in parallel, with respect to a large number of steps, more quickly than using (1.1) directly for the same number of steps.

The Steffensen iteration above is the basis for a more realistic algorithm that will be proposed in the next section. So we complete this theoretical background with an estimate of the convergence rate.

For the sequence $\{\boldsymbol{u}_n^{(k)}\}$ define the *error*

$$\boldsymbol{e}_n^{(k)} := \boldsymbol{y}_n - \boldsymbol{u}_n^{(k)} \tag{2.2}$$

and the *maximum error*

$$E_n^{(k)} := \max_{\nu \leqslant n} \left\| \boldsymbol{e}_\nu^{(k)} \right\| \tag{2.3}$$

where $\|\cdot\|$ will be used to denote the maximum norm over $\mathbb{R}^m$.

According to the theory of the Steffensen method (see, for example, [9]), we have the following convergence result, the proof of which is standard and can be found in [2].

Theorem 2.3. *Formulae* (2.1) *define an iterative method which is locally quadratically convergent. More precisely, there exist positive constants* $\{C_n\}$, *whose magnitudes are determined by the first and second partial derivatives of the functions* $\boldsymbol{F}_\nu$, $\nu \leqslant n$, *such that, for every* $\{\boldsymbol{u}_n^{(0)}\}$ *in a suitable neighbourhood of the solution* $\{\boldsymbol{y}_n\}$,

$$E_{n+1}^{(k+1)} \leqslant C_{n+1}\left(E_n^{(k)}\right)^2 \quad \forall k \geqslant 0. \tag{2.4}$$

Remark 2.4. A careful look at the proof of Theorem 2.3 (see [2]) shows that the smaller are the first and second partial derivatives of the functions $\boldsymbol{F}_n$, the faster is the convergence of the iterative method and the larger is the neighbourhood of the solution $\{\boldsymbol{y}_n\}$ in which the convergence itself is guaranteed. In particular, linear problems are exactly solved at the first iteration for any initial sequence $\{\boldsymbol{u}_n^{(0)}\}$.

3. Practical implementation: the parallel algorithm

In view of Remarks 2.2 and 2.4, we have in mind problems of the type (1.1) in which the functions $\boldsymbol{F}_n$ are very expensive to compute and, at the same time, not large in their first and second partial derivatives:

In order to make practicable the iterative method defined by (2.1), there are mainly four points one is faced with, which impose the choice of a suitable strategy.

(i) There may be a mismatch between the number of processors available and the number which could be effectively utilized with the algorithm.

(ii) The method requires sufficiently accurate initial values $\boldsymbol{u}_n^{(0)}$.

(iii) We need a criterion to stop the iteration and, in view of (i), to advance with respect to the index n.

(iv) The term $(\boldsymbol{v}_n^{(k+1)} - \boldsymbol{u}_n^{(k)})_j$ in (2.1d) might vanish or lose many of its significant digits in the computer arithmetic.

We propose the following solutions to them:

(i) Observe that, as mentioned in Section 1, the evaluation of each function $\boldsymbol{F}_n$ may be done by using more than one processor, say p processors (parallelism across the method and across the problem). Moreover, the computation of the vectors $\boldsymbol{w}_{j,n+1}^{(k+1)}$, $j = 1, \ldots, m$, in (2.1c) and of the

columns $((\Lambda_{n+1}^{(k+1)})_{1j},\dots,(\Lambda_{n+1}^{(k+1)})_{mj})^{\mathrm{T}}$, $j=1,\dots,m$, in (2.1d) can be done concurrently with respect to the index j. Therefore, if our architecture possesses P processors, we shall be able to treat in parallel a number N of steps that does not exceed the ratio $P/(pm)$.

(ii) Apart from the first value $\boldsymbol{u}_0^{(0)}=\boldsymbol{y}_0$, the other guesses $\boldsymbol{u}_n^{(0)}$, $n\leqslant N$, could be inspired by some a priori qualitative knowledge of the behaviour of the sequence $\{\boldsymbol{y}_n\}$. This is the case, for instance, of models describing physical phenomena. Moreover, if it is known that $\{\boldsymbol{y}_n\}$ has a limit as $n\to\infty$ (in particular, some stiff ODEs), a good choice could be $\boldsymbol{u}_n^{(0)}:=\boldsymbol{y}_0$ for all $n\leqslant N$. Of course, there are also some situations in which no reasonable suggestions are available: in these cases one could adopt once again the strategy $\boldsymbol{u}_n^{(0)}:=\boldsymbol{y}_0$ for all $n\leqslant N$. In any case, convergence may not be rapid, but in view of Remark 2.1, it is guaranteed.

(iii) In a first approach, one could perform as many iterations as necessary to guarantee a maximum error $E_N^{(k)}$ less than a given ϵ and afterwards process in the same way the next N steps, and so on. On the other hand, a better approach is obtained as follows. Note first that the maximum error $E_n^{(k)}$ is nondecreasing with respect to n and, by Theorem 2.3, decreasing with respect to k for all n, provided that the initial guesses are given in a suitable neighbourhood of the solution $\{\boldsymbol{y}_n\}$. Therefore the largest index n such that $E_n^{(k)}$ fulfils the error test

$$E_n^{(k)}\leqslant\epsilon,\quad \epsilon \text{ fixed},$$

is a nondecreasing function of k. This fact suggests to us that once N values, say $\boldsymbol{u}_n^{(k)}$, $n=n_1,\dots,n_1+N-1$, have been computed, some initial segment, say $\boldsymbol{u}_n^{(k)}$, $n=n_1,\dots,n_2-1$, could be accepted (possibly all of them) so that at the next iteration we can shift forward and still process N steps. The first part of these new N steps should consist in values $\boldsymbol{u}_n^{(k)}$, $n=n_2,\dots,n_1+N-1$, which have already been iterated at least once, and the latter part $\boldsymbol{u}_n^{(k)}$, $n=n_1+N,\dots,n_2+N-1$, in new initial guesses $\boldsymbol{u}_n^{(0)}$, $n=n_1+N,\dots,n_2+N-1$. Note that here and below we adopt the notation $\boldsymbol{u}_n^{(k)}:=\boldsymbol{u}_n^{(0)}$ for $n\geqslant n_1+N$, i.e., unless accepted as sufficiently accurate, all values have their iteration index promoted even if they do not actually participate in the computations.

Now a procedure is to be given to decide how many values $\boldsymbol{u}_n^{(k)}$, $n=n_1,\dots,n_2-1$, to accept at each iteration and how many of the remaining ones are suitable to be reiterated or must be rejected and reinitialized. For this, we define the *local error*

$$\boldsymbol{\tau}_n^{(k)}:=\boldsymbol{v}_n^{(k+1)}-\boldsymbol{u}_n^{(k)} \tag{3.1}$$

and prove the following theorem.

Theorem 3.1. *For every $n\geqslant 1$ there exists a constant K_n depending on the first partial derivatives of the functions $\boldsymbol{F}_\nu$, $\nu\leqslant n$, such that, for $\{\boldsymbol{u}_n^{(k)}\}$ in a suitable neighbourhood of the solution $\{\boldsymbol{y}_n\}$, we have*

$$E_n^{(k)}\leqslant K_n\max_{\nu\leqslant n}\left\|\boldsymbol{\tau}_\nu^{(k)}\right\|. \tag{3.2}$$

Proof. By (2.2), we have

$$\begin{aligned}\boldsymbol{e}_{\nu+1}^{(k)}&=\boldsymbol{y}_{\nu+1}-\boldsymbol{v}_{\nu+1}^{(k+1)}+\boldsymbol{v}_{\nu+1}^{(k+1)}-\boldsymbol{u}_{\nu+1}^{(k)}\\&=\boldsymbol{F}_{\nu+1}(\boldsymbol{y}_\nu)-\boldsymbol{F}_{\nu+1}\left(\boldsymbol{u}_\nu^{(k)}\right)+\boldsymbol{\tau}_{\nu+1}^{(k)}\\&=D_{\nu+1}^{(k+1)}\boldsymbol{e}_\nu^{(k)}+\boldsymbol{\tau}_{\nu+1}^{(k)},\end{aligned} \tag{3.3}$$

where

$$D_{\nu+1}^{(k+1)}:=\int_0^1(\boldsymbol{F}_{\nu+1})'\left(\boldsymbol{u}_\nu^{(k)}+\vartheta\boldsymbol{e}_\nu^{(k)}\right)\mathrm{d}\vartheta.$$

So there exists a nondecreasing sequence of positive constants $\{\lambda_n\}$ depending respectively on the first partials of $\boldsymbol{F}_\nu$, $\nu \leqslant n$, such that

$$\left\| \boldsymbol{e}_{\nu+1}^{(k)} \right\| \leqslant \lambda_{\nu+1} \left\| \boldsymbol{e}_{\nu}^{(k)} \right\| + \left\| \boldsymbol{\tau}_{\nu+1}^{(k)} \right\|.$$

Hence, by (2.3), for $r \leqslant n-1$,

$$E_{r+1}^{(k)} \leqslant \lambda_{r+1} E_r^{(k)} + \max_{\nu \leqslant n} \left\| \boldsymbol{\tau}_{\nu}^{(k)} \right\|.$$

Therefore, by some standard analysis, the proof is complete. □

On the basis of (3.2) we fix a tolerance TOL and accept $\boldsymbol{u}_n^{(k)}$, $n = n_1, \ldots, n_2 - 1$, provided

$$\left\| \boldsymbol{\tau}_{\nu}^{(k)} \right\| \leqslant \mathrm{TOL} \quad \text{for all } \nu \leqslant n_2 - 1. \tag{3.4}$$

In such a way, on any finite segment $[0, n^*]$, we get approximations z_n such that the errors $\| y_n - z_n \|$ are uniformly bounded by a constant K_{n^*} times the tolerance TOL. This strategy is quite similar to the step-size control strategy employed by the ODE solvers. Of course, the constant K_{n^*} is unknown although, but as we shall see in Section 4, a very small amount of extra computation allows us to estimate the error.

As for the unaccepted values, we decide to keep for reiterating only those values satisfying

$$\max_{\nu \leqslant n} \left\| \boldsymbol{\tau}_{\nu}^{(k)} \right\| \leqslant \max_{\nu \leqslant n} \left\| \boldsymbol{\tau}_{\nu}^{(k-1)} \right\|. \tag{3.5}$$

From this it is expected, at least on heuristic grounds, that $E_n^{(k)}$ is bounded by $E_n^{(k-1)}$ and that the values $\boldsymbol{u}_\nu^{(k)}$, $\nu \leqslant n$, are sufficiently accurate to insure rapid convergence of the iterative process. If (3.5) fails for some n_3, then under conditions mentioned below, we reset $\boldsymbol{u}_n^{(k)}$, $n \geqslant n_3$, to their initial values.

Now it may happen that among the current N values $\boldsymbol{u}_n^{(k)}$, $n = n_1, \ldots, n_1 + N - 1$, only a small number $n_2 - n_1$ pass the error test (3.4). Furthermore, it could happen that among the values $\boldsymbol{u}_n^{(k)}$, $n = n_2, \ldots, n_1 + N - 1$, a relatively large number $n_3 - n_2$ pass the error test (3.5), suggesting that $\boldsymbol{u}_n^{(k)}$, $n = n_3, \ldots, n_1 + N - 1$, be reinitialized. Then in order to fully exploit the parallel architecture, it seems natural to assign initial values to $\boldsymbol{u}_n^{(k)}$, $n = n_3, \ldots, n_2 + N - 1$, and to perform the next iteration with $\boldsymbol{u}_n^{(k)}$, $n = n_2, \ldots, n_2 + N - 1$. However, in this case, only a few extra function evaluations $\boldsymbol{v}_{n+1}^{(k+1)} = \boldsymbol{F}_{n+1}(\boldsymbol{u}_n^{(k)})$, $n = n_3, \ldots, n_2 + N - 1$, need be computed in parallel, and this small burden may not warrant the expense of time and computer resources. Therefore, unless the number of extra function evaluations $n_2 + N - n_3$ exceeds $\frac{1}{2}N$, we decide not to initialize $\boldsymbol{u}_n^{(k)}$, $n = n_3, \ldots, n_2 + N - 1$, but to iterate with only the values $\boldsymbol{u}_n^{(k)}$, $n = n_2, \ldots, n_3 - 1$. Finally, note that in the latter case when $n_2 + N - n_3 \leqslant \frac{1}{2}N$, the number of steps processed is at least $n_3 - n_2 \geqslant \frac{1}{2}N$.

(iv) The number $(\Lambda_{n+1}^{(k+1)})_{ij}$ in (2.1d) has a double definition according to the value $(\boldsymbol{v}_n^{(k+1)} - \boldsymbol{u}_n^{(k)})_j = (\boldsymbol{\tau}_n^{(k)})_j$. If it vanishes, then the partial derivative $\partial(\boldsymbol{F}_{n+1})_i / \partial y_j$ is involved. Evaluating the latter could of course be very laborious, especially when the partial derivatives are complicated to express. Even if $(\tau_n^{(k)})_j \neq 0$, it may happen that it loses all its significant digits. From the computational point of view this case is equivalent to the previous one and we need again the partial derivative $\partial(\boldsymbol{F}_{n+1})_i / \partial y_j$.

In order to avoid the need of the derivatives, we shall still use difference quotients also when $(\boldsymbol{\tau}_n^{(k)})_j \simeq 0$. This will be done by modifying the vector $\boldsymbol{v}_{j,n}^{(k+1)}$ with $\tilde{\boldsymbol{v}}_{j,n}^{(k+1)}$ such that the difference

$(\tilde{v}_{j,n}^{(k+1)} - u_n^{(k)})_j$ has a minimal number of significant digits determined by an assigned minimal relative precision ω.

Finally, we sketch our parallel algorithm. For a given n^*, it produces approximations z_n to the solution of (1.1) for $n \leqslant n^*$ satisfying the local error test (3.4) for a given tolerance TOL. Each stage of the algorithm is labeled by a number followed by S or by P, according whether the stage is carried out sequentially or in parallel with respect to the index n. In any case, the computations are made in parallel with respect to the indices i, j.

Algorithm 3.2. Assign the tolerance TOL and the minimal precision ω. Then set $\mu := \nu^* := \nu := 0$ and $z_0 := y_0$ and proceed as follows:

(1P) if $\nu^* = \mu$ then set $u_{\nu^*} := z_{\nu^*}$
set $\mu^* := \min\{\nu^* + N;\ n^*\}$
for $n = \mu + 1, \ldots, \mu^*$
assign the initial guess u_n

(2P) for $n = \mu + 1, \ldots, \mu^*$
compute $v_n := F_n(u_{n-1})$

(3P) for $n = \mu + 1, \ldots, \mu^*$
compute the local error $\tau_n := v_n - u_n$
and its norm $\| \tau_n \|$
if $\nu^* = \mu$ then set $\nu^* := \nu^* + 1$ and $z_{\nu^*} := v_{\nu^*}$
if $\nu^* = n^*$ then STOP
set $\mu := \mu^*$

(4P) for $n = \nu^*, \ldots, \mu - 1$ and j $= 1, \ldots, m$
set $M_{j,n} := \omega \max\{1;\ |(u_n)_j|; |(v_n)_j|\}$
if $|(\tau_n)_j| < M_{j,n}$ then $v_{j,n} := u_n + M_{j,n} \operatorname{sign}((\tau_n)_j) e_j$
else $v_{j,n} := u_n + (\tau_n)_j e_j$
(here e_j denotes the jth element of the canonical basis of $\mathbb{R}^m$)

(5P) for $n = \nu^*, \ldots, \mu - 1$ and $j = 1, \ldots, m$
compute $w_{j,n+1} := F_{n+1}(v_{j,n})$

(6P) for $n = \nu^*, \ldots, \mu - 1$ and $i, j = 1, \ldots, m$
compute $(\Lambda_{n+1})_{ij} := \dfrac{(w_{j,n+1} - v_{n+1})_i}{(v_{j,n} - u_n)_j}$

(7S) set $\delta_{\nu^*} := \tau_{\nu^*}$
for $n = \nu^*, \ldots, \mu - 1$
compute $\delta_{n+1} := \Lambda_{n+1}\delta_n + \tau_{n+1}$

(8P) set $u_{\nu^*} := z_{\nu^*}$
for $n = \nu^* + 1, \ldots, \mu$
define the new approximation $u_n := u_n + \delta_n$

(9P) for $n = \nu^*, \ldots, \mu$
set $\sigma_n := \| \tau_n \|$

(10P) set $\nu := \nu^*$
for $n = \nu + 1, \ldots, \mu$
compute $v_n := F_n(u_{n-1})$

(11P) for $n = \nu + 1, \ldots, \mu$
compute the local error $\tau_n := v_n - u_n$
and its norm $\| \tau_n \|$

(12S) compute $\nu^* := \min\{\mu;\ n \in [\nu+1,\ \mu] \mid \|\tau_n\| > \mathrm{TOL}\}$
(13P) for $n = \nu+1, \ldots, \nu^*-1$
set $z_n := u_n$
set $z_{\nu^*} := v_{\nu^*}$
if $\nu^* = n^*$ then STOP
if $\nu^* = \mu$ then go to (1P)
(14S) set $\Sigma_\nu := \sigma_\nu$
for $n = \nu+1, \ldots, \mu$
set $\Sigma_n := \max\{\Sigma_{n-1};\ \sigma_n\}$
(15S) compute $\mu^* := \min\{\mu+1;\ n \in [\nu^*+1,\ \mu] \mid \|\tau_n\| > \Sigma_n\} - 1$
set $\mu := \mu^*$
if $\mu - \nu^* \leqslant \frac{1}{2}N$ then go to (1P)
else go to (4P)

4. Some comments and examples

In Algorithm 3.2 a certain segment $[N_k,\ M_k]$ of $[0,\ n^*]$, whose length $L_k := M_k - N_k$ varies between $\frac{1}{2}N$ and N, is handled each iteration. If k^* is the number of iterations necessary to approximate the sequence $\{y_n\}$ over the whole interval $[0,\ n^*]$, then the total number of steps handled by the algorithm is $\sum_{k=1}^{k^*} L_k$. Unless the problem (1.1) is linear, this number is greater than n^*.

In order to give a rough estimate of the speedup factor, we need first an estimate of the time needed to implement both the recursive formula (1.1) and the parallel Algorithm 3.2. Note that the latter includes both some parallel stages and some sequential stages. The only parallel stages really time consuming are (2P), (5P) and (10P), which involve the computation of the functions F_n. As for the sequential stages (7S), (12S), (14S) and (15S) to be implemented each iteration on the segment $[N_k,\ M_k]$, we observe that each of them, owing to its typically linear character (with respect to a certain associative composition rule), can be also implemented by a suitable parallel strategy, passing from a time proportional to L_k to a time proportional to $\log_2(L_k)$. The philosophy of this approach can be found, for example, in [10].

Then we define the *time unit* u as the time needed to implement one step of all the sequential stages, that is a matrix–vector multiplication and a vector addition (stage (7S)), the maximum between two real numbers (stage (14S)) and two comparisons between two real numbers (stages (12S) and (15S)). Moreover, let T be the average number of time units u per step needed for the computation of each function F_n and let PFE be the number of parallel function evaluations necessary to approximate the sequence $\{y_n\}$ over the whole interval $[0,\ n^*]$. Observe that it is always the case that $2k^* + 1 \leqslant \mathrm{PFE} \leqslant 3k^*$, since each iteration requires two parallel function evaluations (stages (5P) and (10P)) and sometimes a third one (stage (2P)), according to the result of the if-statement in (15S). In any case, the first iteration always involves three parallel function evaluations.

Summarizing, the time needed for the sequential computation of (1.1) is n^*Tu. On the contrary, Algorithm 3.2 takes $\mathrm{PFE} \cdot T \cdot u$ for the parallel stages and $(\sum_{k=1}^{k^*} \log_2(L_k))u$ for the sequential stages. Since $\frac{1}{2}N \leqslant L_k \leqslant N$, a reasonable bound to the latter contribution is

$k^* \log_2(N)u$. Therefore we get the estimate

$$\text{speedup} \simeq \frac{n^* T}{k^* \log_2(N) + \text{PFE} \cdot T}. \tag{4.1}$$

Of course, formula (4.1) is merely indicative, since it is based on the naive assumption that computations carried out concurrently take the time of just one of them. However, in any case, it remains true that the larger is T and the lower are k^* and PFE, the higher is the speedup. Before proceeding, we wish to follow up on the claim made in Section 3 that it is possible to estimate the error $y_n - z_n$. Indeed it is sufficient to consider formula (3.3) and to use an approximation of the matrix $D_{\nu+1}^{(k+1)}$. Such an approximation is obtained from the matrix $\Lambda_{\nu+1}^{(k)}$ computed at the last iteration. The cost of this estimate equals the cost of the recursive formula in (7S).

We conclude this paper by giving some numerical results. For the sake of simplicity we limit ourselves to the scalar case, which is completely sufficient to explain the behaviour of the algorithm satisfactorily. In fact the nonscalar case differs only in the number of processors used in the parallel stages. We apply Algorithm 3.2 to approximate the solution of

$$\begin{aligned} y_{n+1} &= F_{n+1}(y_n) \\ &= -\sin\, y_n + \left[y_n \operatorname{arctg}\, y_n - 0.5 \log(1 + y_n^2) - \cos\, y_n \right]/(n+1) + y_n/(n+1)^2, \\ y_0 &:= 2. \end{aligned}$$

The derivatives of F_n are

$$F_n'(y) = -\cos\, y + (\operatorname{arctg}\, y + \sin\, y)/n + 1/n^2$$

and

$$F_n''(y) = \sin\, y + \left[1/(1+y^2) + \cos\, y\right]/n.$$

Hence we have the bounds

$$|F_n'(y)| \leqslant 1 + (1 + \tfrac{1}{2}\pi)/n + 1/n^2$$

and

$$|F_n''(y)| \leqslant 1 + 2/n.$$

Moreover, the solution $\{y_n\}$ vanishes as $n \to \infty$. Therefore constant initial guesses in (1P) seem to be a reasonable choice. More precisely, we assign

$$u_n := u_\mu, \quad n = \mu + 1, \ldots, \mu^*.$$

The experiments were simulated on a CYBER CDC 170/730. On this machine each function F_n takes about 7.6 time units u (i.e. $T = 7.6$).

In Table 1 we give the results in [0, 1000] (i.e. $n^* = 1000$) for different values of the tolerance TOL and of the parallelism N.

First of all, we observe the satisfactory response of the error E_{1000} to the tolerance TOL and also the good estimate of it. Secondly, we observe that, while E_{1000} is almost independent of the parallelism N, this is not the case for the number of iterations k^* (and hence for the number of parallel function evaluations PFE), which is decreasing with N. Indeed it is evident that for fixed accuracy in the initial guesses there should correspond a reduction of k^* proportional to $1/N$ and hence a speedup factor increasing like $N/(\text{const} + \log_2(N))$. On the other hand, one expects

Table 1

TOL	N	k^*	PFE	E_{1000}	Estimated E_{1000}	Speedup	$\frac{1000}{\text{PFE}}$
10^{-3}	50	22	64	1.1E-02	1.3E-02	12.4	15.6
	100	12	34	1.1E-02	1.3E-02	22.5	29.4
	200	7	19	1.0E-02	1.1E-02	38.4	52.6
	400	5	13	8.0E-03	7.6E-03	53.5	76.9
10^{-5}	50	30	81	6.5E-04	9.8E-04	9.7	12.3
	100	18	47	8.3E-04	1.1E-03	15.9	21.3
	200	11	28	5.5E-04	6.5E-04	25.6	35.7
	400	7	17	5.8E-04	6.0E-04	40.1	58.8
10^{-7}	50	43	121	9.0E-07	9.1E-07	6.5	8.3
	100	26	63	1.7E-06	1.7E-06	11.7	15.9
	200	16	38	3.3E-06	3.4E-06	18.5	26.3
	400	10	23	3.1E-06	3.1E-06	29.1	43.5

that the larger is the parallelism N, the worse are the guessed values in the last part of the segment $[N_k, M_k]$, and therefore the lower is the convergence rate for them. In fact we detect the loss of proportionality between k^* and $1/N$, although the speedup remains an increasing function of N.

From (4.1) one sees that the speedup is always increasing with T and that it cannot exceed n^*/PFE. In the example T is not high, although the speedup factors are not far from the bounds n^*/PFE.

References

[1] A. Bellen and M. Zennaro, Algoritmi paralleli per problemi iniziali di equazioni differenziali lineari ordinarie e con ritardo, *Proc. Convegno di Analisi Numerica*, De Frede, Napoli (1986) 43–55.

[2] A. Bellen and M. Zennaro, Parallel algorithms for initial value problems for nonlinear difference and differential equations, Quaderno Matematico N. 140, Dipartimento di Scienze Matematiche, University of Trieste, 1987.

[3] M.N. El-Tarazi, Iterative methods for systems of first order differential equations, *IMA J. Numer. Anal.* **5** (1985) 29–39.

[4] C.W. Gear, The potential for parallelism in ordinary differential equations, Report No. UIUCDCS-R-86-1246, Dept. of Computer Science, University of Illinois, 1986.

[5] C.W. Gear, Parallel methods for ordinary differential equations, *Calcolo* **XXV(1)** (1988), in press.

[6] O.A. Karakashian, On Runge–Kutta methods for parabolic problems with time dependent coefficients, *Math. Comp.* **47** (1986) 77–106.

[7] I. Lie, Some aspects of parallel Runge–Kutta methods, Mathematics and Computation 3/87, Dept. of Numerical Mathematics, Univ. of Trondheim, 1987.

[8] J. Nievergelt, Parallel methods for integrating ordinary differential equations, *Comm. ACM* **7** (1964) 731–733.

[9] J.M. Ortega and W.C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables* (Academic Press, New York, 1970).

[10] U. Schendel, *Introduction to Numerical Methods for Parallel Computers* (Wiley, New York, 1984).

[11] V.A. Vinokurov, A method for the numerical solution of linear differential equations, *U.S.S.R. Comput. Math. Math. Phys.* **22** (1982) 58–73.