ARTICLE IN PRESS APNUM:2119

Available online at www.sciencedirect.com

ScienceDirect

Applied Numerical Mathematics ••• (••••) •••–•••

APPLIED NUMERICAL MATHEMATICS

www.elsevier.com/locate/apnum

Parallel solution in time of ODEs: some achievements and perspectives [☆]

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

[a] Dipartimento di Matematica, Via Orabona 4, 70125 Bari, Italy
[b] Dipartimento di Matematica "U. Dini", Viale Morgagni 67/A, 50134 Firenze, Italy

Abstract

The parallel solution of initial value problems for ordinary differential equations (ODE-IVPs) has received much interest from many researchers in the past years. In general, the possibility of using parallel computing in this setting concerns different aspects of the numerical solution of ODEs, depending on the parallel platform to be used and/or the complexity of the problem to be solved. In particular, in this paper we examine possible extensions of a parallel method previously proposed in the mid-nineties [P. Amodio, L. Brugnano, Parallel implementation of block boundary value methods for ODEs, J. Comput. Appl. Math. 78 (1997) 197–211; P. Amodio, L. Brugnano, Parallel ODE solvers based on block BVMs, Adv. Comput. Math. 7 (1997) 5–26], and analyze its connections with subsequent approaches to the parallel solution of ODE-IVPs, in particular the "Parareal" algorithm proposed in [J.L. Lions, Y. Maday, G. Turinici, Résolution d'EDP par un schéma en temps "pararéel", C. R. Acad. Sci. Paris, Ser. I 332 (2001) 661–668; Y. Maday, G. Turinici, A parareal in time procedure for the control of partial differential equations, C. R. Acad. Sci. Paris, Ser. I 335 (2002) 387–392].
© 2008 IMACS. Published by Elsevier B.V. All rights reserved.

PACS: 65L05; 65L06; 65L80; 65H10

Keywords: Ordinary Differential Equations; Initial Value Problems; Stiff Problems; Parallel Computing; Parallel methods "in time" for ODEs; Boundary Value Methods (BVMs); Block one step methods; Parallel factorizations; "Parareal" algorithm

1. Introduction

The parallel solution of ODE-IVPs has received interest from many researchers, starting in the sixties, leading to many approaches able to exploit, at different levels, the possibility of using parallel computing platforms. In order to have a good account of what has going on about this subject up to the mid-nineties, good references are the monograph [13] and the special issue [14], both edited by K. Burrage. Even though the various approaches to the problem may differ substantially each other, all the parallel methods devised so far can be roughly collected in three main categories (see, for example, the introduction of [14]):

[☆] Research supported by the Italian M.I.U.R.
[*] Corresponding author.
E-mail addresses: amodio@dm.uniba.it (P. Amodio), brugnano@math.unifi.it (L. Brugnano).

0168-9274/$30.00 © 2008 IMACS. Published by Elsevier B.V. All rights reserved.
doi:10.1016/j.apnum.2008.03.024

- *Parallelism across the method:* in which the computation required to perform a single integration step of a given numerical method is split (in some way) among more parallel computing units;
- *Parallelism across the system:* in which the parallelism is exploited at the level of the problem to be solved, e.g. by defining suitable splittings of the continuous problem and corresponding Picard's type iterations;
- *Parallelism across the steps:* in which several integration steps are performed concurrently with a given numerical method.

In this paper we shall be concerned with parallel methods in the third category which are, therefore, aimed to perform several integration steps at the same time. In particular, we shall consider the approach which has been proposed in [3,4] (see also [10,12,11,5]). This approach was at first introduced with particular reference to *block Boundary Value Methods (BVMs)* [12] but it is clearly generalizable to any block method for ODEs. The basic idea on which this method relies is the definition of a *coarse mesh*, defined by a suitable partition of the integration interval, on which the problem can be solved in parallel, provided that a global information, consisting in the solution of a corresponding *reduced system*, is obtained. In Section 2 we provide a description of this approach, by emphasizing that it is not related to any specific integration procedure. We shall here consider the application of the method to the linear problem

$$y' = Ly + g(t), \quad t \in [t_0, T], \qquad y(t_0) = y_0 \in \mathbb{R}^m, \tag{1}$$

which is sufficient to grasp the main features of this approach.

More recently, much attention has been devoted to another method, exploiting parallelism across the steps, which has been named "Parareal" algorithm [19,20]. In particular, this approach has become quite popular among people involved in domain decomposition methods (see, e.g., [9,16,17,21,24,25]). It turns out that this method is deeply related to the previous approach and, therefore, in Section 3 we show the existing connections between the two methods.

The parallel algorithm described in Section 2 has been originally implemented by using a direct factorization of the resulting discrete problem. This, however, may be impractical when the dimension, m, of problem (1) is very large, with L a sparse matrix. In Section 4 we show how it is possible, in such a case, to modify the original approach, in order to take account of this feature. Finally, in Section 5 we report some numerical tests to assess the potentialities of the proposed extension, along with a few concluding remarks.

2. The parallel method

Let us consider a suitable *coarse mesh*, defined by the following partition of the integration interval in (1):

$$t_0 \equiv \tau_0 < \tau_1 < \cdots < \tau_p \equiv T. \tag{2}$$

Suppose, for simplicity, that inside each subinterval we apply a given method with constant stepsize

$$h_i = \frac{\tau_i - \tau_{i-1}}{N}, \quad i = 1, \ldots, p, \tag{3}$$

to approximate the problem

$$y' = Ly + g(t), \quad t \in [\tau_{i-1}, \tau_i], \qquad y(\tau_{i-1}) = y_{0i}, \quad i = 1, \ldots, p. \tag{4}$$

If $y(t)$ denotes the solution of problem (1), and we call

$$y_{ni} \approx y(\tau_{i-1} + nh_i), \quad n = 0, \ldots, N, \ i = 1, \ldots, p, \tag{5}$$

the entries of the discrete approximation, then, in order the numerical solutions of (1) and (4) to be equivalent, we require that (see (2) and (5))

$$y_{01} = y_0, \qquad y_{0i} \equiv y_{N,i-1}, \quad i = 2, \ldots, p. \tag{6}$$

For convention, we also set

$$y_{01} \equiv y_{N0}. \tag{7}$$

Let now suppose that the numerical approximations to the solutions of (4) are obtained by solving discrete problems in the form

$$M_i\mathbf{y}_i = \mathbf{v}_i y_{0i} + \mathbf{g}_i, \quad \mathbf{y}_i = (y_{1i}, \ldots, y_{Ni})^T,\ i = 1, \ldots, p, \tag{8}$$

where the matrices $M_i \in \mathbb{R}^{mN\times mN}$ and $\mathbf{v}_i \in \mathbb{R}^{mN\times m}$, and the vector $\mathbf{g}_i \in \mathbb{R}^{mN}$, do depend on the chosen method (see, e.g., [3,4], for the case of block BVMs) and on the problems (4). Clearly, this is a quite general framework, which encompasses most of the currently available methods for solving ODE-IVPs. By taking into account all the above facts, one obtains that the global approximation to the solution of (1) is obtained by solving a discrete problem in the form (hereafter, I_r will denote the identity matrix of dimension r):

$$\begin{aligned} M\mathbf{y} &\equiv \begin{pmatrix} I_m &&&& \\ -\mathbf{v}_1 & M_1 &&& \\ & -V_2 & M_2 && \\ && \ddots & \ddots & \\ &&& -V_p & M_p \end{pmatrix} \begin{pmatrix} y_{N0} \\ \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_p \end{pmatrix} = \begin{pmatrix} y_0 \\ \mathbf{g}_1 \\ \mathbf{g}_2 \\ \vdots \\ \mathbf{g}_p \end{pmatrix}, \\ V_i &= [O \mid \mathbf{v}_i] \in \mathbb{R}^{mN\times mN}, \quad i = 2, \ldots, p. \end{aligned} \tag{9}$$

Obviously, this problem may be solved in a sequential fashion, by means of the iteration (see (6)–(7)):

$$y_{N0} = y_0, \qquad M_i\mathbf{y}_i = \mathbf{g}_i + \mathbf{v}_i y_{N,i-1}, \quad i = 1, \ldots, p.$$

Nevertheless, in [3,4] it has been suggested to use a suitable *parallel factorization* of the matrix in (9), in order to derive a corresponding parallel algorithm. By the way, we mention that the concept of "parallel factorization" has been first introduced in [1] (see also [8,2]) for tridiagonal matrices, but it can be extended straightforwardly to the case of block tridiagonal (and then, lower block bidiagonal) matrices, as well as to more general settings (see, e.g., [6,7]). In particular, we consider the factorization:

$$M = DW \equiv \begin{pmatrix} I_m &&&& \\ & M_1 &&& \\ && M_2 && \\ &&& \ddots & \\ &&&& M_p \end{pmatrix} \begin{pmatrix} I_m &&&& \\ -\mathbf{w}_1 & I_{mN} &&& \\ & -W_2 & I_{mN} && \\ && \ddots & \ddots & \\ &&& -W_p & I_{mN} \end{pmatrix},$$

where (see (9))

$$W_i = [O \mid \mathbf{w}_i] \in \mathbb{R}^{mN\times mN}, \quad \mathbf{w}_i = M_i^{-1}\mathbf{v}_i \in \mathbb{R}^{mN\times m}. \tag{10}$$

Consequently, at first we solve, in parallel, the systems

$$M_i\mathbf{z}_i = \mathbf{g}_i, \quad \mathbf{z}_i = (z_{1i}, \ldots, z_{Ni})^T, \quad i = 1, \ldots, p, \tag{11}$$

and, then, (see (10) and (6)) recursively update the local solutions,

$$\begin{aligned} \mathbf{y}_1 &= \mathbf{z}_1 + \mathbf{w}_1 y_{01}, \\ \mathbf{y}_i &= \mathbf{z}_i + W_i\mathbf{y}_{i-1} \equiv \mathbf{z}_i + \mathbf{w}_i y_{0i}, \quad i = 2, \ldots, p. \end{aligned} \tag{12}$$

The latter recursion, however, has still much parallelism. Indeed, if we consider the partitionings (see (8), (11), and (10))

$$\mathbf{y}_i = \begin{pmatrix} \hat{\mathbf{y}}_i \\ y_{Ni} \end{pmatrix}, \qquad \mathbf{z}_i = \begin{pmatrix} \hat{\mathbf{z}}_i \\ z_{Ni} \end{pmatrix}, \qquad \mathbf{w}_i = \begin{pmatrix} \hat{\mathbf{w}}_i \\ w_{Ni} \end{pmatrix}, \quad w_{Ni} \in \mathbb{R}^{m\times m}, \tag{13}$$

then (12) is equivalent to solve, at first, the *reduced system*

$$\begin{pmatrix} I_m &&& \\ -w_{N1} & I_m && \\ & \ddots & \ddots & \\ && -w_{N,p-1} & I_m \end{pmatrix} \begin{pmatrix} y_{01} \\ y_{02} \\ \vdots \\ y_{0p} \end{pmatrix} = \begin{pmatrix} y_0 \\ z_{N1} \\ \vdots \\ z_{N,p-1} \end{pmatrix}, \tag{14}$$

i.e.,

$$y_{01} = y_0, \qquad y_{0,i+1} = z_{Ni} + w_{Ni} y_{0i}, \quad i = 1, \dots, p-1, \tag{15}$$

after which performing the p parallel updates

$$\hat{\mathbf{y}}_i = \hat{\mathbf{z}}_i + \hat{\mathbf{w}}_i y_{0i}, \quad i = 1, \dots, p-1, \qquad \mathbf{y}_p = \mathbf{z}_p + \mathbf{w}_p y_{0p}. \tag{16}$$

We observe that:

- the parallel solution of the p systems in (11) is equivalent to compute the approximate solution of the following p ODE-IVPs,
$$z' = Lz + g(t), \quad t \in [\tau_{i-1}, \tau_i], \qquad z(\tau_{i-1}) = 0, \quad i = 1, \dots, p, \tag{17}$$
in place of the corresponding ones in (4);
- the solution of the reduced system (14)–(15) consists in computing the proper initial values $\{y_{0i}\}$ for the previous ODE-IVPs;
- the parallel updates (16) update the approximate solutions of the ODE-IVPs (17) to those of the corresponding ODE-IVPs in (4).

Remark 1. Clearly, the solution of the first (parallel) system in (11) and the first (parallel) update in (12) (see also (16)) can be executed together, by solving the linear system (see (6))

$$M_1 \mathbf{y}_1 = \mathbf{g}_1 + \mathbf{v}_1 y_0, \tag{18}$$

thus directly providing the final discrete approximation on the first processor; indeed, this is possible, since the initial condition y_0 is given.

We end this section by emphasizing that one obtains an almost perfect parallel speed-up, if p processors are used, provided that the cost for the solution of the reduced system (14) and of the parallel updates (16) is small, with respect to that of (11) (see [3,4] for more details). This is, indeed, the case when the parameter N in (3) is large enough and the coarse partition (2) can be supposed to be *a priori* given.

3. Connections with the "Parareal" algorithm

We now briefly describe the "Parareal" algorithm introduced in [19,20] (see also [17,21,24]), showing the existing connections with the parallel method previously described. This method, originally defined for solving PDE problems, for example linear or quasi-linear parabolic problems, can be directly cast into the ODE setting via the semi-discretization of the space variables; that is, by using the method of lines. In more detail, let consider the problem

$$\frac{\partial}{\partial t} y = \mathcal{L} y, \quad t \in [t_0, T], \qquad y(t_0) = y_0, \tag{19}$$

where $\mathcal{L}$ is an operator from a Hilbert space V into V'. Let us consider again the partition (2) of the time interval, and consider the problems

$$\frac{\partial}{\partial t} y = \mathcal{L} y, \quad t \in [\tau_{i-1}, \tau_i], \qquad y(\tau_{i-1}) = y_{0i}, \quad i = 1, \dots, p. \tag{20}$$

Clearly, in order (19) and (20) to be equivalent, one must require that

$$y_{0i} = y(\tau_{i-1}), \quad i = 1, \dots, p. \tag{21}$$

The initial data (21) are then formally related by means of suitable *propagators* $\mathcal{F}_i$ such that

$$y_{0,i+1} = \mathcal{F}_i y_{0i}, \qquad i = 1, \dots, p-1. \tag{22}$$

The previous relations can be cast in matrix form as ($\mathcal{I}$ is now the identity operator)

$$F\mathbf{y} \equiv \begin{pmatrix} \mathcal{I} & & & \\ -\mathcal{F}_1 & \mathcal{I} & & \\ & \ddots & \ddots & \\ & & -\mathcal{F}_{p-1} & \mathcal{I} \end{pmatrix} \begin{pmatrix} y_{01} \\ y_{02} \\ \vdots \\ y_{0p} \end{pmatrix} = \begin{pmatrix} y_0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \equiv \boldsymbol{\eta}. \tag{23}$$

For solving (23), the authors essentially define the splitting

$$F = (F - G) + G, \quad G = \begin{pmatrix} \mathcal{I} & & & \\ -\mathcal{G}_1 & \mathcal{I} & & \\ & \ddots & \ddots & \\ & & -\mathcal{G}_{p-1} & \mathcal{I} \end{pmatrix},$$

with *coarse propagators*

$$\mathcal{G}_i \approx \mathcal{F}_i, \quad i = 1, \dots, p,$$

and consider the iterative procedure

$$G\mathbf{y}^{(k+1)} = (G - F)\mathbf{y}^{(k)} + \boldsymbol{\eta}, \quad k = 0, 1, \dots,$$

with an obvious meaning of the upper index. This is equivalent to solve the problems

$$\begin{aligned} y_{01}^{(k+1)} &= y_0, \\ y_{0,i+1}^{(k+1)} &= \mathcal{G}_i y_{0i}^{(k+1)} + (\mathcal{F}_i - \mathcal{G}_i) y_{0i}^{(k)}, \quad i = 1, \dots, p-1, \end{aligned} \tag{24}$$

thus providing good parallel features, if we can assume that the coarse operators $\mathcal{G}_i$ are "cheap" enough. The iteration (24) defines the "Parareal" algorithm, which is iterated until

$$\left\| y_{0i}^{(k+1)} - y_{0i}^{(k)} \right\|, \quad i = 2, \dots, p,$$

are suitably small. In the practice, in case of linear operators, problem (19) becomes, via the method of lines, an ODE in the form (1), with L a huge and very sparse matrix. Similarly, problems (20) become in the form (4). Similarly, the propagator $\mathcal{F}_i$ consists in the application of a suitable discrete method for approximating the solution of the corresponding ith problem in (4), and the coarse propagator $\mathcal{G}_i$ describes the application of a much cheaper method for solving the same problem. As a consequence, if the discrete problems corresponding to the propagators $\{\mathcal{F}_i\}$ are in the form (8), then the discrete version of the recurrence (22) becomes exactly (15), as well as the discrete counterpart of the matrix form (23) becomes (14).

We can then conclude that the "Parareal" algorithm in [19,20] *exactly* coincides with the iterative solution of the reduced system (14), induced by a suitable splitting. For a corresponding convergence analysis, we refer to [15,16]. We observe that the previous iterative procedure may be very appropriate, when the matrix L is large and sparse since, in this case, the computations of the block vectors $\{\mathbf{w}_i\}$ in (10), and then of the matrices $\{w_{Ni}\}$ (see (13)) would be clearly impractical. In the next section, we propose a modification of the approach described in Section 2, able to overcome this drawback.

4. Extensions

In this section, we consider an extension of the parallel algorithm described in Section 2, which is able to cope with the case in which the matrix L in (1) is very large and sparse, such as for problems deriving from the application of the method of lines for solving PDEs. Problem (1) indeed results from the discretization of linear parabolic PDEs, which we shall here consider, even though the whole approach can be straightforwardly extended to the case of quasi-linear problems. Our purpose is then that of reformulating the parallel algorithm, by avoiding the factorization of the involved matrices. Let us analyze the basic steps. Whatever the numerical ODE-methods used, at first we have to solve, in parallel, the p linear systems (11) of dimension mN. This can be done by using a suitable (possibly preconditioned) iterative solver, without requiring any factorization. After this has been done (with an almost perfect parallel speed-up, provided that the coarse mesh (2) has been suitably chosen), we have to solve the reduced system (14)–(15).

Nevertheless, this would require the computation of the matrices w_{Ni}, $i=1,\dots,p$, in (13) which, in turn, would require the solution of the linear systems (see (10))

$$M_i\mathbf{w}_i=\mathbf{v}_i, \quad i=1,\dots,p. \tag{25}$$

However, this computation would be too costly, since the right-hand sides have m columns, with m very large. Nevertheless, we observe that the last block entry of $\mathbf{w}_i$, i.e. w_{Ni}, is nothing but a discrete approximation to $\mathrm{e}^{(\tau_i-\tau_{i-1})L}$. Therefore, it suffices to get an accurate enough approximation, say φ_i, of

$$w_{Ni}y_{0i}\approx \mathrm{e}^{(\tau_i-\tau_{i-1})L}y_{0i},$$

in order to be able to compute (approximately) the recurrence (15). In more detail, we have:

$$y_{01}=y_0, \qquad y_{0,i+1}=z_{Ni}+\varphi_i, \quad i=1,\dots,p-1, \tag{26}$$

where

$$\varphi_i\approx \mathrm{e}^{(\tau_i-\tau_{i-1})L}y_{0i}, \quad i=1,\dots,p-1. \tag{27}$$

Once the $\{y_{0i}\}$ have been computed, we can solve, in place of the parallel updates (16) (which would again require the vectors $\{\mathbf{w}_i\}$ in (25)), the linear systems

$$M_i\mathbf{y}_i=\mathbf{g}_i+\mathbf{v}_i y_{0i}, \quad i=1,\dots,p. \tag{28}$$

Evidently, they can be solved in parallel with the same iterative solver used for (11). Moreover, this can be done by using a good initial guess, due to the knowledge of the (approximated) $\{y_{0i}\}$. We observe that the solution of (28) is nothing but the approximate solution of (4). The problem is, therefore, that of computing the approximations (27). This will be done by means of a low-rank Krylov approximation of $\mathrm{e}^{(\tau_i-\tau_{i-1})L}$, able to provide a suitably accurate φ_i in (27). The approach that we shall consider is similar to what suggested, for example, in [23,18], even though different approaches can be also considered (see, e.g., [22]). Here are the details. For sake of simplicity, let us skip the indexes, thus computing the generic approximation, for given $y\in\mathbb{R}^m$ and $\Delta\tau>0$,

$$\varphi\approx \mathrm{e}^{\Delta\tau L}y. \tag{29}$$

Let us then initialize (hereafter, $\|\cdot\|$ will denote the Euclidean norm):

$$u_1=\frac{y}{\|y\|}, \tag{30}$$

and compute the following Arnoldi iteration,

$$\begin{aligned}
&v_j=Lu_j,\\
&h_{ij}=u_i^Tv_j, \quad v_j\leftarrow v_j-h_{ij}u_i, \quad i=1,\dots,j,\\
&h_{j+1,j}=\|v_j\|,\\
&u_{j+1}=v_j/h_{j+1,j}, \quad j=1,\dots,k,
\end{aligned} \tag{31}$$

until a suitable index $k\ll m$. The choice of k (see also [18]) will be discussed later. Let then be

$$U_k=(u_1,\dots,u_k)\in\mathbb{R}^{m\times k}, \qquad H_k=(h_{ij})\in\mathbb{R}^{k\times k}. \tag{32}$$

Clearly, the columns of U_k are orthonormal vectors and H_k is an upper Hessemberg matrix (symmetric tridiagonal, if L is symmetric). We then consider the approximation (see (30))

$$\varphi\equiv\varphi^{(k)}=U_k\mathrm{e}^{\Delta\tau H_k}U_k^Ty=U_k\mathrm{e}^{\Delta\tau H_k}e_1\|y\|, \tag{33}$$

where, as usual, e_1 is the first unit vector in $\mathbb{R}^k$. This, in turn, requires to store the k vectors u_j and to compute the matrix exponential of a $k\times k$ matrix (actually, only the first column of the matrix exponential is needed). Nevertheless, since we expect k to be very small, this is not a problem at all.

Remark 2. We observe that, when L is not symmetric, a Lanczos procedure could be used in place of the Arnoldi process (31) (see, e.g., [23]), thus always resulting in a tridiagonal H_k. In this case, however, possible breakdowns of the procedure, requiring a corresponding "look-ahead" recovery, could occur.

We end this section by recalling that a dynamic error estimate in the approximation (29)–(33) can be obtained, as described in [23], with a very small extra cost, with respect to that for computing (33), at least when the spectral radius of $\Delta\tau L$ is suitably small (see [23, Sections 2, 4, and 5]). In more detail, after the Arnoldi iteration (31), we know the matrices (see (32))

$$U_{k+1} = (U_k\ u_{k+1}) \in \mathbb{R}^{m\times k+1}, \qquad \begin{pmatrix} H_k \\ \hline h_{k+1,k}(e_k^T) \end{pmatrix} \in \mathbb{R}^{k+1\times k},$$

with $e_k \in \mathbb{R}^k$ the kth unit vector. We observe that the vector u_{k+1}, as well as $h_{k+1,k}$, are not actually involved in the approximation (33). However, by setting

$$\hat{H}_k = \left(\begin{array}{c|c} H_k & 0 \\ \hline h_{k+1,k}(e_k^T) & 0 \end{array}\right) \in \mathbb{R}^{k+1\times k+1},$$

it can be proved that (see [23, Section 2.3])

$$e^{\Delta\tau \hat{H}_k} = \left(\begin{array}{c|c} e^{\Delta\tau H_k} & 0 \\ \hline h_{k+1,k}(e_k^T \psi(\Delta\tau H_k)) & 1 \end{array}\right), \qquad \psi(z) = \frac{e^z - 1}{z},$$

and moreover, by setting $\hat{e}_1 \in \mathbb{R}^{k+1}$ the first unit vector,

$$\begin{aligned} \hat{\varphi}^{(k)} &= U_{k+1} e^{\Delta\tau \hat{H}_k} \hat{e}_1 \|y\| \\ &= U_k e^{\Delta\tau H_k} e_1 \|y\| + u_{k+1}\big(e_k^T \psi(\Delta\tau H_k) e_1\big)\big(h_{k+1,k}\|y\|\big) \\ &\equiv \varphi^{(k)} + u_{k+1}\big(e_k^T \psi(\Delta\tau H_k) e_1\big)\big(h_{k+1,k}\|y\|\big), \end{aligned}$$

is an approximation to $e^{\Delta\tau L} y$ which is more accurate than $\varphi^{(k)}$, when the spectral radius of $\Delta\tau L$ is suitably small. Consequently, the estimate

$$\|\varphi^{(k)} - e^{\Delta\tau L} y\| \approx \|\varphi^{(k)} - \hat{\varphi}^{(k)}\|$$

can be conveniently used, since $\hat{\varphi}^{(k)}$ can be computed with a very little overhead, with respect to that for computing $\varphi^{(k)}$. As a matter of fact, one actually computes, at first, the vector

$$\hat{p}_k = e^{\Delta\tau \hat{H}_k} \hat{e}_1 \|y\| \equiv \begin{pmatrix} p_k \\ \alpha_k \end{pmatrix} \in \mathbb{R}^{k+1}, \quad p_k \in \mathbb{R}^k.$$

Then, one obtains that

$$\varphi^{(k)} = U_k p_k, \qquad \big\|\varphi^{(k)} - \hat{\varphi}^{(k)}\big\| \equiv |\alpha_k|. \tag{34}$$

If only a rough estimate is needed, one can then compute directly

$$\varphi^{(k)} = U_k e^{\Delta\tau H_k} e_1 \|y\| \equiv U_k p_k,$$

and then use the approximation (see [23, Section 5.2])

$$\big\|\varphi^{(k)} - \hat{\varphi}^{(k)}\big\| \approx h_{k+1,k}\big|e_k^T p_k\big|. \tag{35}$$

When the spectral radius $\Delta\tau L$ is not sufficiently small, then the estimate

$$\big\|\varphi^{(k)} - e^{\Delta\tau L} y\big\| \approx \big\|\varphi^{(k)} - \varphi^{(k+1)}\big\| \equiv \Delta\varphi^{(k)} \tag{36}$$

turns out to be more appropriate, even though it is slightly more expensive than (34) and (35).

5. Numerical results and concluding remarks

In this section we consider some numerical tests, obtained by using the approach proposed in Section 4, and provide some comments and concluding remarks. We consider the heat equation,

$$\frac{\partial}{\partial t} y(x_1, x_2, t) = \Delta y(x_1, x_2, t), \quad (x_1, x_2, t) \in (0,4)^2 \times (0, 6\pi],$$

with initial and boundary conditions respectively given by:

$$\begin{aligned}
&y(x_1, x_2, 0) = \cos\frac{\pi}{4}x_1 \cos\frac{\pi}{4}x_2,\\
&y(0, x_2, t) = -y(4, x_2, t) = \cos\frac{\pi}{4}x_2 \cos t,\\
&y(x_1, 0, t) = -y(x_1, 4, t) = \cos\frac{\pi}{4}x_1 \cos t,\\
&\quad x_1, x_2 \in [0, 4],\ t \in [0, 6\pi].
\end{aligned}$$

Second order semi-discretization of the space variables with stepsizes

$$h_{x_1} = h_{x_2} = \frac{4}{\nu + 1},$$

then leads to a differential problem in the form (1) with $t_0 = 0$ and $T = 6\pi$,

$$L = C_\nu \otimes I_\nu + I_\nu \otimes C_\nu, \quad C_\nu = (\nu + 1)^2 \begin{pmatrix} -2 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ 5 & & 1 & -2 \end{pmatrix} \in \mathbb{R}^{\nu\times\nu},$$

and $g(t) \in \mathbb{R}^{\nu^2}$ suitably defined. The dimension of the problem is, therefore, $m = \nu^2$. For simplicity, we consider the following uniform partition, which defines the coarse mesh (2),

$$\tau_i = \frac{6\pi}{p} i, \quad i = 0, \dots, p.$$

Then, in each subinterval $[\tau_{i-1}, \tau_i]$ we consider discrete problems (8) defined by the application of one initial step of the implicit Euler method, with subsequent $N - 1$ steps of the second order BDF, with stepsize h_i as defined in (3). One then obtains (8) with

$$M_i = A_N \otimes I_m - h_i I_N \otimes L, \qquad \mathbf{v}_i = v_N \otimes I_m, \tag{37}$$

where

$$A_N = \frac{1}{2}\begin{pmatrix} 2 & & & & \\ -4 & 3 & & & \\ 1 & -4 & 3 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -4 & 3 \end{pmatrix} \in \mathbb{R}^{N\times N}, \qquad v_N = \frac{1}{2}\begin{pmatrix} 2 \\ -1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \in \mathbb{R}^N.$$

Finally, at the right-hand side in (8),

$$\mathbf{g}_i = h_i (g_{1i}, \dots, g_{Ni})^T, \quad g_{ni} = g(\tau_{i-1} + n h_i), \tag{38}$$

where the same ordering in (5) has been used. The corresponding linear systems (11) are solved by using the conjugate gradient method with diagonal preconditioning. In doing this, we obviously exploit the lower block triangular structure of the matrices (37). The stopping criterion used for this method, on the generic ith processor, is (see (38)),

$$\|r_{ni}^{(\ell)}\| \leqslant \mathrm{tol}\big(h_i \|g_{ni}\|\big), \quad n = 1, \dots, N, \tag{39}$$

where $r_{ni}^{(\ell)}$ is the residual vector at the ℓth iteration, and tol is a prescribed tolerance. Moreover, with the only exception of the first processor (see Remark 1), the initial approximations

$$z_{1i}^{(0)} = 0, \qquad z_{n+1,i}^{(0)} = z_{ni}^{(\ell_{ni})}, \quad n = 1, \dots, N - 1, \tag{40}$$

are considered for the iterative solver, where ℓ_{ni} is the number of iterations needed for convergence, when computing z_{ni}. The approximation (33) and the error estimate (36) have been used for solving the reduced system (26), with stopping criterion, on processor i,

$$\Delta\varphi_i^{(k)} \leqslant \min\big\{\mathrm{tol}\big\|z_{Ni} + \varphi_i^{(k)}\big\|, \sqrt{\mathrm{tol}}\,\big\|\varphi_i^{(k)}\big\|\big\}. \tag{41}$$

Table 1
Execution statistics, $p = 4$ and $\nu = 50$

N	$\ell^{(1)}_{\min}$–$\ell^{(1)}_{\max}$	$K_{\min}$–$K_{\max}$	K_{tot}	$\ell^{(2)}_{\min}$–$\ell^{(2)}_{\max}$	ℓ_{seq}	max err	s_p
100	2481–2510	24–34	58	2466–2475	9929	8.4×10^{-5}	2.0
200	3311–3343	24–34	58	3286–3327	13 270	4.6×10^{-5}	2.0
400	4055–4146	24–34	58	4030–4125	16 343	5.6×10^{-5}	2.0

Table 2
Execution statistics, $p = 4$ and $\nu = 100$

N	$\ell^{(1)}_{\min}$–$\ell^{(1)}_{\max}$	$K_{\min}$–$K_{\max}$	K_{tot}	$\ell^{(2)}_{\min}$–$\ell^{(2)}_{\max}$	ℓ_{seq}	max err	s_p
100	4627–4711	42–63	105	4619–4630	18 571	1.3×10^{-4}	2.0
200	6408–6474	42–63	105	6369–6399	25 620	1.1×10^{-4}	2.0
400	8319–8394	42–63	105	8265–8358	33 318	6.0×10^{-5}	2.0

Table 3
Execution statistics, $p = 8$ and $\nu = 50$

N	$\ell^{(1)}_{\min}$–$\ell^{(1)}_{\max}$	$K_{\min}$–$K_{\max}$	K_{tot}	$\ell^{(2)}_{\min}$–$\ell^{(2)}_{\max}$	ℓ_{seq}	max err	s_p
50	1159–1345	23–34	179	1153–1316	9929	8.4×10^{-5}	3.5
100	1542–1818	23–33	177	1522–1778	13 270	4.6×10^{-5}	3.5
200	1875–2266	23–33	177	1841–2218	16 343	5.6×10^{-5}	3.5

Table 4
Execution statistics, $p = 8$ and $\nu = 100$

N	$\ell^{(1)}_{\min}$–$\ell^{(1)}_{\max}$	$K_{\min}$–$K_{\max}$	K_{tot}	$\ell^{(2)}_{\min}$–$\ell^{(2)}_{\max}$	ℓ_{seq}	max err	s_p
50	2137–2539	39–63	326	2129–2481	18 571	1.3×10^{-4}	3.5
100	2948–3519	39–63	326	2939–3454	25 620	1.1×10^{-4}	3.5
200	3842–4600	39–63	326	3790–4518	33 318	6.0×10^{-5}	3.5

Table 5
Execution statistics, $p = 16$ and $\nu = 50$

N	$\ell^{(1)}_{\min}$–$\ell^{(1)}_{\max}$	$K_{\min}$–$K_{\max}$	K_{tot}	$\ell^{(2)}_{\min}$–$\ell^{(2)}_{\max}$	ℓ_{seq}	max err	s_p
25	492–762	23–39	451	479–752	9929	1.2×10^{-4}	5.1
50	644–1065	23–39	451	618–1046	13 270	6.7×10^{-5}	5.2
100	774–1375	23–39	451	724–1332	16 343	9.2×10^{-5}	5.2

At last, the linear systems (28) are solved by the same preconditioned iterative solver and similar stopping criterion as above, with initial guesses obtained by the knowledge of the approximated local initial values $\{y_{0i}\}$. In Tables 1–6 we list the obtained results, for $\nu = 50, 100$ and when the tolerance $\text{tol} = 10^{-5}$ is used in each subinterval, in the cases $p = 4, 8, 16$, respectively. Clearly, the product pN is kept constant on the ith row of each table, $i = 1, 2, 3$. All executions are performed in Matlab, thus only simulating a parallel computing platform with p processors. The reference solution is obtained by solving sequentially the linear systems (8), by means of the same iterative solver and initial approximations as described above. The parameters listed in the tables are:

Table 6
Execution statistics, $p = 16$ and $\nu = 100$

N	$\ell_{\min}^{(1)}$–$\ell_{\max}^{(1)}$	$K_{\min}$–$K_{\max}$	K_{tot}	$\ell_{\min}^{(2)}$–$\ell_{\max}^{(2)}$	ℓ_{seq}	max err	s_p
25	898–1461	38–66	813	870–1446	18 571	1.4×10^{-4}	5.0
50	1217–2062	38–66	813	1173–2042	25 620	1.1×10^{-4}	5.2
100	1550–2759	38–66	813	1481–2712	33 318	8.8×10^{-5}	5.3

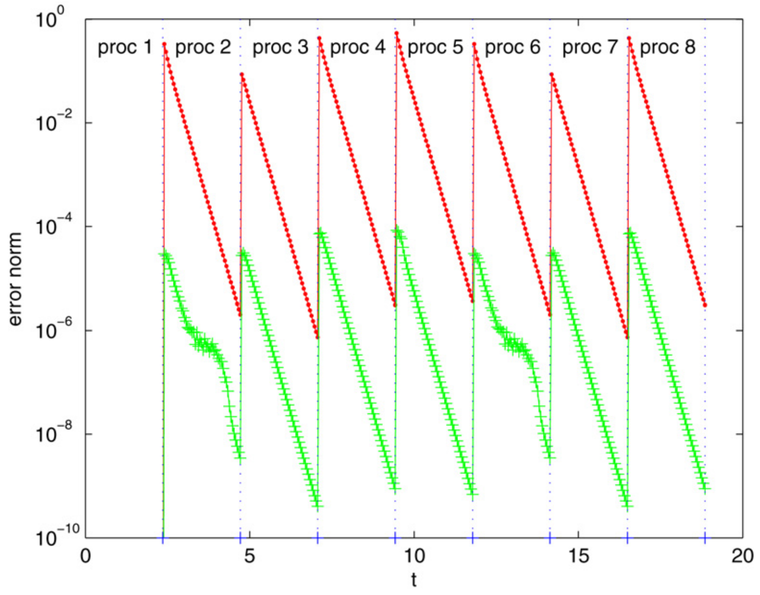

Fig. 1. Global error in the local solutions after the first parallel step (dot-line) and after the parallel updates (plus-line), $\nu = 100$, $N = 50$, $p = 8$.

- N, i.e. the block size of the local problems;
- $\ell_{\min}^{(1)}$–$\ell_{\max}^{(1)}$, i.e. the minimum and maximum number of conjugate gradient iterations required to satisfy (39), when solving the problems (11). In more detail (see (40)),
$$\ell_{\min}^{(1)} = \min_{i=1,\dots,p} \sum_{n=1}^{N} \ell_{ni}, \qquad \ell_{\max}^{(1)} = \max_{i=1,\dots,p} \sum_{n=1}^{N} \ell_{ni};$$
- $K_{\min}$–$K_{\max}$, i.e. the minimum and maximum dimensions of the matrices H_k used in the approximations (33), satisfying the accuracy test (41);
- K_{tot}, i.e., the total number of Arnoldi iterations for the approximate solution of the reduced system;
- $\ell_{\min}^{(2)}$–$\ell_{\max}^{(2)}$, i.e. the minimum and maximum number of conjugate gradient iterations required for solving the problems (28). They are defined similarly as $\ell_{\min}^{(1)}$ and $\ell_{\max}^{(1)}$, respectively;
- ℓ_{seq}, i.e. the total number of conjugate gradient iterations for solving (8) sequentially;
- max err, i.e. the maximum absolute error in the parallel solution, with respect to the sequential one;
- s_p, which is a rough estimate of the maximum speed-up of the parallel algorithm, over its sequential implementation, defined as
$$s_p = \frac{\ell_{\text{seq}}}{\ell_{\max}^{(1)} + K_{\text{tot}} + \ell_{\max}^{(2)}}.$$
Indeed, as said above, we can assume that each conjugate gradient and Arnoldi iteration has, practically, the same computational cost. This cost, in turn, constitutes the bulk of the computation.

For completeness, in Fig. 1 we also plot the error in the computed solution, after the initial parallel step (11) (see also Remark 1) and after the parallel updates (28), in the case where $\nu = 100$, $N = 50$, and $p = 8$. We observe that in the window which concerns the first processor (“proc 1”, in the figure) the error is zero, since on this processor (see Remark 1) the discrete solution is directly computed, as in the sequential algorithm, by solving the system (18). We observe that the “spikes” at the beginning of each window are due to the use of the approximation (33), whose accuracy criterion (see (41)) has been chosen in order to obtain the same maximum global error of the sequential implementation.

From the obtained results, one has that

$$\ell_{\min}^{(1)} \approx \ell_{\min}^{(2)}, \qquad \ell_{\max}^{(1)} \approx \ell_{\max}^{(2)}.$$

Consequently, it follows that the maximum asymptotic speed-up for the parallel algorithm is $p/2$. Indeed, it would be (approximately) achieved, provided that the choice of the coarse mesh (2) would result in

$$K_{\text{tot}} \ll \ell_{\min}^{(j)} = \ell_{\max}^{(j)}, \quad j = 1, 2.$$

This is certainly not the case for some of the considered tests (as also confirmed by the column labeled s_p in the tables), even though it is a realistic goal, at least for linear and quasi-linear parabolic problems. Moreover, we observe that in each table the columns labeled $K_{\min}$–$K_{\max}$ and K_{tot} contain almost exactly the same values. This is a remarkable result which confirms that the sequential section of the algorithm has a cost which is independent of N, as we expect from the theory. We would like to emphasize that the maximum asymptotic speed-up $p/2$ of the algorithm is considerably better than that of the “Parareal” algorithm, which cannot exceed p/k^*, if k^* ($\geqslant 2$) iterations of (24) are needed for its convergence.

5.1. Summary and concluding remarks

In this paper we have recalled the basic facts regarding a parallel method “across the steps” for solving ODE-IVPs. Such method is characterized by the definition of the reduced system (14), whose solution allows to decouple the original problem into p independent sub-problems. Moreover, we have put into evidence the existing connections with a subsequent approach, known as “Parareal” algorithm. In fact, the latter is obtained when an iterative solution of the reduced system (14), induced by a suitable operator splitting, is considered. At last, we propose a straight modification of the first parallel method, able to cope with problems with large and sparse Jacobians, in particular problems deriving by the solution, via the method of lines, of linear or quasi-linear parabolic problems. Numerical tests, carried out on a prototype problem, show that the maximum asymptotic speed-up for this approach, when p parallel processor are used, is $p/2$.

References

[1] P. Amodio, L. Brugnano, Parallel factorizations and parallel solvers for tridiagonal linear systems, Linear Algebra Appl. 172 (1992) 347–364.
[2] P. Amodio, L. Brugnano, The parallel QR factorization algorithm for tridiagonal linear systems, Parallel Computing 21 (1995) 1097–1110.
[3] P. Amodio, L. Brugnano, Parallel implementation of block boundary value methods for ODEs, J. Comput. Appl. Math. 78 (1997) 197–211.
[4] P. Amodio, L. Brugnano, Parallel ODE solvers based on block BVMs, Adv. Comput. Math. 7 (1997) 5–26.
[5] P. Amodio, L. Brugnano, ParalleloGAM: a parallel code for ODEs, Appl. Numer. Math. 28 (1998) 95–106.
[6] P. Amodio, I. Gladwell, G. Romanazzi, Numerical solution of general Bordered ABD linear systems by cyclic reduction, J. Numer. Anal. Ind. Appl. Math. 1 (2006) 5–12.
[7] P. Amodio, G. Romanazzi, BABDCR: a Fortran 90 package for the solution of Bordered ABD linear systems, ACM Trans. Math. Software 32 (2006) 597–608.
[8] P. Amodio, L. Brugnano, T. Politi, Parallel factorizations for tridiagonal matrices, SIAM J. Numer. Anal. 30 (1993) 813–823.
[9] C. Le Bris, Computational chemistry from the perspective of numerical analysis, Acta Numerica 14 (2005) 363–444.
[10] L. Brugnano, D. Trigiante, On the potentiality of sequential and parallel codes based on extended trapezoidal rules (ETRs), Appl. Numer. Math. 25 (1997) 169–184.
[11] L. Brugnano, D. Trigiante, Parallel implementation of block boundary value methods on nonlinear problems: theoretical results, Appl. Numer. Math. 78 (1997) 197–211.
[12] L. Brugnano, D. Trigiante, Solving Differential Problems by Multistep Initial and Boundary Value Methods, Gordon and Breach, Amsterdam, 1998.
[13] K. Burrage, Parallel and Sequential Methods for Ordinary Differential Equations, Clarendon Press, Oxford, 1995.
[14] K. Burrage (Ed.), Parallel Methods for ODEs, Adv. Comput. Math. 7 (1–2) (1997).
[15] M.J. Gander, E. Hairer, Nonlinear convergence analysis for the Parareal algorithm, in: U. Langer, M. Discacciati, D.E. Keyes, O.B. Widlund, W. Zulehner (Eds.), Domain Decomposition Methods in Science and Engineering XVII, in: Lecture Notes in Computational Science and Engineering, vol. 60, Springer, Berlin, 2008, pp. 45–56.

[16] M.J. Gander, S. Vandewalle, On the superlinear and linear convergence of the Parareal algorithm, in: O.B. Widlund, D.E. Keyes (Eds.), Domain Decomposition Methods in Science and Engineering XVI, in: Lecture Notes in Computational Science and Engineering, vol. 55, Springer, Berlin, 2007, pp. 291–298.

[17] M.J. Gander, S. Vandewalle, Analysis of the Parareal time-parallel time-integration method, SIAM J. Sci. Comput. 29 (2007) 556–578.

[18] M. Hochbruck, C. Lubich, On Krylov subspace approximations to the matrix exponential operator, SIAM J. Numer. Anal. 34 (1997) 1911–1925.

[19] J.L. Lions, Y. Maday, G. Turinici, Résolution d'EDP par un schéma en temps "pararéel", C. R. Acad. Sci. Paris, Ser. I 332 (2001) 661–668.

[20] Y. Maday, G. Turinici, A parareal in time procedure for the control of partial differential equations, C. R. Acad. Sci. Paris, Ser. I 335 (2002) 387–392.

[21] Y. Maday, G. Turinici, The Parareal in time iterative solver: a further direction to parallel implementation, in: T.J. Barth, M. Griebel, D.E. Keyes, R.M. Nieminen, D. Roose, T. Schlick, R. Kornhuber, R. Hoppe, J. Périaux, O. Pironneau, O. Widlund, J. Xu (Eds.), Domain Decomposition Methods in Science and Engineering, in: Lecture Notes in Computational Science and Engineering, vol. 40, Springer, Berlin, 2005, pp. 441–448.

[22] I. Moret, P. Novati, RD-rational approximations of the matrix exponential, BIT 44 (2004) 595–615.

[23] Y. Saad, Analysis of some Krylov subspace approximations to the matrix exponential operator, SIAM J. Numer. Anal. 29 (1992) 209–228.

[24] G.A. Staff, The Parareal algorithm. A survey of present work, Report of the Norwegian University of Science and Technology, Dept. of Math. Sciences, 2003.

[25] S. Ulbrich, Generalized SQP-methods with "Parareal" time decomposition for time-dependent PDE-constrained optimization, Technical Report, Fachbereich Mathematik, TU Darmstadt, 2005.