Parallel Factorizations and Parallel Solvers for Tridiagonal Linear Systems*

P. Amodio and L. Brugnano†
Dipartimento di Matematica
Università di Bari,
I-70125 Bari, Italy

Submitted by Robert J. Plemmons

ABSTRACT

We formalize the concept of *parallel factorization* as a set of scalar factorizations. By means of this concept we are able to give a unified approach to the problem of solving tridiagonal linear systems on parallel computers. A parallel tridiagonal solver is associated with each parallel factorization, but a parallel factorization can be associated with many parallel tridiagonal solvers. As an example, some parallel factorizations are obtained by simple extension of well-known scalar ones. Numerical tests, obtained on a network of transputers, are reported for comparison.

1. INTRODUCTION

The solution of tridiagonal linear systems is necessary in many application fields of numerical analysis. The best scalar algorithm (which derives from the LU factorization of the coefficient matrix) is very inefficient if we want to use a parallel or vector computer.

Many parallel tridiagonal solvers have been proposed: the most important are the partition methods [5, 6, 9, 16], the domain decomposition methods

*Work supported by the Ministero della Ricerca Scientifica e Tecnologica (40%) and by the European Community (P.C.A. contract 4040). Partial support was also given by the P.F. Calcolo Parallelo (sottoprogetto 1) of C.N.R.

†E-mail: 00110570@VM.CSATA.IT.

 0024-3795/92/$5.00

[13], and cyclic reduction [1, 7, 10, 11]. In this paper we propose a unified approach to the problem of solving tridiagonal systems on parallel computers when the size of the linear system is much greater than the number of parallel processors used. Our approach is based on the concept of *parallel factorization*, and allows us to optimize existing parallel solvers, as well as to derive new algorithms from the class of partition and domain decomposition methods.

In Sections 3, 4, 5, 6, and 9 five parallel factorizations are considered; in Section 7 one of the corresponding algorithms is described in more detail, and in Section 8 some parallel solvers are compared on a network of transputers.

2. PARALLEL FACTORIZATIONS

Let us consider the problem of solving the linear system

$$A\mathbf{x} = \mathbf{f}, \tag{2.1}$$

whose coefficient matrix is tridiagonal,

$$A = \begin{pmatrix} a_1 & c_1 & & & & \\ b_2 & a_2 & c_2 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & \ddots & \ddots & c_{n-1} \\ & & & & b_n & a_n \end{pmatrix},$$

$$\mathbf{f} = \begin{pmatrix} f_1 & \cdots & f_n \end{pmatrix}^T, \qquad \mathbf{x} = \begin{pmatrix} x_1 & \cdots & x_n \end{pmatrix}^T, \tag{2.2}$$

on a parallel computer with p processors. For simplicity we shall assume $n = kp - 1$. In order to derive a wide class of parallel solvers for (2.1), we consider the following partition of A:

$$\mathbf{A} = \begin{pmatrix} A^{(0)} & c_{k-1}^{(0)}\mathbf{e}_{k-1} & & & & & \\ b_k^{(0)}\mathbf{e}_{k-1}^T & a_k^{(0)} & c_0^{(1)}\mathbf{e}_1^T & 0 & & & \\ & b_1^{(1)}\mathbf{e}_1 & A^{(1)} & c_{k-1}^{(1)}\mathbf{e}_{k-1} & & & \\ & 0 & b_k^{(1)}\mathbf{e}_{k-1}^T & a_k^{(1)} & & & \\ & & & & \ddots & & \\ & & & & & a_k^{(p-2)} & c_0^{(p-1)}\mathbf{e}_1^T \\ & & & & & b_1^{(p-1)}\mathbf{e}_1 & A^{(p-1)} \end{pmatrix}, \tag{2.3}$$

where $\mathbf{e}_1$ and $\mathbf{e}_{k-1}$ are the first and the last unit vectors in R^{k-1} respectively, and

$$A^{(i)} = \begin{pmatrix} a_1^{(i)} & c_1^{(i)} & & \\ b_2^{(i)} & a_2^{(i)} & \ddots & \\ & \ddots & \ddots & c_{k-2}^{(i)} \\ & & b_{k-1}^{(i)} & a_{k-1}^{(i)} \end{pmatrix},$$

$$a_r^{(i)} = a_{ik+r}, \qquad b_r^{(i)} = b_{ik+r}, \qquad c_r^{(i)} = c_{ik+r}. \tag{2.4}$$

If the blocks $A^{(i)}$ are nonsingular, we can factorize A as follows:

$$A = FT, \tag{2.5.1}$$

$$F = \begin{pmatrix} A^{(0)} & 0 & & & & & & \\ \mathbf{v}^{(0)T} & 1 & \mathbf{w}^{(1)T} & 0 & & & & \\ & \mathbf{0} & A^{(1)} & \mathbf{0} & & & & \\ & 0 & \mathbf{v}^{(1)T} & 1 & \mathbf{w}^{(2)T} & 0 & & \\ & & & \mathbf{0} & A^{(2)} & \mathbf{0} & & \\ & & & 0 & \mathbf{v}^{(2)T} & 1 & & \\ & & & & & & \ddots & \\ & & & & & & 1 & \mathbf{w}^{(p-1)T} \\ & & & & & & \mathbf{0} & A^{(p-1)} \end{pmatrix}, \tag{2.5.2}$$

$$T = \begin{pmatrix} I_{k-1} & \mathbf{y}^{(0)} & & & & & & \\ \mathbf{0}^T & \alpha^{(0)} & \mathbf{0}^T & \gamma^{(1)} & & & & \\ & \mathbf{z}^{(1)} & I_{k-1} & \mathbf{y}^{(1)} & & & & \\ & \beta^{(1)} & \mathbf{0}^T & \alpha^{(1)} & \mathbf{0}^T & \gamma^{(2)} & & \\ & & & \mathbf{z}^{(2)} & I_{k-1} & \mathbf{y}^{(2)} & & \\ & & & \beta^{(2)} & \mathbf{0}^T & \alpha^{(2)} & & \\ & & & & & & \ddots & \\ & & & & & & \alpha^{(p-2)} & \mathbf{0}^T \\ & & & & & & \mathbf{z}^{(p-1)} & I_{k-1} \end{pmatrix}, \tag{2.5.3}$$

where I_{k-1} is the identity matrix of order $k-1$ and

$$\mathbf{w}^{(i)} = c_0^{(i)}\mathbf{e}_1, \qquad \mathbf{v}^{(i)} = b_k^{(i)}\mathbf{e}_{k-1},$$

$$\mathbf{z}^{(i)} = b_1^{(i)}\left(A^{(i)}\right)^{-1}\mathbf{e}_1, \qquad \mathbf{y}^{(i)} = c_{k-1}^{(i)}\left(A^{(i)}\right)^{-1}\mathbf{e}_{k-1},$$

$$\alpha^{(i)} = a_k^{(i)} - b_k^{(i)}c_{k-1}^{(i)}\mathbf{e}_{k-1}^T\left(A^{(i)}\right)^{-1}\mathbf{e}_{k-1} - b_1^{(i+1)}c_0^{(i+1)}\mathbf{e}_1^T\left(A^{(i+1)}\right)^{-1}\mathbf{e}_1,$$

$$\beta^{(i)} = -b_1^{(i)}b_k^{(i)}\mathbf{e}_{k-1}^T\left(A^{(i)}\right)^{-1}\mathbf{e}_1, \qquad \gamma^{(i)} = -c_0^{(i)}c_{k-1}^{(i)}\mathbf{e}_1^T\left(A^{(i)}\right)^{-1}\mathbf{e}_{k-1}. \quad (2.6)$$

This formalism is the same used by Johnsson [9] to derive his algorithm. However, from (2.5) it is possible to derive different parallel solvers by observing that the matrices $A^{(i)}$ may be independently factorized.

DEFINITION. We define a *parallel factorization* of the matrix A as a set of p independent factorizations for the blocks $A^{(i)}$, $i = 0, \ldots, p-1$.

The parallel factorizations are very useful in deriving efficient parallel algorithms for solving (2.1); from (2.5) it follows that the solution of (2.1) is completely parallelizable on p processors. The only sequential part of the algorithm concerns the solution of the *reduced system* with the *reduced* matrix [see (2.5.3)]:

$$T_p = \begin{pmatrix} \alpha^{(0)} & \gamma^{(1)} & & \\ \beta^{(1)} & \alpha^{(1)} & \ddots & \\ & \ddots & \ddots & \gamma^{(p-2)} \\ & & \beta^{(p-2)} & \alpha^{(p-2)} \end{pmatrix}_{(p-1)\times(p-1)} . \quad (2.7)$$

By means of (2.5) it is possible to derive many partition methods by simply considering different parallel factorizations of the matrix A. It is also possible to derive domain decomposition methods [13] if we consider the permuted matrix

$$P \quad A \quad P^T, \quad (2.8)$$

where P is the following permutation matrix:

$$P = \begin{pmatrix} I_{k-1} & \mathbf{0} & O & \cdots & & & & \\ O & \mathbf{0} & I_{k-1} & \mathbf{0} & O & \cdots & & \\ O & \mathbf{0} & O & \mathbf{0} & I_{k-1} & \mathbf{0} & O & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \mathbf{0}^T & 1 & \mathbf{0}^T & \cdots & & & & \\ \mathbf{0}^T & 0 & \mathbf{0}^T & 1 & \mathbf{0}^T & \cdots & & \\ \mathbf{0}^T & 0 & \mathbf{0}^T & 0 & \mathbf{0}^T & 1 & \mathbf{0}^T & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{pmatrix}.$$

From (2.8) it follows that the reduced system (2.7) can be obtained by performing one step of the block cyclic reduction of the matrix A (see also [9]).

Given the problem (2.1), for each solver derived from a parallel factorization, the reduced matrix (2.7) is the same [this is easily derived from (2.5)]. This fact is very important, because the stability of the corresponding parallel solver depends on:

(1) the stability of the parallel factorization;
(2) the stability of the algorithm for solving the reduced system.

Consequently, the two problems can be examined separately.

Regarding the reduced matrix (2.7), if the parallel factorization (2.5) exists, then the following results hold true [2, 9]:

THEOREM 2.1. *The reduced matrix* (2.7) *preserves the properties of diagonal dominance of the matrix* (2.2).

THEOREM 2.2. *If the matrix* (2.2) *is irreducible, then the reduced matrix* (2.7) *is irreducible.*

THEOREM 2.3. *The reduced matrix* (2.7) *preserves the symmetry and positive definiteness of the matrix* (2.2).

Let us derive an alternative formalism to represent the partition (2.3), which is more convenient for our purposes. Let us consider the $(n + p - 1) \times n$ matrix R_p recursively defined as follows:

$$R_1 = I_{k-1}, \qquad R_i = \begin{pmatrix} I_{k-1} & & \\ & \begin{matrix} 1 \\ 1 \end{matrix} & \\ & & R_{i-1} \end{pmatrix}, \quad i = 2, \ldots, p.$$

We have

$$A = R_p^T \quad M \quad R_p,$$

where M is block diagonal:

$$M = \begin{pmatrix} M^{(0)} & & \\ & \ddots & \\ & & M^{(p-1)} \end{pmatrix},$$

and

$$M^{(0)} = \begin{pmatrix} A^{(0)} & c_{k-1}^{(0)}\mathbf{e}_{k-1} \\ b_k^{(0)}\mathbf{e}_{k-1}^T & a_k^{(0)} \end{pmatrix},$$

$$M^{(i)} = \begin{pmatrix} 0 & c_0^{(i)}\mathbf{e}_1^T & 0 \\ b_1^{(i)}\mathbf{e}_1 & A^{(i)} & c_{k-1}^{(i)}\mathbf{e}_{k-1} \\ 0 & b_k^{(i)}\mathbf{e}_{k-1}^T & a_k^{(i)} \end{pmatrix}, \qquad i = 1, \ldots, p-2, \quad (2.9)$$

$$M^{(p-1)} = \begin{pmatrix} 0 & c_0^{(p-1)}\mathbf{e}_1^T \\ b_1^{(p-1)}\mathbf{e}_1 & A^{(p-1)} \end{pmatrix}.$$

If the parallel factorization exists, we can state (with obvious differences for $M^{(0)}$ and $M^{(p-1)}$) that

$$M^{(i)} = \begin{pmatrix} 1 & \mathbf{w}^{(i)T} & 0 \\ \mathbf{0} & A^{(i)} & \mathbf{0} \\ 0 & \mathbf{v}^{(i)T} & 1 \end{pmatrix} \begin{pmatrix} \alpha_1^{(i)} & \mathbf{0}^T & \gamma^{(i)} \\ \mathbf{z}^{(i)} & I_{k-1} & \mathbf{y}^{(i)} \\ \beta^{(i)} & \mathbf{0}^T & \alpha_2^{(i)} \end{pmatrix}. \quad (2.10)$$

The vectors $\mathbf{v}^{(i)}, \mathbf{w}^{(i)}, \mathbf{z}^{(i)}, \mathbf{y}^{(i)}$ and the scalars $\beta^{(i)}$ and $\gamma^{(i)}$ are the same as in (2.5). Moreover, we have [see (2.6) and (2.7)]

$$\alpha^{(i)} = \alpha_2^{(i)} + \alpha_1^{(i+1)}, \qquad i = 0, \ldots, p-2,$$

$$\alpha_1^{(i)} = -b_1^{(i)}c_0^{(i)}\mathbf{e}_1^T\left(A^{(i)}\right)^{-1}\mathbf{e}_1, \qquad \alpha_2^{(i)} = a_k^{(i)} - b_k^{(i)}c_{k-1}^{(i)}\mathbf{e}_{k-1}^T\left(A^{(i)}\right)^{-1}\mathbf{e}_{k-1}.$$

From (2.10) it follows that a parallel factorization of the matrix (2.2) can be characterized by the factorizations of the blocks $A^{(i)}$.

Some authors (for instance [6]) also examine the problem of solving the reduced system in parallel. We assume $n \gg p$ (the number of parallel processing units); in this case, that problem is not relevant.

3. TWISTED FACTORIZATION

To this factorization corresponds a parallel method for only $p = 2$ processors [4, 14, 15]. Nevertheless, on two processors this algorithm is optimal; it has the same scalar count of operations as the scalar LU algorithm, that is, $8n$ operations.

For this method, the corresponding parallel factorization is the following [see (2.9)]:

$$A^{(0)} = L^{(0)}U^{(0)}, \qquad A^{(1)} = U^{(1)}L^{(1)}.$$

It follows that

$$M^{(0)} = \begin{pmatrix} L^{(0)} & \mathbf{0} \\ \mathbf{v}^{(0)} & \alpha_2^{(0)} \end{pmatrix} \begin{pmatrix} U^{(0)} & \mathbf{y}^{(0)} \\ \mathbf{0}^T & 1 \end{pmatrix},$$
$$M^{(1)} = \begin{pmatrix} \alpha_1^{(1)} & \mathbf{w}^{(1)T} \\ \mathbf{0} & U^{(1)} \end{pmatrix} \begin{pmatrix} 1 & \mathbf{0}^T \\ \mathbf{z}^{(1)} & L^{(1)} \end{pmatrix}. \tag{3.1}$$

The vectors in (3.1) are defined in a manner such that a LU factorization of $M^{(0)}$ and a UL factorization of $M^{(1)}$ are obtained.

From Theorems 2.1 and 2.2, it follows that the solver is stable if the matrix (2.2) is diagonally dominant or weakly diagonally dominant and irreducible.

4. PARALLEL LU FACTORIZATION

In several papers, for instance in [5, 9], the LU factorization has been used to factorize the blocks $A^{(i)}$. In particular, Johnsson's algorithm has a scalar count of (parallelizable) operations of about $22n$ (see [9]). However, the parallel factorization corresponding to this algorithm is (2.10) with $A^{(i)} = L^{(i)}U^{(i)}$. This leads to a cost of about $18n$ (parallelizable) operations [2].

It is better to consider the alternative parallel factorization defined by

$$M^{(i)} = \begin{pmatrix} \alpha_1^{(i)} & \mathbf{w}^{(i)T} & \gamma^{(i)} \\ \mathbf{0} & L^{(i)} & \mathbf{0} \\ \beta^{(i)} & \mathbf{v}^{(i)T} & \alpha_2^{(i)} \end{pmatrix} \begin{pmatrix} 1 & \mathbf{0}^T & 0 \\ \mathbf{z}^{(i)} & U^{(i)} & \mathbf{y}^{(i)} \\ 0 & \mathbf{0}^T & 1 \end{pmatrix}. \tag{4.1}$$

The matrices $L^{(i)}, U^{(i)}$, the vectors $\mathbf{v}^{(i)}, \mathbf{y}^{(i)}$, and the scalar $\alpha_2^{(i)}$ in (4.1) are defined so that

$$\begin{pmatrix} L^{(i)} & \mathbf{0} \\ \mathbf{v}^{(i)T} & \alpha_2^{(i)} \end{pmatrix} \begin{pmatrix} U^{(i)} & \mathbf{y}^{(i)} \\ \mathbf{0}^T & 1 \end{pmatrix}$$

is an LU factorization of the submatrix [see (2.9)]

$$\begin{pmatrix} A^{(i)} & c_{k-1}^{(i)} \mathbf{e}_{k-1} \\ b_k^{(i)} \mathbf{e}_{k-1}^T & a_k^{(i)} \end{pmatrix}.$$

This implies that the matrix $L^{(i)}$ is lower bidiagonal, the matrix $U^{(i)}$ is upper bidiagonal, and only the last component of the vectors $\mathbf{y}^{(i)}$ and $\mathbf{v}^{(i)}$ is nonzero. Moreover, from (4.1) and (2.9) we obtain

$$L^{(i)}\mathbf{z}^{(i)} = b_1^{(i)}\mathbf{e}_1, \qquad U^{(i)T}\mathbf{w}^{(i)} = c_0^{(i)}\mathbf{e}_1,$$

$$\alpha_1^{(i)} = -\mathbf{w}^{(i)T}\mathbf{z}^{(i)}, \qquad \beta^{(i)} = -\mathbf{v}^{(i)T}\mathbf{z}^{(i)}, \qquad \gamma^{(i)} = -\mathbf{w}^{(i)T}\mathbf{y}^{(i)}.$$

We observe that $\mathbf{z}^{(i)}$ and $\mathbf{w}^{(i)}$ are full vectors (*fill-in* vectors). It is a simple matter to show that the corresponding parallel solver has a scalar count of (parallelizable) operations of about $17n$.

From widely known results concerning the LU factorizations and from Theorems 2.1 and 2.2, it follows that the parallel solver corresponding to (4.1) is stable if A is diagonally dominant or weakly diagonally dominant and irreducible.

5. PARALLEL *LUD* FACTORIZATION

An alternative factorization, which preserves the band structure of the matrix $A^{(i)}$, derives from the Gauss-Jordan elimination algorithm. The matrix $A^{(i)}$ is factorized in the form $L^{(i)}U^{(i)}D^{(i)}$ where $L^{(i)}$ and $U^{(i)}$ are lower and

upper bidiagonal with unitary diagonal entries, respectively, and $D^{(i)}$ is diagonal. It follows that

$$M^{(i)} = \begin{pmatrix} 1 & \mathbf{0}^T & 0 \\ \mathbf{0} & L^{(i)} & \mathbf{0} \\ 0 & \mathbf{v}^{(i)T} & 1 \end{pmatrix} \begin{pmatrix} 1 & \mathbf{w}^{(i)T} & 0 \\ \mathbf{0} & U^{(i)} & \mathbf{0} \\ 0 & \mathbf{0}^T & 1 \end{pmatrix} \begin{pmatrix} \alpha_1^{(i)} & \mathbf{0}^T & \gamma^{(i)} \\ \mathbf{z}^{(i)} & D^{(i)} & \mathbf{y}^{(i)} \\ \beta^{(i)} & \mathbf{0}^T & \alpha_2^{(i)} \end{pmatrix}, \quad (5.1)$$

where

$$L^{(i)}U^{(i)}\mathbf{z}^{(i)} = b_1^{(i)}\mathbf{e}_1, \qquad D^{(i)}\mathbf{w}^{(i)} = c_0^{(i)}\mathbf{e}_1,$$

$$U^{(i)}\mathbf{y}^{(i)} = c_{k-1}^{(i)}\mathbf{e}_{k-1}, \quad D^{(i)}\mathbf{v}^{(i)} = b_k^{(i)}\mathbf{e}_{k-1},$$

$$\alpha_1^{(i)} = -\mathbf{w}^{(i)T}\mathbf{z}^{(i)}, \qquad \alpha_2^{(i)} = a_k^{(i)} - \mathbf{v}^{(i)T}\mathbf{y}^{(i)},$$

$$\gamma^{(i)} = -\mathbf{w}^{(i)T}\mathbf{y}^{(i)}, \qquad \beta^{(i)} = -\mathbf{v}^{(i)T}\mathbf{z}^{(i)}.$$

Both the vectors $\mathbf{v}^{(i)}$ and $\mathbf{w}^{(i)}$ have only one nonzero entry, while $\mathbf{z}^{(i)}$ and $\mathbf{y}^{(i)}$ are fill-in vectors.

Concerning the stability of the factorization, the results are similar to those for the parallel LU factorization. The parallel solver corresponding to (5.1) is an optimized version of Wang's algorithm [12, 14, 16]; the scalar count of (parallelizable) operations is decreased from about $21n$ (see [16]) to about $17\ n$.

6. PARALLEL CYCLIC REDUCTION FACTORIZATION

Cyclic reduction is an interesting algorithm for the solution of linear tridiagonal systems on vector and parallel computers [7, 8, 10, 13]. However, its parallel implementation requires synchronization among the processors at each step of the reduction (see [10]).

To overcome this problem, in [2] we proposed a block variant of the algorithm. Our proposal is to apply the cyclic reduction factorization to each block $A^{(i)}$ in (2.9). This implies that communication is necessary only for solving the reduced system (2.7).

First, we recall some notions concerning the cyclic reduction factorization [3]. If we consider an odd-even permutation matrix P_1 of dimension $k-1$, it follows that

$$A^{(i)} = P_1^T \begin{pmatrix} C_1 & T_1 \\ S_1 & B_1 \end{pmatrix} P_1$$

(we omit the upper index to simplify the notation). C_1 and B_1 are diagonal matrices containing the odd and even diagonal entries of $A^{(i)}$, respectively; S_1 and T_1 are bidiagonal matrices with the off-diagonal entries on the even and odd rows of $A^{(i)}$. If C_1^{-1} exists, then we define

$$A^{(i)} = P_1^T L_1 D_1 U_1 P_1 = P_1^T \begin{pmatrix} I & \\ S_1 C_1^{-1} & I \end{pmatrix} \begin{pmatrix} I & \\ & A_1 \end{pmatrix} \begin{pmatrix} C_1 & T_1 \\ & I \end{pmatrix} P_1, \quad (6.1)$$

where $A_1 = B_1 - S_1 C_1^{-1} T_1$ is again tridiagonal (of dimension $\lfloor (k-1)/2 \rfloor$). We can again repeat the same operations for D_1 by considering the matrix

$$P_2 = \begin{pmatrix} I & \\ & Q_2 \end{pmatrix},$$

where Q_2 is the odd-even permutation matrix of order $\lfloor (k-1)/2 \rfloor$. The process stops when D_r ($r = \lceil \log_2 k \rceil$) is the identity matrix of order $k-1$.

Then, we extend the factorization (6.1) to the matrix $M^{(i)}$ by means of the following matrices:

$$\hat{P}_1 = \begin{pmatrix} 1 & & \\ & P_1 & \\ & & 1 \end{pmatrix},$$

$$\hat{L}_1 = \begin{pmatrix} 1 & \tilde{\mathbf{w}}_1^T & \mathbf{0}^T & 0 \\ \mathbf{0} & I & O & \mathbf{0} \\ \mathbf{0} & S_1 C_1^{-1} & I & \mathbf{0} \\ 0 & \tilde{\mathbf{v}}_1^T & \mathbf{0}^T & 1 \end{pmatrix}, \qquad \hat{U}_1 = \begin{pmatrix} 1 & \mathbf{0}^T & \mathbf{0}^T & 0 \\ \tilde{\mathbf{z}}_1 & C_1 & T_1 & \tilde{\mathbf{y}}_1 \\ \mathbf{0} & O & I & \mathbf{0} \\ 0 & \mathbf{0}^T & \mathbf{0}^T & 1 \end{pmatrix},$$

and

$$\hat{D}_1 = \begin{pmatrix} \hat{\alpha}_1 & \mathbf{0}^T & \hat{\mathbf{w}}_1^T & \hat{\gamma} \\ \mathbf{0} & I & O & \mathbf{0} \\ \hat{\mathbf{z}}_1 & O & A_1 & \hat{\mathbf{y}}_1 \\ \hat{\beta} & \mathbf{0}^T & \hat{\mathbf{v}}_1^T & \hat{\alpha}_2 \end{pmatrix}. \tag{6.2}$$

By defining the vector $\mathbf{w}_1^T = (\tilde{\mathbf{w}}_1^T \ \hat{\mathbf{w}}_1^T)$, and similarly the vectors $\mathbf{v}_1$, $\mathbf{z}_1$, and $\mathbf{y}_1$, we obtain [see (6.1)]

$$\begin{aligned} U_1^T \mathbf{w}_1 &= c_0^{(i)} \mathbf{e}_1, & U_1^T \mathbf{v}_1 &= b_k^{(i)} P_1^T \mathbf{e}_{k-1}, \\ L_1 \mathbf{z}_1 &= b_1^{(i)} \mathbf{e}_1, & L_1 \mathbf{y}_1 &= c_{k-1}^{(i)} P_1^T \mathbf{e}_{k-1}, \\ \hat{\alpha}_1 &= -\tilde{\mathbf{w}}_1^T \tilde{\mathbf{z}}_1, & \hat{\alpha}_2 &= a_k^{(i)} - \tilde{\mathbf{v}}_1^T \tilde{\mathbf{y}}_1, \\ \hat{\beta} &= -\tilde{\mathbf{v}}_1^T \tilde{\mathbf{z}}_1, & \hat{\gamma} &= -\tilde{\mathbf{w}}_1^T \tilde{\mathbf{y}}_1. \end{aligned}$$

It results that all the vectors $\tilde{\mathbf{w}}_1$, $\tilde{\mathbf{z}}_1$, $\tilde{\mathbf{v}}_1$, $\tilde{\mathbf{y}}_1$, $\hat{\mathbf{w}}_1$, $\hat{\mathbf{z}}_1$, $\hat{\mathbf{v}}_1$, $\hat{\mathbf{y}}_1$ have, at most, one nonzero entry. The absence of fill-in vectors implies that the corresponding parallel solver has a minimum memory requirement. In fact, only four vectors of length k per processor are needed, while six vectors are necessary for the parallel LU and the parallel $LU\,D$ factorization algorithms.

We observe that $\hat{\beta} = 0$ and $\hat{\gamma} = 0$ if $r > 1$; moreover, if one removes the blocks on the second row and second column of $\hat{D}_1$, the resulting matrix is still tridiagonal.

The structure of the matrices $\hat{D}_i$ (for $i = 2, \ldots, r-1$) is similar to (6.2). At the rth step, we have

$$\hat{D}_r = \begin{pmatrix} \alpha_1^{(i)} & \mathbf{0}^T & \gamma^{(i)} \\ \mathbf{0} & I_{k-1} & \mathbf{0} \\ \beta^{(i)} & \mathbf{0}^T & \alpha_2^{(i)} \end{pmatrix},$$

where $\alpha_1^{(i)}$, $\alpha_2^{(i)}$, $\beta^{(i)}$, $\gamma^{(i)}$ are the scalars defined in (2.10).

The corresponding parallel solver, which we call the *parallel cyclic reduction algorithm* is stable when A is strongly diagonally dominant or weakly diagonally dominant and irreducible (see [3, 7]). Moreover, its scalar count of (parallelizable) operations is of about $17n$.

Finally, we observe that, as the cyclic reduction algorithm is vectorizable, the parallel cyclic reduction algorithm can be efficiently implemented on a parallel computer with vector facilities. This is not true for the algorithms examined in the previous sections.

7. THE PARALLEL CYCLIC REDUCTION ALGORITHM

The algorithms deriving from the twisted factorization, the parallel LU factorization, and the parallel $LU\ D$ factorization are (if not already known) very straightforward to derive. This is not true for the parallel cyclic reduction algorithm, derived from the parallel cyclic reduction factorization, which is not so immediate (moreover, it will be used later in Section 9). Therefore, we shall describe it by using a programming-like language. We shall assume that processor i is involved with the corresponding block $M^{(i)}$ in (2.9) for $i = 0, \ldots, p-1$. The vectors $a(0:k)$, $b(1:k)$, $c(0:k-1)$ contain the three diagonals of $M^{(i)}$, while the vector $x = x(0:k)$ is initialized with $x(0) = 0$ and $x(1:k) = (f_{ik+1} \cdots f_{(i+1)k})^T$; in output $x(1:k)$ will contain $(x_{ik+1} \cdots x_{(i+1)k})^T$ [see (2.1) and (2.2)]; it is obvious that f_{pk} and x_{pk} are not considered when $i = p-1$:

```
%
% Parallel cyclic reduction algorithm
%
begin procedure on processor i
%
% Initialization
%
< input of data >
%
  r := 0
  s := 1
  neq := k - 1
  x(0) := 0
  do while ( neq > 0 )
    r := r + 1
    flag := 1 - mod(neq,2)
    neq := int(neq/2)
    n1 := s
    n2 := n1 + s
    n3 := n2 + s
    if ( i > 0 )                                          (*)
      c(0) := -c(0)/a(n1)
      a(0) := a(0) + c(0)*b(n1)
      x(0) := x(0) + c(0)*x(n1)
      c(0) := c(0)*c(n1)
    end if
    do m := 1 to (neq - flag)
```

```
        b(n2) := -b(n2)/a(n1)
        c(n2) := -c(n2)/a(n3)
        a(n2) := a(n2) + c(n1)*b(n2) + c(n2)*b(n3)
        x(n2) := x(n2) + x(n1)*b(n2) + c(n2)*x(n3)
        b(n2) := b(n1)*b(n2)
        c(n2) := c(n2)*c(n3)
        n1 := n3
        n2 := n1 + s
        n3 := n2 + s
      end do
      if ( flag = 0 )
        n2 := k
      end if
      if ( (p-1-i+flag) > 0 )                                    (*)
        b(n2) := -b(n2)/a(n1)
        a(n2) := a(n2) + c(n1)*b(n2)
        x(n2) := x(n2) + x(n1)*b(n2)
        b(n2) := b(n1)*b(n2)
      end if
      s := s*2
    end do
  %
  % The information concerning the reduced system is in:
  %  a(0)  c(0)  x(0)
  %  b(k)  a(k)  x(k)
  %
  < solution of the reduced system >
  %
    do j := 1 to r
      s := s/2
      n1 := 0
      n2 := n1 + s
      n3 := n2 + s
      do m := 1 to neq
        x(n2) := ( x(n2) -x(n1)*b(n2) - c(n2)*x(n3) )/a(n2)
        n1 := n3
        n2 := n1 + s
        n3 := n2 + s
      end do
      neq := neq*2
      if (n2 < k )
        x(n2) := ( x(n2) - x(n1)*b(n2) - c(n2)*x(k) )/a(n2)
```

```
        neq := neq + 1
      end if
    end do
  %
  end procedure
```

8. NUMERICAL TESTS

The parallel methods examined in the previous sections are here compared with the scalar *LU* algorithm for solving the linear system (2.1). The twisted factorization is neglected, because it is an algorithm for two processors only.

The algorithms examined in Sections 4, 5, and 6 have been implemented in Parallel Fortran [17] with the Express communication library [18] on a MicroWay Multiputer, which has a network of 32 transputers T800-20, each one with a local memory of 1 Mb. We have not considered the implementation of other parallel tridiagonal solvers, because of their higher cost, either in the number of parallelizable operations (see Sections 4 and 5), or in communication overhead (see Section 6).

For all the parallel methods, the topology of interconnection among the processors is a pipeline, since the reduced system is solved with a *UL* factorization algorithm. If the parallel solution of the reduced system is needed, then a hypercube configuration is more suitable, since cyclic reduction is the algorithm of choice [6]. The scalar *LU* algorithm has been implemented in Fortran on a single T800-20 with 16 Mb of memory.

The speedup (with respect to the dimension of the problem) obtained on 4, 8, 16, and 32 processors is outlined in Figure 1; the solid line is for the parallel *LU* algorithm, the dashed line is for the parallel cyclic reduction algorithm, and the dotted line is for the parallel *LU D* algorithm.

9. MODIFIED PARALLEL CYCLIC REDUCTION ALGORITHM

If we have a small number of parallel processors, then it might be useful to have an algorithm with slightly better performance than those examined in the previous section. From Figure 1 it seems that the parallel cyclic reduction algorithm proposed in Sections 6 and 7 is the best one. Nevertheless, an improvement is possible if the blocks $A^{(0)}$ and $A^{(p-1)}$ in (2.5.2) are factorized

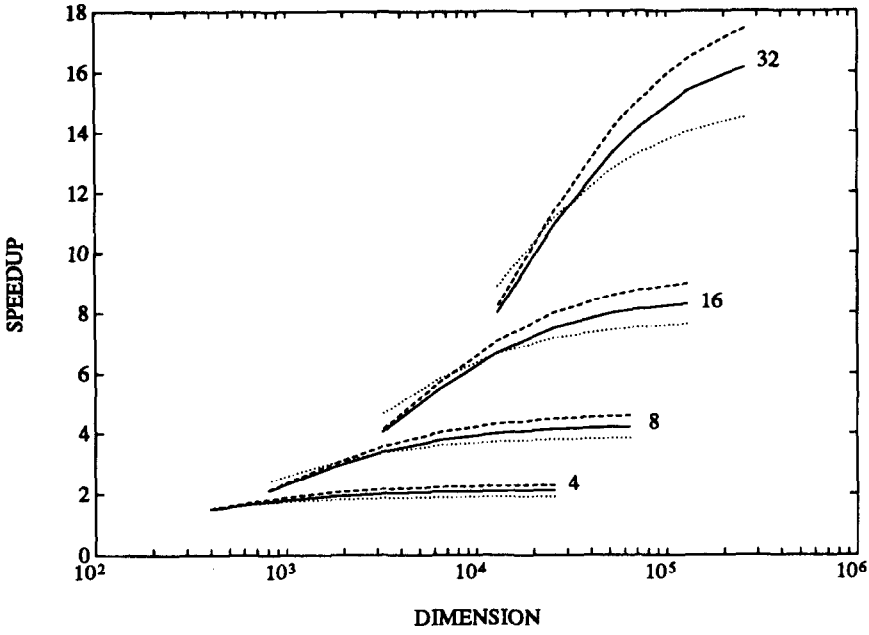

FIG. 1. Measured speedup.

as LU and UL, respectively, and cyclic reduction is used for the blocks $A^{(i)}$, $i = 1, \ldots, p-2$. Concerning what was stated in Section 3 about the twisted factorization, it is implied that if the data are equally distributed among the processors, the first and the last processors perform a smaller number of scalar operations than the intermediate ones. It follows that the computational load can be redistributed in order to have all the processors performing the same number of operations. This results in better performance of the whole algorithm. The improvement is not so evident if we have many parallel processors, but it is significant when p is small.

The final algorithm is constituted by that described in Section 7 running on processor i, $i = 1, 2, \ldots, p-2$ (the two controls with the mark (*) are no longer necessary in this case), while the algorithms on processors 0 and $p-1$ are listed below:

```
%
% Modified parallel cyclic reduction algorithm
%
begin procedure on processor 0
%
```

```
% Initialization
%
< input of data >
%
  do m := 2 to k_0
    b(m) := -b(m)/a(m - 1)
    a(m) := a(m) + b(m)*c(m - 1)
    x(m) := x(m) + b(m)*x(m - 1)
  end do
%
% The information concerning the reduced system is in:
%  a(k_0)  x(k_0)
%
< solution of the reduced system >
%
  do m := k_0 - 1 to 1 step -1
    x(m) := ( x(m) - c(m)*x(m + 1) )/a(m)
  end do
%
end procedure

%
% Modified parallel cyclic reduction algorithm
%
begin procedure on processor p - 1
%
% Initialization
%
< input of data >
%
  do m := k_{p-1} - 2 to 0 step -1
    c(m) := -c(m)/a(m + 1)
    a(m) := a(m) + c(m)*b(m + 1)
    x(m) := x(m) + c(m)*x(m + 1)
  end do
%
% The information concerning the reduced system is in:
%  a(0)  x(0)
%
< solution of the reduced system >
%
  do m := 1 to k_{p-1} - 1
```

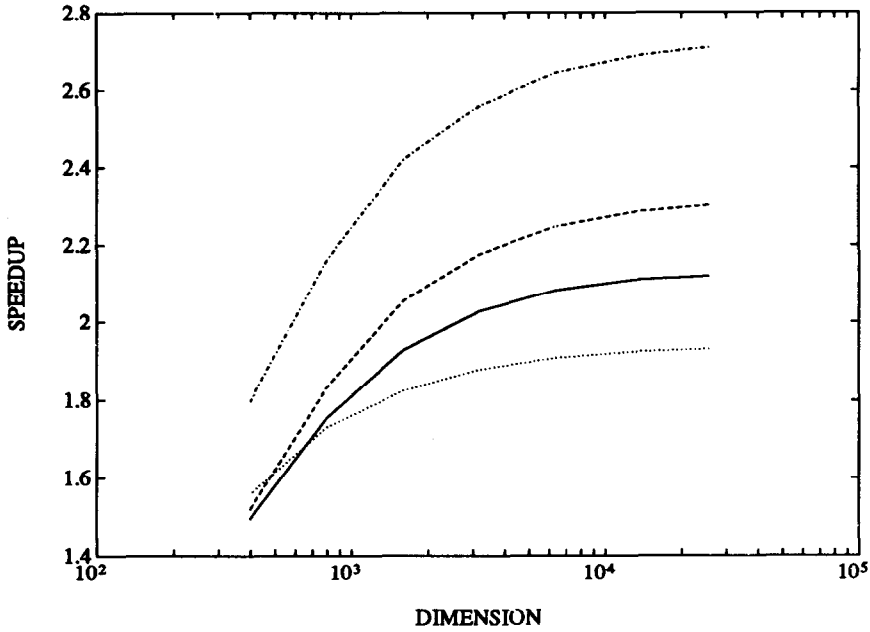

FIG. 2. Speedup on four processors.

```
    x(m) := ( x(m) - b(m)*x(m - 1) )/a(m)
  end do
%
end procedure
```

In Figure 2 ($p = 4$) the algorithms of Figure 1 are compared with the modified version of the parallel cyclic reduction algorithm (dash-dotted line).

We express our thanks to Mrs. Paulene Butts for her help in the preparation of the manuscript.

REFERENCES

1 P. Amodio, Optimized cyclic reduction for the solution of linear tridiagonal systems on parallel computers, submitted for publication.

2 P. Amodio, L. Brugnano, and T. Politi, Parallel factorizations for tridiagonal matrices, submitted for publication.

3 P. Amodio and F. Mazzia, Stability of the cyclic reduction for the solution of tridiagonal systems, submitted for publication.
4 I. Babuska, Numerical stability in problems of linear algebra, *SIAM J. Numer. Anal.* 9:53–77 (1972).
5 L. Brugnano, A parallel solver for tridiagonal linear systems for distributed memory parallel computers, *Parallel Comput.* 17:1017–1023 (1991).
6 I. N. Hajj and S. Skelboe, A multilevel parallel solver for block tridiagonal and banded linear systems, *Parallel Comput.* 15:21–45 (1990).
7 D. Heller, Some aspects of the cyclic reduction algorithm for block tridiagonal linear systems, *SIAM J. Numer. Anal.* 13:484–496 (1976).
8 R. W. Hockney and C. R. Jesshope, *Parallel Computers*, Adam Hilger, Bristol, 1981.
9 S. L. Johnsson, Solving narrow banded systems on ensemble architectures, *ACM Trans. Math. Software* 11:271–288 (1985).
10 S. L. Johnsson, Solving tridiagonal systems on ensemble architectures, *SIAM J. Sci. Statist. Comput.* 8:354–392 (1987).
11 D. Kershaw, Solution of single tridiagonal linear systems and vectorization of the ICCG algorithm on the Cray 1, in *Parallel Computations* (G. Rodrigue, Ed.), Academic, New York, 1982, pp. 85–99.
12 P. H. Michielse and H. A. Van Der Vorst, Data transport in Wang's partition methods, *Parallel Comput.* 7:87–95 (1988).
13 J. M. Ortega, *Introduction to Parallel and Vector Solution of Linear Systems*, Plenum, New York, 1988.
14 H. A. Van Der Vorst, Large tridiagonal and block tridiagonal linear systems on vector and parallel computers, *Parallel Comput.* 5:45–54 (1987).
15 H. A. Van Der Vorst, Analysis of a parallel solution method for tridiagonal linear systems, *Parallel Comput.* 5:303–311 (1987).
16 H. H. Wang, A parallel method for tridiagonal equations, *ACM Trans. Math. Software* 7:170–183 (1981).
17 *Parallel Fortran User Guide*, 3L Ltd., 1988.
18 *Express: A Communication Environment for Parallel Computers*, ParaSoft Corp., 1988.

Received 18 July 1991; final manuscript accepted 10 February 1992