FLORE
Repository istituzionale dell'Università degli Studi di Firenze

# Parallel Factorizations in Numerical Analysis

Questa è la Versione finale referata (Post print/Accepted manuscript) della seguente pubblicazione:

*Original Citation:*

Parallel Factorizations in Numerical Analysis / Pierluigi Amodio; Luigi Brugnano. - In: SCALABLE COMPUTING. PRACTICE AND EXPERIENCE. - ISSN 1895-1767. - ELETTRONICO. - 10, No.4:(2009), pp. 385-396.

*Availability:*

The webpage https://hdl.handle.net/2158/369363 of the repository was last updated on

*Terms of use:*

Open Access

La pubblicazione è resa disponibile sotto le norme e i termini della licenza di deposito, secondo quanto stabilito dalla Policy per l'accesso aperto dell'Università degli Studi di Firenze (https://www.sba.unifi.it/upload/policy-oa-2016-1.pdf)

*Publisher copyright claim:*

La data sopra indicata si riferisce all'ultimo aggiornamento della scheda del Repository FloRe - The above-mentioned date refers to the last update of the record in the Institutional Repository FloRe

(Article begins on next page)

18 April 2025

Scalable Computing: Practice and Experience
Volume 10, Number 4, pp. 385–396. http://www.scpe.org
ISSN 1895-1767
© 2009 SCPE

# PARALLEL FACTORIZATIONS IN NUMERICAL ANALYSIS*

PIERLUIGI AMODIO† AND LUIGI BRUGNANO‡

**Abstract.** In this paper we review the parallel solution of sparse linear systems, usually deriving by the discretization of ODE-IVPs or ODE-BVPs. The approach is based on the concept of *parallel factorization* of a (block) tridiagonal matrix. This allows to obtain efficient parallel extensions of many known matrix factorizations, and to derive, as a by-product, a unifying approach to the parallel solution of ODEs.

**Key words:** ordinay differential equations (ODEs), initial value problems (IVPs), boundary value problems (BVPs), parallel factorizations, linear systems, sparse matrices, parallel solution, "parareal" algorithm.

**1. Introduction.** The numerical solution of ODEs requires the solution of sparse and structured linear systems. The parallel solution of these problems may be obtained in two ways: for BVPs, since the size of the associated linear system is large, we need to develop parallel solvers for the obtained linear systems; for IVPs we need to define appropriate numerical methods that allow to obtain "parallelizzable" linear systems.

In both cases, the main problem can then be taken back to the solution of special sparse linear systems, whose solution is here approached through the use of *parallel factorizations*, originally introduced for deriving efficient parallel tridiagonal solvers [2, 9], and subsequently generalized to block tridiagonal, Almost Block Diagonal (ABD), and Bordered Almost Block Diagonal (BABD) systems [10, 11, 15, 16, 17, 19].

With this premise, the structure of the paper is the following: in Section 2 the main facts about *parallel factorizations* and their extensions are briefly recalled; then, in Section 3 their application for solving ODE problems is sketched; finally, in Section 4 we show that this approach also encompasses the so called "Parareal" algorithm, recently introduced in [23, 24].

**2. Parallel factorizations.** In this section we consider several parallel algorithms in the class of partition methods for the solution of linear systems,

$$Ax = f, \tag{2.1}$$

where $A$ is a $n \times n$ sparse and structured matrix, and $x$ and $f$ are vectors of length $n$. We will investigate the parallel solution of (2.1) on $p$ processors, supposing $p \ll n$ in order for the number of sequential operations to be much smaller than that of parallel ones.

The coefficient matrices $A$ here considered are (block) banded, tridiagonal, bidiagonal, or even Almost Block Diagonal (ABD). All these structures may be rearranged in the form

$$A = \begin{pmatrix} A^{(1)} & \boldsymbol{c}_1^{(1)} & & & & & \\ \boldsymbol{b}_1^{(1)^T} & a^{(1)} & \boldsymbol{c}_0^{(2)^T} & & & & \\ & \boldsymbol{b}_0^{(2)} & A^{(2)} & \boldsymbol{c}_1^{(2)} & & & \\ & & \boldsymbol{b}_1^{(2)^T} & a^{(2)} & & & \\ & & & & \ddots & & \\ & & & & & a^{(p-1)} & \boldsymbol{c}_0^{(p)^T} \\ & & & & & \boldsymbol{b}_0^{(p)} & A^{(p)} \end{pmatrix} \tag{2.2}$$

where the diagonal blocks are square and the superscript $(i)$ indicates that this block is handled only by processor $i$. The size of the blocks $a^{(i)}$, $A^{(i)}$, $\boldsymbol{b}_j^{(i)}$, and $\boldsymbol{c}_j^{(i)}$ is in general independent of both $i$ and $j$, and only depends on the sparsity structure of the coefficient matrix $A$. In particular, the size of the blocks $a^{(i)}$ is quite important, since the sequential section of the algorithm is proportional to it. Therefore, the blocks $a^{(i)}$ should be as small as possible. As an example, if $A$ is (block) tridiagonal, $a^{(i)}$ reduces to a single (block) entry. Vice versa, in case of banded (block) matrices, the (block) size of $a^{(i)}$ equals to $\max(s, r)$, where $s$ and $r$ denote

*Work developed within the project "Numerical methods and software for differential equations"
†Dipartimento di Matematica, Università di Bari, Bari, Italy (amodio@dm.uniba.it)
‡Dipartimento di Matematica, Università di Firenze, Firenze, Italy (luigi.brugnano@unifi.it)

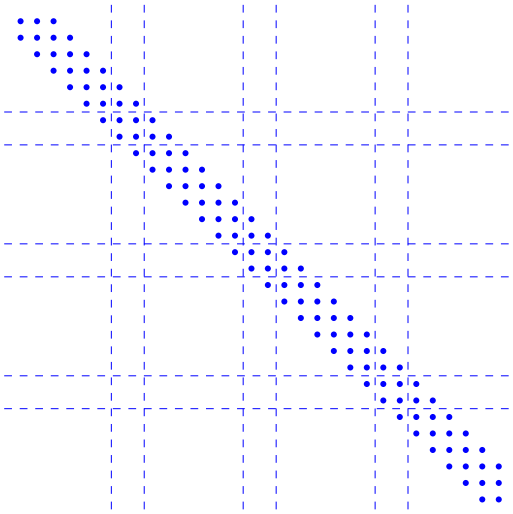

FIG. 2.1. *Partitioning of a banded matrix. Each point represents a (block) entry of the matrix.*

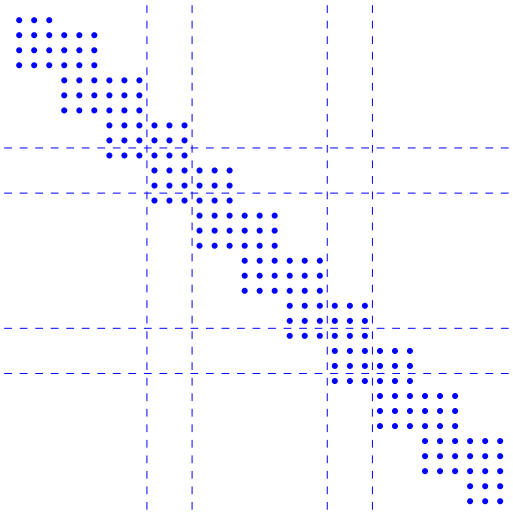

FIG. 2.2. *Partitioning of an ABD matrix. Each point represents an entry of the matrix.*

the number of lower and upper off (block) diagonals (see Figure 2.1), respectively. In case of ABD matrices, $a^{(i)}$ is a block of size equal to the number of rows in each block row of the coefficient matrix (see Figure 2.2). Since row and column permutations inside each block do not destroy the sparsity structure of the coefficient matrix, in ABD matrices we may permute the elements inside $a^{(i)}$ to improve stability properties. Blocks $A^{(i)}$ have the same sparsity structure as the original matrix, and are locally handled by using any suitable sequential algorithm.

In order to keep track of any parallel algorithm, we consider the following factorization [2, 9]

$$A = F\,T\,G, \tag{2.3}$$

where

$$F = \begin{pmatrix} N^{(1)} & \boldsymbol{o} & & & & & & \\ \boldsymbol{v}^{(1)^T} & I & \boldsymbol{w}^{(2)^T} & & & & & \\ & \boldsymbol{o} & N^{(2)} & \boldsymbol{o} & & & & \\ & & \boldsymbol{v}^{(2)^T} & I & \boldsymbol{w}^{(3)^T} & & & \\ & & & \boldsymbol{o} & N^{(3)} & \boldsymbol{o} & & \\ & & & & \boldsymbol{v}^{(3)^T} & I & & \\ & & & & & & \ddots & \\ & & & & & & I & \boldsymbol{w}^{(p)^T} \\ & & & & & & \boldsymbol{o} & N^{(p)} \end{pmatrix}, \tag{2.4}$$

$$T = \begin{pmatrix} \hat{I} & \boldsymbol{o} & & & & & & \\ \boldsymbol{o}^T & \alpha^{(1)} & \boldsymbol{o}^T & \gamma^{(2)} & & & & \\ & \boldsymbol{o} & \hat{I} & \boldsymbol{o} & & & & \\ & \beta^{(2)} & \boldsymbol{o}^T & \alpha^{(2)} & \boldsymbol{o}^T & \gamma^{(3)} & & \\ & & & \boldsymbol{o} & \hat{I} & \boldsymbol{o} & & \\ & & & \beta^{(3)} & \boldsymbol{o}^T & \alpha^{(3)} & & \\ & & & & & & \ddots & \\ & & & & & & \alpha^{(p-1)} & \boldsymbol{o}^T \\ & & & & & & \boldsymbol{o} & \hat{I} \end{pmatrix}, \tag{2.5}$$

$$G = \begin{pmatrix} S^{(1)} & \boldsymbol{y}^{(1)} & & & & & & \\ \boldsymbol{o}^T & I & \boldsymbol{o}^T & & & & & \\ & \boldsymbol{z}^{(2)} & S^{(2)} & \boldsymbol{y}^{(2)} & & & & \\ & & \boldsymbol{o}^T & I & \boldsymbol{o}^T & & & \\ & & & \boldsymbol{z}^{(3)} & S^{(3)} & \boldsymbol{y}^{(3)} & & \\ & & & & \boldsymbol{o}^T & I & & \\ & & & & & & \ddots & \\ & & & & & & I & \boldsymbol{o}^T \\ & & & & & & \boldsymbol{z}^{(p)} & S^{(p)} \end{pmatrix}, \tag{2.6}$$

$I$, $\hat{I}$ and $\boldsymbol{o}$ are identity and null matrices of appropriate sizes, and $N^{(i)}S^{(i)}$ is any suitable factorization of the block $A^{(i)}$. The remaining entries of $F$, $T$, and $G$ can be derived from (2.3) by direct identification.

Factorization (2.3) may be computed in parallel on the $p$ processors. For simplicity, we analyze the factorization of the sub-matrix identified by the superscript $(i)$ (with obvious differences for $i = 1$ and $i = p$)

$$M^{(i)} = \begin{pmatrix} 0 & \boldsymbol{c}_0^{(i)^T} & \\ \boldsymbol{b}_0^{(i)} & A^{(i)} & \boldsymbol{c}_1^{(i)} \\ & \boldsymbol{b}_1^{(i)^T} & a^{(i)} \end{pmatrix}. \tag{2.7}$$

Following (2.2)–(2.6) we have

$$M^{(i)} = \begin{pmatrix} I & \boldsymbol{w}^{(i)^T} & \\ \boldsymbol{o} & N^{(i)} & \boldsymbol{o} \\ & \boldsymbol{v}^{(i)^T} & I \end{pmatrix} \begin{pmatrix} \alpha_1^{(i)} & \boldsymbol{o}^T & \gamma^{(i)} \\ \boldsymbol{o} & I & \boldsymbol{o} \\ \beta^{(i)} & \boldsymbol{o}^T & \alpha_2^{(i)} \end{pmatrix} \begin{pmatrix} I & \boldsymbol{o}^T & \\ \boldsymbol{z}^{(i)} & S^{(i)} & \boldsymbol{y}^{(i)} \\ & \boldsymbol{o}^T & I \end{pmatrix}, \tag{2.8}$$

where $\alpha_2^{(i)} + \alpha_1^{(i+1)} = \alpha^{(i)}$, and the other entries are the same as defined in (2.4)–(2.6).

Consequently, by considering any given factorization for $A^{(i)}$, it is possible to derive corresponding parallel extensions of such factorizations, which cover most of the parallel algorithms in the class of partition methods. In particular, the following ones easily derive for matrices with well conditioned sub-blocks $A^{(i)}$ (this means, for example, that pivoting is unnecessary or does not destroy the sparsity structure):

- *LU factorization*, by setting in (2.8) $N^{(i)} = L^{(i)}$ and $S^{(i)} = U^{(i)}$, where $L^{(i)}U^{(i)}$ is the $LU$ factorization of the matrix $A^{(i)}$. In this case, the (block) vectors $\boldsymbol{y}^{(i)}$ and $\boldsymbol{v}^{(i)}$ maintain the same sparsity structure as that of $\boldsymbol{c}_1^{(i)}$ and $\boldsymbol{b}_1^{(i)}$, respectively, while the vectors $\boldsymbol{z}^{(i)}$ and $\boldsymbol{w}^{(i)}$ are non-null fill-in (block) vectors, obtained by solving two triangular systems.

- *LUD factorization* (which derives from the Gauss-Jordan elimination algorithm), by setting in (2.8) $S^{(i)} = D^{(i)}$, a diagonal matrix, and

$$\begin{pmatrix} I & \boldsymbol{w}^{(i)^T} & \\ \boldsymbol{o} & N^{(i)} & \boldsymbol{o} \\ & \boldsymbol{v}^{(i)^T} & I \end{pmatrix} = \begin{pmatrix} I & \boldsymbol{o}^T & \\ \boldsymbol{o} & L^{(i)} & \boldsymbol{o} \\ & \boldsymbol{v}^{(i)^T} & I \end{pmatrix} \begin{pmatrix} I & \boldsymbol{w}^{(i)^T} & \\ \boldsymbol{o} & U^{(i)} & \boldsymbol{o} \\ & \boldsymbol{o}^T & I \end{pmatrix},$$

  where $L^{(i)}$ and $U^{(i)}$ are lower and upper triangular matrices, respectively, with unit diagonal. Therefore, $\boldsymbol{v}^{(i)}$ and $\boldsymbol{w}^{(i)}$ maintain the same sparsity structure as that of $\boldsymbol{b}_1^{(i)}$ and $\boldsymbol{c}_0^{(i)}$, respectively, while $\boldsymbol{z}^{(i)}$ and $\boldsymbol{y}^{(i)}$ are non-null fill-in (block) vectors.

- *cyclic reduction* algorithm [2, 9] (see also [1, 12, 15, 16]), which is one of the best known parallel algorithms but that, in its original form, requires a synchronization at each step of reduction. In fact, the idea of this algorithm is to perform several reductions that, at each step, halve the size of the system. On the other hand, to obtain a factorization in the form (2.8), it is possible to consider cyclic reduction as a sequential algorithm to be applied locally,

$$M^{(i)} = (\hat{P}_1^{(i)}\hat{L}_1^{(i)}\hat{P}_2^{(i)}\hat{L}_2^{(i)}\cdots)\hat{D}^{(i)}(\cdots\hat{U}_2^{(i)}\hat{P}_2^{(i)^T}\hat{U}_1^{(i)}\hat{P}_1^{(i)^T}),$$

  where $\hat{P}_i^{(i)}$ are suitable permutation matrices that maintain the first and last row in the reduced matrix. The computational cost, which is much higher if the algorithm is applied to $A$ on a sequential computer, becomes comparable to the previous local factorizations since this algorithm does not compute fill-in block vectors. As a consequence, the corresponding parallel factorization algorithm turns out to be one of the most effective.

- *Alternate row and column elimination* [25] which is an algorithm suitable for ABD matrices. In fact, for such matrices alternate row and column permutations always guarantee stability without fill-in. This feature extends to the parallel algorithm, by taking into account that row permutations between the first block row of $A^{(i)}$ and the block containing $\boldsymbol{c}_0^{(i)}$ (see (2.7)), still make the parallel algorithm stable without introducing fill-in. Such parallel factorization is defined by setting $N^{(i)} = P^{(i)}L^{(i)}$ and $S^{(i)} = U^{(i)}Q^{(i)}$, where $P^{(i)}$ and $Q^{(i)}$ are permutation matrices and $L^{(i)}$ and $U^{(i)}$, after a suitable reordering of the rows and of the columns, are $2 \times 2$ block triangular matrices (see [10] for full details). Finally, the (block) vectors $\boldsymbol{y}^{(i)}$ and $\boldsymbol{z}^{(i)}$ maintain the same sparsity structure as that of $\boldsymbol{b}_1^{(i)}$ and $\boldsymbol{c}_0^{(i)}$, respectively, whereas $\boldsymbol{w}^{(i)}$ and $\boldsymbol{v}^{(i)}$ are fill-in (block) vectors.

For what concerns the solution of the systems associated to the previous parallel factorizations, there is much parallelism inside. The solution of the systems with the matrices $F$ and $G$ may proceed in parallel on the different processors. Conversely, the solution of the system with the matrix $T$ requires a sequential part, consisting in the solution of a *reduced system* with the (block) tridiagonal *reduced matrix*

$$T_p = \begin{pmatrix} \alpha^{(1)} & \gamma^{(2)} & & \\ \beta^{(2)} & \alpha^{(2)} & \ddots & \\ & \ddots & \ddots & \gamma^{(p-1)} \\ & & \beta^{(p-1)} & \alpha^{(p-1)} \end{pmatrix}. \tag{2.9}$$

We observe that the (block) size of $T_p$ only depends on $p$ and is independent of $n$.

For a matrix $A$ with singular or ill-conditioned sub-blocks $A^{(i)}$, the local factorizations may be unstable or even undefined. Consequently, it is necessary to slightly modify the factorization (2.8), in order to obtain stable parallel algorithms. The basic idea is that factorization (2.8) may produce more than two entries in the *reduced system*. In other words, the factorization of $A^{(i)}$ is stopped when the considered sub-block is ill-conditioned (or

the local factorization with a singular factor). As a consequence, the size of the *reduced system* is increased as sketched below. Let then

$$M^{(i)} = \hat{L}_1^{(i)} \hat{D}_1^{(i)} \hat{U}_1^{(i)},$$

where

$$\hat{L}_1^{(i)} = \begin{pmatrix} I & \boldsymbol{w}_1^{(i)^T} & & & \\ \boldsymbol{o} & N_1^{(i)} & \boldsymbol{o} & & \\ & \boldsymbol{v}_1^{(i)^T} & I & \boldsymbol{o}^T & \\ & & \boldsymbol{o} & \hat{I} & \boldsymbol{o} \\ & & & \boldsymbol{o}^T & I \end{pmatrix}, \quad \hat{U}_1^{(i)} = \begin{pmatrix} I & \boldsymbol{o}^T & & & \\ \boldsymbol{z}_1^{(i)} & S_1^{(i)} & \boldsymbol{y}_1^{(i)} & & \\ & \boldsymbol{o}^T & I & \boldsymbol{o}^T & \\ & & \boldsymbol{o} & \hat{I} & \boldsymbol{o} \\ & & & \boldsymbol{o}^T & I \end{pmatrix},$$

$$\hat{D}_1^{(i)} = \begin{pmatrix} \alpha_1^{(i)} & \boldsymbol{o}^T & \gamma_1^{(i)} & & \\ \boldsymbol{o} & I & \boldsymbol{o} & & \\ \beta_1^{(i)} & \boldsymbol{o}^T & \alpha_2^{(i)} & \boldsymbol{c}_2^{(i)^T} & \\ & & \boldsymbol{b}_2^{(i)} & A_2^{(i)} & \boldsymbol{c}_3^{(i)} \\ & & & \boldsymbol{b}_3^{(i)^T} & \alpha_3^{(i)} \end{pmatrix},$$

when the sub-block $A_1^{(i)}$ of $A^{(i)}$,

$$A_1^{(i)} = \begin{pmatrix} N_1^{(i)} & \\ \boldsymbol{v}_1^{(i)^T} & \alpha_2^{(i)} \end{pmatrix} \begin{pmatrix} S_1^{(i)} & \boldsymbol{y}_1^{(i)} \\ & I \end{pmatrix},$$

is singular, because the block $\alpha_2^{(i)}$ is singular (i. e., $\alpha_2^{(i)} = 0$, in the scalar case). Then, $\alpha_2^{(i)}$ is introduced in the *reduced system*. By iterating this procedure on $\hat{D}_1^{(i)}$, we obtain again the factorization (2.3), with the only difference that now the *reduced matrix* in $T_p$ may be of (block) size larger than $p-1$ (compare with (2.9)). However, it may be shown that it still depends only on $p$, whereas it is independent of $n$ [3, 4]. Consequently, the scalar section of the whole algorithm is still negligible, when $n \gg p$.

The parallel algorithms that fall in this class are [3, 4]:

- *LU factorization with partial pivoting*, defined by setting $N_1^{(i)} = (P_1^{(i)})^T L_1^{(i)}$ and $S_1^{(i)} = U_1^{(i)}$ where $P_1^{(i)}$ is a permutation matrix such that $L_1^{(i)} U_1^{(i)}$ is the $LU$ factorization of $P_1^{(i)} A_1^{(i)}$. The remaining (block) vectors are defined similarly as in the case of the $LU$ factorization previously described.
- *QR factorization*, defined by setting $N_1^{(i)} = Q_1^{(i)}$ and $S_1^{(i)} = R_1^{(i)}$. In this case both $\boldsymbol{w}_1^{(i)}$ and $\boldsymbol{z}_1^{(i)}$ are fill-in (block) vectors while $\boldsymbol{v}_1^{(i)}$ and $\boldsymbol{y}_1^{(i)}$ maintain the same sparsity structure as that of the corresponding (block) vectors in $M^{(i)}$.

Factorization (2.3)–(2.6), and the corresponding parallel algorithms mentioned above, are easily generalized to matrices with additional non-null elements in the right-lower and/or left-upper corners. This is the case, for example, of Bordered ABD (BABD) matrices (see Figure 2.3) and matrices with a circulant-like structure (see Figure 2.4). Supposing the non-null elements are located in the right-upper corner (this is always possible by means of suitable permutation), then the coefficient matrix is partitioned in the form

$$A = \begin{pmatrix} a^{(0)} & \boldsymbol{c}_0^{(1)^T} & & & & & & & b \\ \boldsymbol{b}_0^{(1)} & A^{(1)} & \boldsymbol{c}_1^{(1)} & & & & & & \\ & \boldsymbol{b}_1^{(1)^T} & a^{(1)} & \boldsymbol{c}_0^{(2)^T} & & & & & \\ & & \boldsymbol{b}_0^{(2)} & A^{(2)} & \boldsymbol{c}_1^{(2)} & & & & \\ & & & \boldsymbol{b}_1^{(2)^T} & a^{(2)} & & & & \\ & & & & & \ddots & & & \\ & & & & & & a^{(p-1)} & \boldsymbol{c}_0^{(p)^T} & \\ & & & & & & \boldsymbol{b}_0^{(p)} & A^{(p)} & \boldsymbol{c}_1^{(p)} \\ & & & & & & & \boldsymbol{b}_1^{(p)^T} & a^{(p)} \end{pmatrix}, \tag{2.10}$$

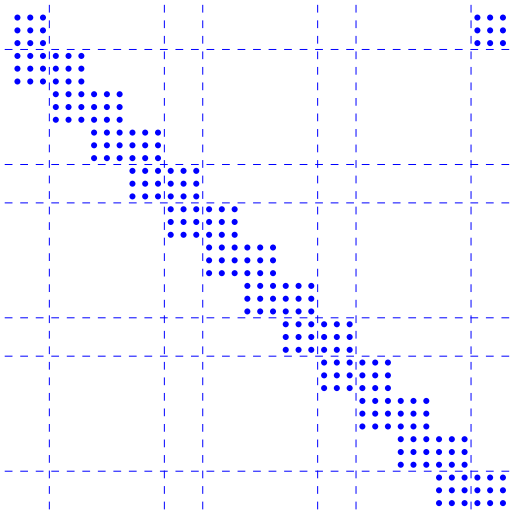

FIG. 2.3. *Partitioning of a BABD matrix. Each point represents an entry of the matrix.*

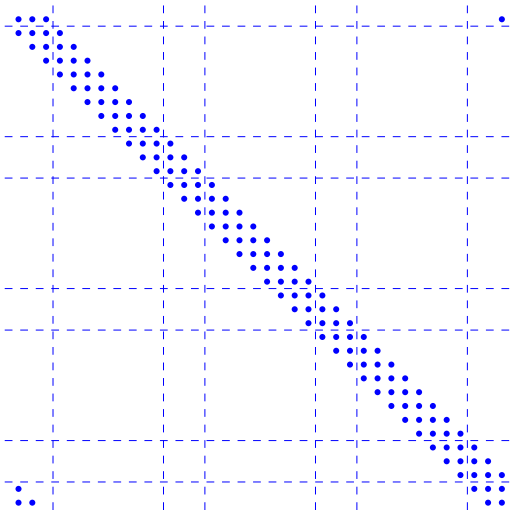

FIG. 2.4. *Partitioning of a matrix with a circulant-like structure. Each point represents a (block) entry of the matrix.*

where $b$ is the smallest rectangular block containing all the corner elements.

A factorization similar to that in (2.3)–(2.6) (the obvious differences are related to the first and last (block) rows) produces a corresponding *reduced system* with the *reduced matrix*

$$T_p = \begin{pmatrix} \alpha^{(0)} & \gamma^{(1)} & & & \beta^{(0)} \\ \beta^{(1)} & \alpha^{(1)} & \gamma^{(2)} & & \\ & \beta^{(2)} & \alpha^{(2)} & \ddots & \\ & & \ddots & \ddots & \gamma^{(p)} \\ & & & \beta^{(p)} & \alpha^{(p)} \end{pmatrix}. \tag{2.11}$$

We observe that, for the very important classes of BABD and circulant-like matrices (the latter, after a suitable row permutation, see Figure 2.5), both the matrix (2.10) and the *reduced matrix* (2.11) have the form of a lower block bidiagonal matrix (i. e., $\boldsymbol{c}_j^{(i)} = 0$ and $\gamma^{(i)} = 0$ for all $i$ and $j$) with an additional right-upper

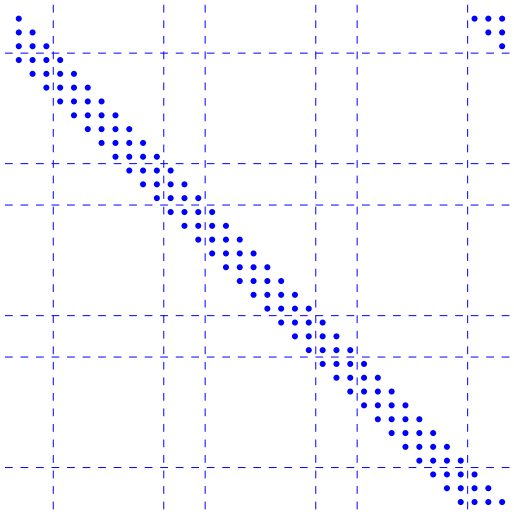

FIG. 2.5. *Partitioning of the matrix in Fig. 2.4 after row permutation. Each point represents a (block) entry of the matrix.*

corner block:

$$A = \begin{pmatrix} a^{(0)} & & & & & & b \\ \boldsymbol{b}_0^{(1)} & A^{(1)} & & & & & \\ & \boldsymbol{b}_1^{(1)^T} & a^{(1)} & & & & \\ & & \boldsymbol{b}_0^{(2)} & \ddots & & & \\ & & & \ddots & a^{(p-1)} & & \\ & & & & \boldsymbol{b}_0^{(p)} & A^{(p)} & \\ & & & & & \boldsymbol{b}_1^{(p)^T} & a^{(p)} \end{pmatrix},$$

and

$$T_p = \begin{pmatrix} \alpha^{(0)} & & & \beta^{(0)} \\ \beta^{(1)} & \alpha^{(1)} & & \\ & \ddots & \ddots & \\ & & \beta^{(p)} & \alpha^{(p)} \end{pmatrix}.$$

We have also to note that, for this kind of matrices, the overall computational cost of a parallel factorization algorithm has a very small increase. On a sequential machine, supposing to maintain the same partitioning of the matrix on $p > 1$ processors, we have a computational cost which is similar to that of any efficient sequential algorithm (the corner block $b$ implies the construction of a fill-in (block) vector) but with better stability properties (see [26]). For this reason, a method that is widely and efficiently applied to matrices in the form (2.10), also on a sequential computer, is cyclic reduction (see [11, 17, 19] where cyclic reduction is applied to BABD matrices).

**3. Parallel solution of differential equations.** The numerical solution of ODE-BVPs leads to the solution of large and sparse linear systems of equations that represent the most expensive part of a BVP code. The sparsity structure of the obtained problem depends on the methods implemented. In general, one-step methods lead to ABD or BABD matrices, depending on the boundary conditions (separated or not, respectively), while multistep methods lead to block banded systems (with additional corner blocks in case of non-separated boundary conditions) [5, 6, 22]. The parallel algorithms previously described perfectly cope with this kind of systems. For this reason we do not investigate further on ODE-BVPs.

Conversely, we shall now consider the application of *parallel factorizations* for deriving parallel algorithms for numerically solving ODE-IVPs, which we assume, for sake of simplicity, to be linear and in the form

$$y' = Ly + g(t), \qquad t \in [t_0, T], \qquad y(t_0) = y_0 \in \mathbb{R}^m, \tag{3.1}$$

which is, however, sufficient to grasp the main features of the approach [5, 6, 7, 8, 20, 21, 22].

Let us consider a suitable *coarse mesh*, defined by the following partition of the integration interval in (3.1):

$$t_0 \equiv \tau_0 < \tau_1 < \cdots < \tau_p \equiv T. \tag{3.2}$$

Suppose, for simplicity, that inside each sub-interval we apply a given method with a constant stepsize

$$h_i = \frac{\tau_i - \tau_{i-1}}{N}, \qquad i = 1, \ldots, p, \tag{3.3}$$

to approximate the problem

$$y' = Ly + g(t), \quad t \in [\tau_{i-1}, \tau_i], \qquad y(\tau_{i-1}) = y_{0i}, \qquad i = 1, \ldots, p. \tag{3.4}$$

If $y(t)$ denotes the solution of problem (3.1), and we denote by

$$y_{ni} \approx y(\tau_{i-1} + nh_i), \qquad n = 0, \ldots, N, \quad i = 1, \ldots, p, \tag{3.5}$$

the entries of the discrete approximation, then, in order for the numerical solutions of (3.1) and (3.4) to be equivalent, we require that (see (3.2) and (3.5))

$$y_{01} = y_0, \qquad y_{0i} \equiv y_{N,i-1}, \quad i = 2, \ldots, p. \tag{3.6}$$

For convention, we also set

$$y_{01} \equiv y_{N0}. \tag{3.7}$$

Let now suppose that the numerical approximations to the solutions of (3.4) are obtained by solving discrete problems in the form

$$M_i \boldsymbol{y}_i = \boldsymbol{v}_i y_{0i} + \boldsymbol{g}_i, \qquad \boldsymbol{y}_i = \begin{pmatrix} y_{1i}, & \ldots & , & y_{Ni} \end{pmatrix}^T, \qquad i = 1, \ldots, p, \tag{3.8}$$

where the matrices $M_i \in \mathbb{R}^{mN \times mN}$ and $\boldsymbol{v}_i \in \mathbb{R}^{mN \times m}$, and the vector $\boldsymbol{g}_i \in \mathbb{R}^{mN}$, do depend on the chosen method (see, e.g., [5, 6], for the case of block BVMs) and on the problems (3.4). 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 (3.1) is obtained by solving a discrete problem in the form (hereafter, $I_r$ will denote the identity matrix of dimension $r$):

$$M\boldsymbol{y} \equiv \begin{pmatrix} I_m & & & & \\ -\boldsymbol{v}_1 & M_1 & & & \\ & -V_2 & M_2 & & \\ & & \ddots & \ddots & \\ & & & -V_p & M_p \end{pmatrix} \begin{pmatrix} y_{N0} \\ \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \\ \vdots \\ \boldsymbol{y}_p \end{pmatrix} = \begin{pmatrix} y_0 \\ \boldsymbol{g}_1 \\ \boldsymbol{g}_2 \\ \vdots \\ \boldsymbol{g}_p \end{pmatrix}, \tag{3.9}$$

$$V_i = [O \,|\, \boldsymbol{v}_i] \in \mathbb{R}^{mN \times mN}, \qquad i = 2, \ldots, p.$$

Obviously, this problem may be solved in a sequential fashion, by means of the iteration (see (3.6)-(3.7)):

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

Nevertheless, by following arguments similar to those in the previous section, we consider the factorization:

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

where (see (3.9))

$$W_i = [O \,|\, \boldsymbol{w}_i] \in \mathbb{R}^{mN \times mN}, \qquad \boldsymbol{w}_i = M_i^{-1} \boldsymbol{v}_i \in \mathbb{R}^{mN \times m}. \tag{3.10}$$

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

$$M_i \boldsymbol{z}_i = \boldsymbol{g}_i, \qquad \boldsymbol{z}_i = \left( \; z_{1i}, \; \ldots \; , \; z_{Ni} \; \right)^T, \qquad i = 1, \ldots, p, \tag{3.11}$$

and, then, (see (3.10) and (3.6)) recursively update the local solutions,

$$\begin{aligned} &\boldsymbol{y}_1 = \boldsymbol{z}_1 + \boldsymbol{w}_1 y_{01}, \\ &\boldsymbol{y}_i = \boldsymbol{z}_i + W_i \boldsymbol{y}_{i-1} \equiv \boldsymbol{z}_i + \boldsymbol{w}_i y_{0i}, \qquad i = 2, \ldots, p. \end{aligned} \tag{3.12}$$

The latter recursion, however, has still much parallelism. Indeed, if we consider the partitionings (see (3.8), (3.11), and (3.10))

$$\boldsymbol{y}_i = \begin{pmatrix} \widehat{\boldsymbol{y}}_i \\ y_{Ni} \end{pmatrix}, \qquad \boldsymbol{z}_i = \begin{pmatrix} \widehat{\boldsymbol{z}}_i \\ z_{Ni} \end{pmatrix}, \qquad \boldsymbol{w}_i = \begin{pmatrix} \widehat{\boldsymbol{w}}_i \\ w_{Ni} \end{pmatrix}, \quad w_{Ni} \in \mathbb{R}^{m \times m}, \tag{3.13}$$

then (3.12) is equivalent to solve, at first, the *reduced system*

$$\begin{pmatrix} I_m & & & \\ -w_{N1} & I_m & & \\ & \ddots & \ddots & \\ & & -w_{N,p-1} & I_m \end{pmatrix} \begin{pmatrix} y_{01} \\ y_{02} \\ \vdots \\ y_{0p} \end{pmatrix} = \begin{pmatrix} y_0 \\ z_{N1} \\ \vdots \\ z_{N,p-1} \end{pmatrix}, \tag{3.14}$$

i. e.,

$$y_{01} = y_0, \qquad y_{0,i+1} = z_{Ni} + w_{Ni} y_{0i}, \qquad i = 1, \ldots, p-1, \tag{3.15}$$

after which performing the $p$ parallel updates

$$\widehat{\boldsymbol{y}}_i = \widehat{\boldsymbol{z}}_i + \widehat{\boldsymbol{w}}_i y_{0i}, \qquad i = 1, \ldots, p-1, \qquad \boldsymbol{y}_p = \boldsymbol{z}_p + \boldsymbol{w}_p y_{0p}. \tag{3.16}$$

We observe that:

- the parallel solution of the $p$ systems in (3.11) is equivalent to compute the approximate solution of the following $p$ ODE-IVPs,

$$z' = Lz + g(t), \quad t \in [\tau_{i-1}, \tau_i], \quad z(\tau_{i-1}) = 0, \quad i = 1, \ldots, p, \tag{3.17}$$

  in place of the corresponding ones in (3.4);
- the solution of the *reduced system* (3.14)-(3.15) consists in computing the proper initial values $\{y_{0i}\}$ for the previous ODE-IVPs;
- the parallel updates (3.16) update the approximate solutions of the ODE-IVPs (3.17) to those of the corresponding ODE-IVPs in (3.4).

REMARK 1. *Clearly, the solution of the first (parallel) system in (3.11) and the first (parallel) update in (3.12) (see also (3.16)) can be executed together, by solving the linear system (see (3.6))*

$$M_1 \boldsymbol{y}_1 = \boldsymbol{g}_1 + \boldsymbol{v}_1 y_0, \tag{3.18}$$

*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* (3.14) and of the parallel updates (3.16) is small, with respect to that of (3.11) (see [5, 6] for more details). This is, indeed, the case when the parameter $N$ in (3.3) is large enough and the coarse partition (3.2) can be supposed to be *a priori* given.

**4. Connections with the "Parareal" algorithm.** We now briefly describe the "Parareal" algorithm introduced in [23, 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 us consider the problem

$$\frac{\partial}{\partial t} y = \mathcal{L}\, y, \qquad t \in [t_0, T], \qquad y(t_0) = y_0, \tag{4.1}$$

where $\mathcal{L}$ is an operator from a Hilbert space $V$ into $V'$. Let us consider again the partition (3.2) of the time interval, and consider the problems

$$\frac{\partial}{\partial t} y = \mathcal{L}\, y, \qquad t \in [\tau_{i-1}, \tau_i], \qquad y(\tau_{i-1}) = y_{0i}, \qquad i = 1, \ldots, p. \tag{4.2}$$

Clearly, in order for (4.1) and (4.2) to be equivalent, one must require that

$$y_{0i} = y(\tau_{i-1}), \qquad i = 1, \ldots, p. \tag{4.3}$$

The initial data (4.3) are then formally related by means of suitable *propagators* $\mathcal{F}_i$ such that

$$y_{0,i+1} = \mathcal{F}_i y_{0i}, \qquad i = 1, \ldots, p-1. \tag{4.4}$$

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

$$F\boldsymbol{y} \equiv \begin{pmatrix} \mathcal{I} & & & \\ -\mathcal{F}_1 & \mathcal{I} & & \\ & \ddots & \ddots & \\ & & -\mathcal{F}_{p-1} & \mathcal{I} \end{pmatrix} \begin{pmatrix} y_{01} \\ y_{02} \\ \vdots \\ y_{0p} \end{pmatrix} = \begin{pmatrix} y_0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \equiv \boldsymbol{\eta}. \tag{4.5}$$

For solving (4.5), the authors essentially define the splitting

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

with *coarse propagators*

$$\mathcal{G}_i \approx \mathcal{F}_i, \qquad i = 1, \ldots, p,$$

and consider the iterative procedure

$$G\boldsymbol{y}^{(k+1)} = (G - F)\boldsymbol{y}^{(k)} + \boldsymbol{\eta}, \qquad k = 0, 1, \ldots,$$

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

$$\begin{aligned} &y_{01}^{(k+1)} = y_0, \\ &y_{0,i+1}^{(k+1)} = \mathcal{G}_i y_{0i}^{(k+1)} + (\mathcal{F}_i - \mathcal{G}_i) y_{0i}^{(k)}, \qquad i = 1, \ldots, p-1, \end{aligned} \tag{4.6}$$

thus providing good parallel features, if we can assume that the coarse operators $\mathcal{G}_i$ are "cheap" enough. The iteration (4.6) defines the "Parareal" algorithm, which is iterated until

$$\|y_{0i}^{(k+1)} - y_{0i}^{(k)}\|, \qquad i = 2, \ldots, p,$$

are suitably small. In the practice, in case of linear operators, problem (4.1) becomes, via the method of lines, an ODE in the form (3.1), with $L$ a huge and very sparse matrix. Consequently, problems (4.2) become in the form (3.4). Similarly, the propagator $\mathcal{F}_i$ consists in the application of a suitable discrete method for approximating the solution of the corresponding $i$th problem in (3.4), 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 (3.8), then the discrete version of the recurrence (4.4) becomes exactly (3.15), as well as the discrete counterpart of the matrix form (4.5) becomes (3.14).

We can then conclude that the "Parareal" algorithm in [23, 24] *exactly* coincides with the iterative solution of the *reduced system* (3.14), induced by a suitable splitting.

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 $\{\boldsymbol{w}_i\}$ in (3.10), and then of the matrices $\{w_{Ni}\}$ (see (3.13)) would be clearly impractical. Moreover, it can be considerably improved by observing that

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

Consequently, by considering a suitable approximation to the matrix exponential, the corresponding parallel algorithm turns out to become semi-iterative and potentially very effective, as recently shown in [8].

REFERENCES

[1] P. Amodio. Optimized cyclic reduction for the solution of linear tridiagonal systems on parallel computers. *Comput. Math. Appl.* **26** (1993) 45–53.

[2] P. Amodio and L. Brugnano. Parallel factorizations and parallel solvers for tridiagonal linear systems. *Linear Algebra Appl.* **172** (1992) 347–364.

[3] P. Amodio and L. Brugnano, The parallel $QR$ factorization algorithm for tridiagonal linear systems. *Parallel Comput.* **21** (1995) 1097–1110.

[4] P. Amodio and L. Brugnano. Stable Parallel Solvers for General Tridiagonal Linear Systems. *Z. Angew. Math. Mech.* **76**, suppl. 1 (1996) 115–118.
[5] P. Amodio and L. Brugnano. Parallel implementation of block boundary value methods for ODEs, *J. Comput. Appl. Math.* **78** (1997) 197–211.
[6] P. Amodio and L. Brugnano. Parallel ODE solvers based on block BVMs, *Adv. Comput. Math.* **7** (1997) 5–26.
[7] P. Amodio and L. Brugnano. ParalleloGAM: a Parallel Code for ODEs, *Appl. Numer. Math.* **28** (1998) 95–106.
[8] P. Amodio and L. Brugnano. Parallel solution in time of ODEs: some achievements and perspectives. *Appl. Numer. Math.* **59** (2009) 424–435.
[9] P. Amodio, L. Brugnano and T. Politi. Parallel factorizations for tridiagonal matrices. *SIAM J. Numer. Anal.* **30** (1993) 813–823.
[10] P. Amodio, J.R. Cash, G. Roussos, R.W. Wright, G. Fairweather, I. Gladwell, G.L. Kraut and M. Paprzycki. Almost block diagonal linear systems: sequential and parallel solution techniques, and applications. *Numer. Linear Algebra Appl.* **7**, no. 5 (2000) 275–317.
[11] P. Amodio, I. Gladwell and G. Romanazzi. Numerical solution of general Bordered ABD linear systems by cyclic reduction. *JNAIAM J. Numer. Anal. Ind. Appl. Math.* **1**, no. 1 (2006) 5–12.
[12] P. Amodio, N. Mastronardi. A parallel version of the cyclic reduction algorithm on a Hypercube. *Parallel Comput.* **19** (1993) 1273–1281.
[13] P. Amodio and F. Mazzia. A parallel Gauss-Seidel method for block tridiagonal linear systems. *SIAM J. Sci. Comput.* **16** (1995) 1451–1461.
[14] P. Amodio and F. Mazzia. Parallel iterative solvers for banded linear systems. *Lecture Notes in Comput. Sci.* **1196**, (Numerical Analysis and its Applications. L. Vulkov, J. Wàsniewski, and P. Yalamov editors, Springer, Berlin), 17–24, 1997.
[15] P. Amodio and M. Paprzycki. Parallel solution of Almost Block Diagonal systems on a Hypercube. *Linear Algebra Appl.* **241-243** (1996) 85–103.
[16] P. Amodio and M. Paprzycki. On the parallel solution of Almost Block Diagonal systems. *Control Cybernet.* **25**, no. 3 (1996) 645–656.
[17] P. Amodio and M. Paprzycki. A cyclic reduction approach to the numerical solution of boundary value ODEs. *SIAM J. Sci. Comput.* **18**, no. 1 (1997) 56–68.
[18] P. Amodio, M. Paprzycki and T. Politi. A survey of parallel direct methods for block bidiagonal linear systems on distributed memory computers. *Comput. Math. Appl.* **31** (1996) 111–127.
[19] P. Amodio and G. Romanazzi. BABDCR: a Fortran 90 package for the solution of Bordered ABD systems. *ACM Trans. Math. Software* **32**, no. 4 (2006) 597–608. (Available on the url: `http://www.netlib.org/toms/859`)
[20] L. Brugnano and D. Trigiante. On the potentiality of sequential and parallel codes based on extended trapezoidal rules (ETRs), *Appl. Numer. Math.* **25** (1997) 169–184.
[21] L. Brugnano and D. Trigiante. Parallel implementation of block boundary value methods on nonlinear problems: theoretical results, *Appl. Numer. Math.* **78** (1997) 197–211.
[22] L. Brugnano and D. Trigiante. *Solving Differential Problems by Multistep Initial and Boundary Value Methods*, Gordon and Breach, Amsterdam, 1998.
[23] J. L. Lions, Y. Maday and G. Turinici. Résolution d'EDP par un schéma en temps "pararéel", *C. R. Acad. Sci. Paris, Série I* **332** (2001) 661–668.
[24] Y. Maday and G. Turinici. A parareal in time procedure for the control of partial differential equations, *C. R. Acad. Sci. Paris, Série I* **335** (2002) 387–392.
[25] J. M. Varah. Alternate row and column elimination for solving certain linear systems. *SIAM J. Numer. Anal.* **13** (1976) 71–75.
[26] S. J. Wright. A collection of problems for which Gaussian elimination with partial pivoting is unstable. *SIAM J. Sci. Stat. Comput.* **14** (1993) 231–238.

*Edited by:* Francesca Mazzia
*Received:* June 27, 2009
*Accepted:* September 10, 2009