ELSEVIER

Parallel Computing 21 (1995) 1097–1110

PARALLEL COMPUTING

The parallel QR factorization algorithm for tridiagonal linear systems

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

[a] *Dipartimento di Matematica, Università di Bari, Via Orabona 4, 70125 Bari, Italy*
[b] *Dipartimento di Energetica, Università di Firenze, Via Lombroso 6/17, 50134 Firenze, Italy*

Received 11 February 1994; revised 3 December 1994

Abstract

We describe a new parallel solver in the class of partition methods for general, nonsingular tridiagonal linear systems. Starting from an already known partitioning of the coefficient matrix among the parallel processors, we define a factorization, based on the QR factorization, which depends on the conditioning of the sub-blocks in each processor. Moreover, also the reduced system, whose solution is the only scalar section of the algorithm, has a dimension which depends both on the conditioning of these sub-blocks, and on the number of processors. We analyze the stability properties of the obtained parallel factorization, and report some numerical tests carried out on a net of transputers.

Keywords: Tridiagonal linear systems; Parallel solvers; QR factorization; Transputer array

1. Introduction

Much interest has been devoted to the problem of solving tridiagonal linear systems on parallel computers in recent years, and many parallel tridiagonal solvers have been devised. These solvers can be grouped into two main categories: direct solvers and iterative solvers. Among the direct methods, we can mention the partition methods, which include very efficient parallel algorithms [1,2,4,5,10–15]. Among the iterative methods we mention the Jacobi method and the AGE method [9].

Concerning partition methods, a unifying approach for their derivation and study is given in [1,2], where it is also shown that these methods are stable when

* Corresponding author. Email: 00110570@vm.csata.it

SSDI 0167-8191(95)00011-9

used to solve diagonally dominant linear systems. Diagonal dominance also implies the convergence of the above mentioned iterative solvers.

Nevertheless, to our knowledge, no parallel algorithms exist for solving general nonsingular tridiagonal linear systems. In this paper, we develop a new parallel solver for this purpose. The derivation of the method is outlined in Sections 2 and 3, while its stability properties are discussed in Section 4. In Section 5 the parallel algorithm is examined and a simple model for the expected speedup is reported together with some numerical tests carried out on a linear array of transputers.

2. Parallel factorizations

The derivation of the new method extends the concept of 'parallel factorization' given in [1]. Let us consider the tridiagonal matrix

$$A = \begin{pmatrix} a_1 & c_1 & & \\ b_1 & a_2 & \ddots & \\ & \ddots & \ddots & c_{n-1} \\ & & b_{n-1} & a_n \end{pmatrix}, \tag{1}$$

where for simplicity we assume that $n = kp - 1$ for some integer k, if p is the number of the parallel processors we want to use. To obtain a parallel solver tailored for this number of processors, we consider the matrix A partitioned as follows:

$$A = \begin{pmatrix} A^{(1)} & c_{k-1}^{(1)} e_{k-1} & & & & & \\ b_{k-1}^{(1)} e_{k-1}^T & a_k^{(1)} & c_0^{(2)} e_1^T & & & & \\ & b_0^{(2)} e_1 & A^{(2)} & c_{k-1}^{(2)} e_{k-1} & & & \\ & & b_{k-1}^{(2)} e_{k-1}^T & a_k^{(2)} & & & \\ & & & & \ddots & & \\ & & & & & a_k^{(p-1)} & c_0^{(p)} e_1^T \\ & & & & & b_0^{(p)} e_1 & A^{(p)} \end{pmatrix}, \tag{2}$$

where e_1 and e_{k-1} are, respectively, the first and the last unit vectors of $\mathbb{R}^{k-1}$, the blocks $A^{(i)}$ are tridiagonal of size $k-1$ and $x_j^{(i)} = x_{(i-1)k+j}$ for $x = a, b, c$. Observe that on a distributed memory parallel computer, the upper index identifies the processor where the given quantity is to be stored. Let us then consider the following factorization of the matrix (1):

$$A = FTG, \tag{3}$$

where, by denoting with I_{k-1} the identity matrix of order $k-1$,

$$F = \begin{pmatrix} N^{(1)} & \mathbf{0} & & & & & & \\ v^{(1)T} & 1 & w^{(2)T} & 0 & & & & \\ & \mathbf{0} & N^{(2)} & \mathbf{0} & & & & \\ & \mathbf{0} & v^{(2)T} & 1 & w^{(3)T} & 0 & & \\ & & & \mathbf{0} & N^{(3)} & \mathbf{0} & & \\ & & & 0 & v^{(3)T} & 1 & & \\ & & & & & & 1 & w^{(p)T} \\ & & & & & & \mathbf{0} & N^{(p)} \end{pmatrix},$$

$$T = \begin{pmatrix} I_{k-1} & \mathbf{0} & & & & & & \\ \mathbf{0}^T & \alpha^{(1)} & \mathbf{0}^T & \gamma^{(2)} & & & & \\ & \mathbf{0} & I_{k-1} & \mathbf{0} & & & & \\ & \beta^{(2)} & \mathbf{0}^T & \alpha^{(2)} & \mathbf{0}^T & \gamma^{(3)} & & \\ & & & \mathbf{0} & I_{k-1} & \mathbf{0} & & \\ & & & \beta^{(3)} & \mathbf{0}^T & \alpha^{(3)} & & \\ & & & & & & \ddots & \\ & & & & & & \alpha^{(p-1)} & \mathbf{0}^T \\ & & & & & & \mathbf{0} & I_{k-1} \end{pmatrix},$$

$$G = \begin{pmatrix} S^{(1)} & y^{(1)} & & & & & & \\ \mathbf{0}^T & 1 & \mathbf{0}^T & 0 & & & & \\ & z^{(2)} & S^{(2)} & y^{(2)} & & & & \\ & 0 & \mathbf{0}^T & 1 & \mathbf{0}^T & 0 & & \\ & & & z^{(3)} & S^{(3)} & y^{(3)} & & \\ & & & 0 & \mathbf{0}^T & 1 & & \\ & & & & & & \ddots & \\ & & & & & & 1 & \mathbf{0}^T \\ & & & & & & z^{(p)} & S^{(p)} \end{pmatrix},$$

$N^{(i)}S^{(i)}$ is a suitable factorization of the block $A^{(i)}$, and the other unknowns of F, G and T may be easily obtained from (3) by direct identification. By considering different factorizations for $A^{(i)}$, it is then possible to derive most of the parallel methods in the class of the partition methods. For example in [1,2], supposing $A^{(i)}$ diagonally dominant, the LU factorization, the LUD factorization, and the cyclic reduction were considered.

In the solution of the linear system

$$Ax = f, \tag{4}$$

the two block tridiagonal systems with the matrices F and G are solved without any synchronization or data communication among the processors. Therefore, this section is completely parallel. The only sequential part of the algorithm consists in the solution of the tridiagonal linear system with the 'reduced matrix':

$$T_p = \begin{pmatrix} \alpha^{(1)} & \gamma^{(2)} & & \\ \beta^{(2)} & \alpha^{(2)} & \ddots & \\ & \ddots & \ddots & \gamma^{(p-1)} \\ & & \beta^{(p-1)} & \alpha^{(p-1)} \end{pmatrix}_{(p-1)\times(p-1)} . \tag{5}$$

It must be noted that the size of T_p is independent of n, but depends only on the number p of parallel processors.

In [1,2] it is suggested to study the factorization (3) locally by introducing the $(n+p-1)\times n$ matrix R_p recursively defined as follows:

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

From $A = R_p^T M R_p$, we obtain the matrix M which is $p \times p$ block diagonal with tridiagonal main-diagonal blocks $M^{(i)}$ defined (with some slight differences for the first and the last one (see (2))) as follows:

$$M^{(i)} = \begin{pmatrix} 0 & c_0^{(i)} e_1^T & 0 \\ b_0^{(i)} e_1 & A^{(i)} & c_{k-1}^{(i)} e_{k-1} \\ 0 & b_{k-1}^{(i)} e_{k-1}^T & a_k^{(i)} \end{pmatrix}.$$

Then each $M^{(i)}$ may be factored in the form

$$M^{(i)} = \begin{pmatrix} 1 & w^{(i)T} & 0 \\ 0 & N^{(i)} & 0 \\ 0 & v^{(i)T} & 1 \end{pmatrix} \begin{pmatrix} \alpha_1^{(i)} & 0^T & \gamma^{(i)} \\ 0 & I_{k-1} & 0 \\ \beta^{(i)} & 0^T & \alpha_2^{(i)} \end{pmatrix} \begin{pmatrix} 1 & 0^T & 0 \\ z^{(i)} & S^{(i)} & y^{(i)} \\ 0 & 0^T & 1 \end{pmatrix}, \tag{6}$$

where the matrices $N^{(i)}$ and $S^{(i)}$, the vectors $v^{(i)}$, $w^{(i)}$, $y^{(i)}$, $z^{(i)}$ and the scalars $\beta^{(i)}$, $\gamma^{(i)}$ are the same as in (3), and $\alpha_2^{(i)} + \alpha_1^{(i+1)} = \alpha^{(i)}$, for $i = 1, \ldots, p-1$.

3. The parallel QR factorization

The approach in Section 2 is very general and useful, and very efficient tridiagonal solvers have been obtained [1]. Nevertheless, it is evident that all the blocks $A^{(i)}$ need to be nonsingular for (3) to be defined. This is the case, for example, when the matrix A is diagonally dominant. In this case, also the reduced matrix turns out to be diagonally dominant [1,2].

For A generally nonsingular, it may happen that some of the blocks $A^{(i)}$ are singular or very ill-conditioned. It follows that the parallel solver may be not defined or unstable even if stable scalar solvers exist (for example, the LU factorization method with partial pivoting or the QR factorization method). For this reason, in order to obtain a stable parallel solver, we consider a generalization of the above approach which consists in a finer structuring of the factorization itself.

Suppose that the $(k-1)\times(k-1)$ block $A^{(i)}$ is singular. Then we proceed as follows: let $A_1^{(i)}$ be the largest nonsingular principal submatrix of $A^{(i)}$, and let $j-1$ be its size. We define the following factorization of the block $M^{(i)}$:

$$M^{(i)} = L_1^{(i)} D_1^{(i)} U_1^{(i)}, \tag{7}$$

where

$$L_1^{(i)} = \begin{pmatrix} 1 & w^{(i)T} & 0 & & \\ \mathbf{0} & N_1^{(i)} & \mathbf{0} & & \\ 0 & v_1^{(i)T} & 1 & \mathbf{0}^T & 0 \\ & & \mathbf{0} & I_{k-j-1} & \mathbf{0} \\ & & 0 & \mathbf{0}^T & 1 \end{pmatrix}, \quad U_1^{(i)} = \begin{pmatrix} 1 & \mathbf{0}^T & 0 & & \\ z_1^{(i)} & S_1^{(i)} & y_1^{(i)} & & \\ 0 & \mathbf{0}^T & 1 & \mathbf{0}^T & 0 \\ & & \mathbf{0} & I_{k-j-1} & \mathbf{0} \\ & & 0 & \mathbf{0}^T & 1 \end{pmatrix},$$

$$D_1^{(i)} = \begin{pmatrix} \hat{\alpha}_1^{(i)} & \mathbf{0}^T & \gamma_1^{(i)} & & \\ \mathbf{0} & I_{j-1} & \mathbf{0} & & \\ \beta_1^{(i)} & \mathbf{0}^T & \hat{\alpha}_2^{(i)} & c_j^{(i)}\tilde{\mathbf{e}}_1^T & \\ & & b_j^{(i)}\tilde{\mathbf{e}}_1 & A_2^{(i)} & c_{k-1}^{(i)}\tilde{\mathbf{e}}_{k-j-1} \\ & & & b_{k-1}^{(i)}\tilde{\mathbf{e}}_{k-j-1}^T & a_k^{(i)} \end{pmatrix},$$

and $\tilde{e}_i$ is the ith unit vector of size $k-j-1$. This is equivalent to introducing the variable $x_j^{(i)} = x_{(i-1)k+j}$ in the reduced system. The same procedure is then recursively applied to the tridiagonal block $A_2^{(i)}$ of $D_1^{(i)}$, and so on.

Therefore it follows that the factorization is always defined, eventually by increasing the size of the reduced system, that is, the scalar section of the corresponding parallel solver. We observe that some of the blocks $A_r^{(i)}$ may have zero size; this means that two consecutive components in the solution vector belong to the reduced system. Anyway, if the matrix A is nonsingular, then the block $A^{(i)}$, which has size $k-1$, has rank at most equal to $k-3$. It follows that in exact arithmetic the size of the reduced system is still independent of the size n of the linear system.

Since the blocks $A_r^{(i)}$ are only nonsingular, it is convenient to factorize them by using a suitable stable factorization. Two obvious candidates for this purpose are the LU factorization with partial pivoting and the QR factorization. We shall analyze in more detail the parallel solver obtained by using the second factorization, which we shall call *parallel QR factorization method*, but a similar approach can be repeated for the former. For simplicity, let us suppose that the generic

block $A^{(i)}$ is nonsingular, since the generalization is straightforward. Then, we factorize $M^{(i)}$ as:

$$M^{(i)} = \begin{pmatrix} 1 & w^{(i)T} & 0 \\ 0 & Q^{(i)} & 0 \\ 0 & v^{(i)T} & 1 \end{pmatrix} \begin{pmatrix} \alpha_1^{(i)} & 0^T & \gamma^{(i)} \\ 0 & I_{k-1} & 0 \\ \beta^{(i)} & 0^T & \alpha_2^{(i)} \end{pmatrix} \begin{pmatrix} 1 & 0^T & 0 \\ z^{(i)} & R^{(i)} & y^{(i)} \\ 0 & 0^T & 1 \end{pmatrix}, \tag{8}$$

where $Q^{(i)}R^{(i)}$ is the QR factorization of the matrix $A^{(i)}$, and

$$\begin{aligned} &R^{(i)T}w^{(i)} = c_0^{(i)}e_1, \quad R^{(i)T}v^{(i)} = b_{k-1}^{(i)}e_{k-1}, \\ &Q^{(i)}z^{(i)} = b_0^{(i)}e_1, \quad Q^{(i)}y^{(i)} = c_{k-1}^{(i)}e_{k-1}, \\ &\alpha_1^{(i)} = -w^{(i)T}z^{(i)}, \quad \alpha_2^{(i)} = a_k^{(i)} - v^{(i)T}y^{(i)}, \\ &\beta^{(i)} = -v^{(i)T}z^{(i)}, \quad \gamma^{(i)} = -w^{(i)T}y^{(i)}. \end{aligned} \tag{9}$$

We observe that, since $Q^{(i)}$ is upper Hessemberg and $R^{(i)}$ is upper triangular, then only the vectors $w^{(i)}$ and $z^{(i)}$ are full vectors (*fill-in vectors*), while only the last entry of $v^{(i)}$ and the last two entries of $y^{(i)}$ may be nonzero.

To increase the stability of the factorization, we may also choose $A_1^{(i)}$ as the largest principal submatrix of $A^{(i)}$ which has a condition number smaller than a given tolerance. This choice is possible since we can monitor a very effective approximation of the condition number of the current block. In fact, the matrices $Q^{(i)}$ and $R^{(i)}$, as well as the vector $w^{(i)}$ can be formed step by step. In particular, at the generic step j we have calculated $Q_j^{(i)}$ and $R_j^{(i)}$ which are the principal submatrices of size j of $Q^{(i)}$ and $R^{(i)}$, and the subvector $w_j^{(i)}$ which contains the first j components of the vector $w^{(i)}$. Since $Q_j^{(i)}$ is orthogonal, then the condition number of $A_j^{(i)}$ is the same as the condition number of $R_j^{(i)}$. From (9) it follows that $\eta_j^{(i)} = (c_0^{(i)})^{-1} \| w_j^{(i)} \|_1 \approx \| (R_j^{(i)})^{-1} \|_\infty$, since $\eta_j^{(i)}$ is the norm of the first row of $(R_j^{(i)})^{-1}$. Therefore, we may obtain an estimate of the condition number of $A_j^{(i)}$ at each step. As soon as this estimate is larger than a given tolerance, we increase the size of the reduced system, as previously seen, by introducing in it the variable $x_j^{(i)}$.

3.1. The parallel LU factorization with partial pivoting

To complete this discussion, we now sketch briefly the *parallel LU factorization with partial pivoting*, which is obtained by considering the following factorization of $M^{(i)}$ (again, we shall suppose that $A^{(i)}$ is nonsingular, for simplicity):

$$M^{(i)} = \begin{pmatrix} 1 & w^{(i)T} & 0 \\ 0 & P^{(i)T}L^{(i)} & 0 \\ 0 & v^{(i)T} & 1 \end{pmatrix} \begin{pmatrix} \alpha_1^{(i)} & 0^T & \gamma^{(i)} \\ 0 & I_{k-1} & 0 \\ \beta^{(i)} & 0^T & \alpha_2^{(i)} \end{pmatrix} \begin{pmatrix} 1 & 0^T & 0 \\ z^{(i)} & U^{(i)} & y^{(i)} \\ 0 & 0^T & 1 \end{pmatrix}. \tag{10}$$

Here we factorize $A^{(i)}$ as $P^{(i)}A^{(i)} = L^{(i)}U^{(i)}$ in this case, one has

$$\kappa(A^{(i)}) \le \kappa(L^{(i)})\kappa(U^{(i)}).$$

To estimate the quantities on the right hand side one has, as for the parallel QR factorization,

$$\|(U^{(i)})^{-1}\|_\infty \approx (c_0^{(i)})^{-1}\|w^{(i)}\|_1,$$

while for $\|(L^{(i)})^{-1}\|$ one of the following estimates can be used ($k-1$ is the size of $L^{(i)}$):

$$\|(L^{(i)})^{-1}\| \le 2(k-1), \quad \|(L^{(i)})^{-1}\|_1 \approx (b_0^{(i)})^{-1}\|z^{(i)}\|_1.$$

The actual implementation of this algorithm will be examined in more detail elsewhere.

4. Stability analysis

Let us now examine the stability of the parallel QR factorization. By fixing an upper bound for the conditioning of the sub-blocks $A_r^{(i)}$, the conditioning of the whole system will depend on the reduced system. We shall therefore discuss how perturbations on the matrix A affect the reduced matrix.

For this purpose, let us suppose to have fixed a suitable tolerance σ for the condition number $\kappa(A_r^{(i)})$ of $A_r^{(i)}$. For a given nonsingular tridiagonal matrix A, a corresponding partition is then obtained. This partition can be represented in compact form by introducing the following block odd-even permutation matrix of order n:

$$P = \begin{pmatrix} I & \mathbf{0} & O & \dots & & & & \\ O & \mathbf{0} & I & \mathbf{0} & O & \dots & & \\ O & \mathbf{0} & O & \mathbf{0} & I & \mathbf{0} & O & \dots \\ \dots & \dots & \dots & & & & & \\ \mathbf{0}^T & 1 & \mathbf{0}^T & \dots & & & & \\ \mathbf{0}^T & 0 & \mathbf{0}^T & 1 & \mathbf{0}^T & \dots & & \\ \mathbf{0}^T & 0 & \mathbf{0}^T & 0 & \mathbf{0}^T & 1 & \mathbf{0}^T & \dots \\ \dots & \dots & \dots & & & & & \end{pmatrix},$$

which first takes the rows corresponding to the blocks $A_r^{(i)}$ and then those associated to the reduced system. Here we have denoted by I a generic identity matrix whose size is equal to the size of the corresponding block $A_r^{(i)}$ in the partition. By means of P we obtain

$$PAP^T = \begin{pmatrix} A_1 & C \\ B & A_2 \end{pmatrix}, \tag{11}$$

where A_1 and A_2 are block diagonal, A_1 having $A_r^{(i)}$ as a generic diagonal block, and A_2 of size $\ge p-1$, while B and C are block bidiagonal. The parallel QR factorization is equivalent to factoring the matrix (11) as

$$\begin{pmatrix} Q_1 & \\ BR_1^{-1} & I \end{pmatrix}\begin{pmatrix} R_1 & Q_1^T C \\ & T_p \end{pmatrix}, \tag{12}$$

where Q_1R_1 is the QR factorization of the matrix A_1 and

$$T_p = A_2 - BA_1^{-1}C \tag{13}$$

is the reduced matrix. Now we examine the propagation of small perturbations on the matrix A in the factorization (12). In the following, for every given matrix X the symbol δX will denote its corresponding perturbation (in particular δX^{-1} and δX^T will represent perturbations respectively on X^{-1} and X^T). Then, one has:

$$\begin{aligned} P(A+\delta A)P^T &= \begin{pmatrix} A_1+\delta A_1 & C+\delta C \\ B+\delta B & A_2+\delta A_2 \end{pmatrix} \\ &= \begin{pmatrix} Q_1+\delta Q_1 & \\ (B+\delta B)(R_1^{-1}+\delta R_1^{-1}) & I \end{pmatrix} \\ &\quad \begin{pmatrix} R_1+\delta R_1 & (Q_1+\delta Q_1)^T(C+\delta C) \\ & T_p+\delta T_p \end{pmatrix}. \end{aligned}$$

It is obvious that $\|\delta A_1\|, \|\delta A_2\|, \|\delta B\|, \|\delta C\| \le \|\delta A\|$. Moreover, since both Q_1 and $Q_1+\delta Q_1$ are orthogonal, it follows that $\|\delta Q_1\|$ is also bounded. Then, in the above expression we need only to provide an explicit bound for $\|\delta R_1\|, \|\delta R_1^{-1}\|$ and $\|\delta T_p\|$. By using the 2-norm and a first order analysis, from the relations

$$\begin{aligned} &Q_1R_1 = A_1, \\ &(Q_1+\delta Q_1)(R_1+\delta R_1) = A_1+\delta A_1, \\ &(R_1^{-1}+\delta R_1^{-1})(Q_1^T+\delta Q_1^T) = A_1^{-1}+\delta A_1^{-1}, \end{aligned}$$

one easily derives that

$$\frac{\|\delta R_1\|}{\|R_1\|} \le \frac{\|\delta A_1\|}{\|A_1\|} + \frac{\|\delta Q_1\|}{\|Q_1\|},$$

and

$$\frac{\|\delta R_1^{-1}\|}{\|R_1^{-1}\|} \le \frac{\|\delta A_1^{-1}\|}{\|A_1^{-1}\|} + \frac{\|\delta Q_1\|}{\|Q_1\|}.$$

Similarly, from the perturbation of (13), it can be derived that

$$\begin{aligned} \|\delta T_p\| &\le \|\delta A_2\| + \|B\|\,\|A_1^{-1}\|\,\|C\| \left(\frac{\|\delta A_1^{-1}\|}{\|A_1^{-1}\|} + \frac{\|\delta B\|}{\|B\|} + \frac{\|\delta C\|}{\|C\|} \right) \\ &\approx \|A\| \left(\varepsilon + \kappa(A_1) \left(\frac{\|\delta A_1^{-1}\|}{\|A_1^{-1}\|} + 2\varepsilon \right) \right). \end{aligned}$$

Here we have supposed that the relative perturbations on the matrix A are smaller than ϵ. In the above expressions, the two quantities $\kappa(A_1)$ and $\|\delta A_1^{-1}\|\,\|A_1^{-1}\|^{-1}$ can be noted. Since we use a tolerance σ for $\kappa(A_r^{(i)})$, we have:

$$\kappa(A_1) = \max_{i,r} \{\kappa(A_r^{(i)})\} \le \sigma.$$

Concerning $\| \delta A_1^{-1} \| \, \| A_1^{-1} \|^{-1}$, if we suppose that $\| A_1^{-1} \| \, \| \delta A_1 \| \ll 1$, then from the relation

$$(A_1 + \delta A_1)^{-1} = A_1^{-1} + \delta A_1^{-1},$$

and by using standard results of linear algebra, one obtains:

$$\frac{\| \delta A_1^{-1} \|}{\| A_1^{-1} \|} \leq \frac{\kappa(A_1)}{1 - \| A_1^{-1} \| \, \| \delta A_1 \|} \frac{\| \delta A_1 \|}{\| A_1 \|} \approx \kappa(A_1) \frac{\| \delta A_1 \|}{\| A_1 \|}.$$

Therefore, also this term is proportional to $\kappa(A_1)$. We conclude that by fixing a suitable upper bound σ for $\kappa(A_1)$ (which may depend on the size n of the problem, on the number p of the processors used and on the machine precision u) the factorization (12) is quite stable, at least when $\kappa(A)$ itself is not exaggerately large. Indeed, the algorithm is essentially devoted to matrices which are not diagonally dominant; but nevertheless well-conditioned or weakly well-conditioned (see [6]). In fact, the case where the condition number of the matrix A grows exponentially with its size is of no practical interest, since the problem becomes too much ill-conditioned for relatively small values of n.

5. The parallel QR factorization algorithm

Let us now examine a pseudo-code which implements the parallel QR factorization algorithm; this will allow us to easily derive a simplified model for the expected speedup with respect to the scalar QR factorization algorithm. The QR factorization is obtained by using Givens' rotation matrices. The data distribution is carried out according to (2), where the upper index denotes the processor number. Moreover, in order to obtain a simpler code, we add two null equations at the beginning and at the end of the linear system which we call *rows 0* and $n + 1$, respectively. The pseudo-code, written by using a Matlab-like language, follows.

```
procedure pqr(k, a, b, c, x, tol, proc, nproc)
%
% k=size of the local block on the processor
% a, b, c, x=coefficients of the problem and of the rhs
%       assigned to the processor; in output x contains
%       the computed solution
% tol=tolerance for the norm of the inverse of the blocks
% proc=index of the current processor
% nproc=number of the parallel processors
i=0
nr=1
break(1)=0
while (i<k-1)
    call qr1(k-i, a(i), b(i), c(i), x(i), z(i), tol, j)
```

```
    i=i+j
    nr=nr+1
    break(nr)=i
end
if (i==k-1)
    nr=nr+1
    break(nr)=k
end
%
% the indexes of the variables in the reduced system
% are in the vector break(1:nr) if 1<proc<nproc,
% in the vector break(2:nr) on processor 1, and in the
% vector break(1:nr-1) on processor nproc. These
% indexes are used to recover the coefficients of the
% reduced system from the vectors a, b, c, x.
%
<solution of the reduced system>
%
% update of the rhs and back-substitution
for i=1:nr-1
    j0=break(i)
    j=break(i+1)-1
    if (j>j0)
      x(j)=(x(j)-z(j)*x(j0)-c(j)*x(j+1))/a(j)
      for j=break(i+1)-2:-1:break(i)+1
       x(j)=(x(j)-z(j)*x(j0)-c(j)*x(j+1)-b(j)*x(j+2))/a(j)
      end
    end
end
end procedure


procedure qr1(k, a, b, c, x, z, tol, j)
%
% Performs the QR factorization of the block defined by
% the coefficients a, b, c, updating the corresponding
% rhs x. The vector b is overwritten with the second upper
% diagonal of the matrix R; z is the fill-in vector.
% The informations concerning the reduced system are
% stored as follows:
% a(0)=α1,    c(0)=γ,
% b(0)=β,     a(j)=α2,
% while the last two components of the vector y are stored
% in b(j-2) ans c(j-1), respectively.
```

```
%
toosmall=<tolerance for zero entries>
w=-1      % the fill-in vector w is not actually formed;
w1=0      % the needed informations are stored in w, w1, w2
z(1)=b(0)
b(0)=0
norm=0
tol1=tol*c(0)
for i=1:k-2
    % computes the coefficients of the Givens matrix
    % for the vector (a(i), b(i))^T
    call givens (a(i), b(i), r, s)
    tmp=r*a(i+1)-s*c(i)
    if (abs(tmp)<toosmall & ...
      (abs(b(i+1))<toosmall|i==k-2) | norm>tol1)
      break
    end
    a(i)=r*a(i)+s*b(i)
    c(i)=r*c(i)+s*a(i+1)
    a(i+1)=tmp
    b(i)=s*c(i+1)
    c(i+1)=r*c(i+1)
    z(i+1)=-s*z(i)
    z(i)=r*z(i)
    tmp=r*x(i+1)-s*x(i)
    x(i)=r*x(i)+s*x(i+1)
    x(i+1)=tmp
    w2=w1
    w1=w
    w=-(w2*b(i-2)+w1*c(i-1))/a(i)
    norm=norm+abs(w)
    a(0)=a(0)-w*z(i)
    x(0)=x(0)-w*x(i)
end
j=i+1
w2=w1
w1=w
w=-(w2*b(i-2)+w1*c(i-1))/a(i)
a(0)=a(0)-w*z(i)
x(0)=x(0)-w*x(i)
c(0)=-(w1*b(i-1)+w*c(i))
tmp=-b(i)/a(i)
b(0)=tmp*z(i)
a(j)=a(j)+tmp*c(i)
```

```
x(j)=x(j)+tmp*x(i)
end procedure
```

As one can see, the required storage on each processor essentially amounts to five vectors for the coefficients (a, b and c), the right hand side x, and the fill-in vector z.

Concerning the operation count, it is well known that a call to the procedure givens has a cost of 5 *flops* plus 1 square root (we count as 1 flop one of the four elementary diadic operations). From the above pseudo-code, it is simple to derive that, apart from the solution of the reduced system, the parallel solver requires $\approx 40n$ flops plus n square roots that can be performed in parallel on p processors. The cost of the solution of the reduced system can not be exactly predicted since it depends on the size of the system itself. However, since we observed that the blocks $A^{(i)}$ may have a null space of dimension at most two if the matrix A is nonsingular, one may expect that the size of the reduced system grows linearly with the number p of the processors, thus requiring $O(p)$ flops for its solution. This is true, at least when $\kappa(A)$ itself is not prohibitively large. In fact, if for example $\kappa(A)$ is $O(n)$, then the condition number of the largest principal nonsingular matrix of each block $A^{(i)}$ should behave approximately as $O(n/p)$ and the size of the reduced system can be expected to be at most $3p$.

As a further remark, in [3] it is suggested an approach for solving the reduced system that, for the matrix (5), requires only $O(\log_2 p)$ scalar operations and $\log_2 p - 1$ data transmissions. By applying the same approach (but by using the QR factorization) to the reduced system deriving from the parallel QR factorization described in Section 3, it results that the number of transmissions is unchanged even if the total number of data transmitted may be larger. Therefore, if we suppose $n \gg p$, then the actual scalar cost of this part of the algorithm is negligible. This approach has been used in the parallel code for the numerical tests.

The scalar Givens method requires $\approx 27n$ flops plus n square roots. It follows that the expected asymptotic speedup on p processors is given by

$$\hat{s}_p = \lim_{n\to\infty} s_p(n) = \lim_{n\to\infty} \frac{(27+\eta)n}{(40+\eta)\dfrac{n}{p} + O(p)} = \frac{27+\eta}{40+\eta}p, \tag{14}$$

where η is the ratio between the time to execute 1 square root and the time to execute 1 flop, and $O(p)$ is the scalar section of the code. Finally, it must be said that (14) is only an upper bound for the effective asymptotic speedup, since the operations in the scalar Givens algorithm are better chained and, hence, better optimized by any 'smart' compiler.

6. Numerical tests

Let us now report the measured speedup for the parallel QR factorization algorithm obtained on a linear array of transputers with distributed memory. For

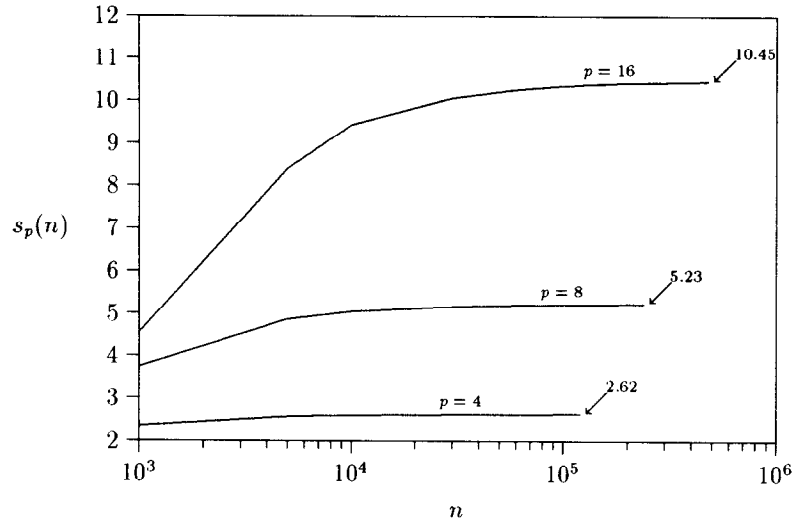

Fig. 1. Measured speedups.

this computer, the value η in (14) is about 3 and hence for n large with respect to p the expected speedup is $\hat{s}_p = 30p/43$. The programming language used is Fortran with the Express [16] parallel libraries. The numerical tests concern the following problem:

$$\begin{pmatrix} 0 & 1 & & \\ -1 & \ddots & \ddots & \\ & \ddots & 0 & 1 \\ & & -1 & 1 \end{pmatrix}_{n\times n} \qquad x = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \tag{15}$$

which derives from the use of the mid-point method as boundary value method for ODE [7,8]. The coefficient matrix in (15) has condition number (in 1-norm or ∞-norm) $2n$ and hence it is quite well-conditioned when $nu \ll 1$, where u is the machine precision. Nevertheless, by using the usual partition methods, factorization (2) may be not defined since some of the blocks $A^{(i)}$, $i = 1, \ldots, p-1$, may be singular. Conversely, by using the parallel QR factorization algorithm described in Section 3, we obtain a stable solver with a reduced system of size at most $2p - 1$. The measured speedups on 4, 8 and 16 processors versus the size n of the problem are shown in Fig. 1. Observe that the obtained results clearly show the linear dependence on p of the measured asymptotic speedup, as predicted in (14).

Acknowledgements

The authors thank Ms. Paulene Butts for her help in the preparation of the manuscript.

References

[1] P. Amodio and L. Brugnano, Parallel factorizations and parallel solvers for tridiagonal linear systems, *Linear Algebra Appl.* 172 (1992) 347–364.
[2] P. Amodio, L. Brugnano and T. Politi, Parallel factorizations for tridiagonal matrices, *SIAM J. Numerical Anal.* 30 (1993) 813–823.
[3] P. Amodio and N. Mastronardi, A parallel version of the cyclic reduction algorithm on a hypercube, *Parallel Comput.* 19 (1993) 1273–1281.
[4] S. Bondeli, Divide and Conquer: a new parallel algorithm for the solution of a tridiagonal linear system of equations, Tech. Report 130, ETH Zürich, Department Informatik, Institut für Wissenschaftliches Rechnen, Zürich, Switzerland, 1990.
[5] L. Brugnano, A parallel solver for tridiagonal linear systems for distributed memory parallel computers, *Parallel Comput.* 17 (1991) 1017–1023.
[6] L. Brugnano and D. Trigiante, Tridiagonal matrices: invertibility and conditioning, *Linear Algebra Appl.* 166 (1992) 131–150.
[7] L. Brugnano and D. Trigiante, Stability properties of some BVM methods, *Applied Numer. Math.* 13 (1993) 291–304.
[8] L. Brugnano and D. Trigiante, A parallel preconditioning technique for BVM methods, *Applied Numer. Math.* 13 (1993) 277–290.
[9] D.J. Evans and W.S. Yousif, The solution of two-point boundary value problems by the alternating group explicit (AGE) method, *SIAM J. Scientific and Statis. Comput.* 9 (1988) 474–484.
[10] I.N. Hajj and S. Skelboe, A multilevel parallel solver for block tridiagonal and banded linear systems, *Parallel Comput.* 15 (1990) 21–45.
[11] S.L. Johnsson, Solving tridiagonal systems on enseble architectures, *SIAM J. Scientific and Statist. Comput.* 8 (1987) 354–392.
[12] A. Krenchel, H.J. Plum and K. Stüben, Parallelization and vectorization aspects of the solution of tridiagonal linear systems, *Parallel Comput.* 14 (1990) 31–49.
[13] V. Mehrmann, Divide and conquer methods for block tridiagonal systems, *Parallel Comput.* 19 (1993) 257–279.
[14] J.M. Ortega, *Introduction to Parallel and Vector Solution of Linear systems* (Plenum Press, New York, 1988).
[15] H.H. Wang, A parallel method for tridiagonal linear systems, *ACM Trans. Math. Software* 7 (1981) 170–183.
[16] Express Fortran User's Guide, ParaSoft Corporation, Pasadena, 1990.