Parallel Computing 17 (1991) 1017–1023
North-Holland

A parallel solver for tridiagonal linear systems for distributed memory parallel computers *

L. Brugnano

Dipartimento di Matematica, Trav. 200 Re David, 70125 Bari, Italy

Received 20 July 1990

Abstract

Brugnano, L., A parallel solver for tridiagonal linear systems for distributed memory parallel computers, Parallel Computing 17 (1991) 1017–1023.

The solution of linear tridiagonal systems is a very common problem in Numerical Analysis. Many algorithms are known for solving such linear systems on vector and parallel computers [3,4,6–9]. In this paper a new parallel method is presented, which is well tailored for message passing distributed memory parallel computers.

Keywords. Linear algebra; tridiagonal linear systems; distributed memory multiprocessors; transputer networks; timing results.

1. Introduction

We start introducing the sequential algorithm [2,5], which is intended to ba applied to the solution of the linear system:

$$Ty = b, \tag{1.1}$$

where $T \in \mathbb{R}^{n\times n}$, y, $b \in \mathbb{R}^n$. The structure of T is the following (in order to simplify the notations):

$$T = \begin{bmatrix} a_1 & c_1 & & & \\ 1 & \ddots & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ & & & 1 & a_n \end{bmatrix}. \tag{1.2}$$

Let us consider the sequences defined as follows:

$$\begin{cases} x_1 = 0 \\ x_{i+1} = c_i \sigma_i, \quad i = 1 \dots n-1, \end{cases} \tag{1.3.1}$$

$$\sigma_i = -(a_i + x_i)^{-1}, \quad i = 1 \dots n, \tag{1.3.2}$$

$$\begin{cases} z_1 = 0 \\ z_{i+1} = (z_i - b_i)\sigma_i, \quad i = 1 \dots n, \end{cases} \tag{1.3.3}$$

* Work supported by the Ministero della Ricerca Scientifica, 40% project, and by the European Community, ESPRIT project (Parallel Computing Action).

and

$$\begin{cases} y_n = z_{n+1} \\ y_{i-1} = x_i y_i + z_i, \quad i = n \dots 2. \end{cases} \tag{1.3.4}$$

If the sequences (1.3.1), (1.3.2) and (1.3.3) are well defined, it is a simple matter to show that the elements defined by (1.3.4) are the components of the solution vector y of problem (1.1). For the stability of this algorithm, usually known as the *sweep* method, see [2,5].

Remark 1.1. The computational cost of algorithm (1.3.1) $\cdots$ (1.3.4), in terms of number of operations, is the same as of the standard LU algorithm.

2. Parallelization of the algorithm for a biputer

The algorithm previously described can be modified, to get a parallel solver tailored for a parallel computer with two processors. The method will be then generalized to any number of processors. Let us number the processors starting from 0, by using the index j. Moreover, let us define the following sequences and let be, for sake of simplicity, $n = 2k$:

$$\begin{cases} x_i^{(j)} = 0 \\ x_{i+1}^{(j)} = c_{i+jk}\sigma_i^{(j)}, \quad i = 1 \dots k-1, \end{cases} \tag{2.1.1}$$

$$\sigma_i^{(j)} = -\left(a_{i+jk} + x_i^{(j)}\right)^{-1}, \quad i = 1 \dots k, \tag{2.1.2}$$

$$\begin{cases} z_1^{(j)} = 0 \\ z_{i+1}^{(j)} = \left(z_i^{(j)} - b_{i+jk}\right)\sigma_i^{(j)}, \quad i = 1 \dots k. \end{cases} \tag{2.1.3}$$

The above expressions are computed on processor j, for $j = 0$, 1. It is evident that these calculations are independent, on each processor. The aim is to transform problem (1.1) into two boundary value problem, with one unknown initial condition. This unknown value is the initial condition for the sequence $\{z_i^{(1)}\}$. We observe that such nonzero initial condition produces an easily valuable variation in the sequence. In fact, one verifies that:

$$z_i^{(j)} = \left(\prod_{r=1}^{i-1} \sigma_r^{(j)}\right) z_1^{(j)} - \sum_{r=1}^{i-1} b_{r+jk} \prod_{s=r}^{i-1} \sigma_s^{(j)}, \quad i = 1 \dots k. \tag{2.2}$$

The second term in (2.2) is relative to the sequence with omogeneous initial condition, that is the one defined by (2.1.3), while the first one takes into account of nonzero ones. It follows that, by defining

$$\psi_s^{(j)} = \prod_{r=1}^{s} \sigma_r^{(j)}, \tag{2.3}$$

and with reference to the elements of sequence (2.1.3), one can define the following sequence:

$$\tilde{z}_i^{(j)} = \psi_{i-1}^{(j)} \tilde{z}_1^{(j)} + z_i^{(j)}. \tag{2.4}$$

With this notations, we define the following:

$$\begin{cases} y_n \equiv y_{2k} = \tilde{z}_{k+1}^{(1)} \\ y_{2k-i} = x_{k-i+1}^{(1)} y_{2k-i+1} + \tilde{z}_{k-i+1}^{(1)}, \quad i = 1 \dots k \\ y_{k-i} = x_{k-i+1}^{(0)} y_{k-i+1} + \tilde{z}_{k-i+1}^{(0)}, \quad i = 1 \dots k-1. \end{cases} \tag{2.5}$$

It must be $\tilde{z}_1^{(0)} = 0$, because $y_0 = 0$. Moreover, recalling that, from (2.5), it is $y_k = \tilde{z}_1^{(1)}$, it follows that

$$y_{k-1} = x_k^{(0)} \tilde{z}_1^{(1)} + z_k^{(0)},$$

and

$$\begin{aligned} y_{k+1} &= x_2^{(1)} y_{k+2} + \tilde{z}_2^{(1)} = x_2^{(1)} y_{k+2} + \psi_1^{(1)} \tilde{z}_1^{(1)} + z_2^{(1)} \\ &= x_2^{(1)} \left(x_3^{(1)} y_{k+3} + \tilde{z}_3^{(1)} \right) + \psi_1^{(1)} \tilde{z}_1^{(1)} + z_2^{(1)} \\ &= \ldots\ldots \\ &= \sum_{r=1}^{k} \left(\prod_{s=2}^{r} x_s^{(1)} \right) \left(\psi_r^{(1)} \tilde{z}_1^{(1)} + z_{r+1}^{(1)} \right) \\ &= \left(\sum_{r=1}^{k} \Phi_r^{(1)} \psi_r^{(1)} \right) \tilde{z}_1^{(1)} + \sum_{r=1}^{k} \Phi_r^{(1)} z_{r+1}^{(1)} \\ &= G^{(1)} \tilde{z}_1^{(1)} + Q^{(1)}, \end{aligned} \tag{2.6}$$

where, in general ($j = 1$ in our case),

$$\Phi_r^{(j)} = \prod_{s=2}^{r} x_s^{(j)}. \tag{2.7}$$

The equations of problem (1.1) are all satisfied, with the only exception for the kth one. It follows that the unknown initial condition $\tilde{z}_1^{(1)}$ is obtained by imposing that the kth equation

$$y_{k-1} + a_k y_k + c_k y_{k+1} = b_k,$$

must be satisfied. One obtains, after simple calculations,

$$\tilde{z}_1^{(1)} = \frac{b_k - c_k Q^{(1)} - z_k^{(0)}}{x_k^{(0)} + a_k + c_k G^{(1)}}. \tag{2.8}$$

3. Generalization of the algorithm to p processors

Let us suppose, as in the previous section, $n = kp$. By numbering the processors from 0 to $(p-1)$, we consider the sequences (2.1.1), (2.1.2) and (2.1.3) defined for $j = 0 \ldots p-1$. As before, the aim is to break problem (1.1) into p smaller boundary value problems, with $p-1$ unknown boundary conditions. The solution vector is obtained by the following:

$$\begin{cases} y_n \equiv y_{pk} = \tilde{z}_{k+1}^{(p-1)} \\ y_{(j+1)k-i} = x_{k-i+1}^{(j)} y_{(j+1)k-i+1} + \tilde{z}_{k-i+1}^{(j)}, \quad i = 1 \ldots k, \quad j = 0 \ldots p-1. \end{cases} \tag{3.1}$$

As before, the initial conditions $\tilde{z}_1^{(j)}$, $j = 1 \ldots p-1$ are unknowns. Such values will be obtained by imposing that the kth, $2k$th, $\ldots$, $(p-1)k$th equations of the problem,

$$y_{jk-1} + a_{jk} y_{jk} + c_{jk} y_{jk+1} = b_{jk}, \quad j = 1 \ldots p-1, \tag{3.2}$$

must be satisfied. One obtains, after some calculations (see (2.1.3), (2.3), (2.4) and (2.7)):

$$\begin{aligned} y_{jk} &= \tilde{z}_1^{(j)}, \\ y_{jk-1} &= x_k^{(j-1)} \tilde{z}_1^{(j)} + \psi_{k-1}^{(j-1)} \tilde{z}_1^{(j-1)} + z_k^{(j-1)}, \\ y_{jk+1} &= G^{(j)} \tilde{z}_1^{(j)} + Q^{(j)} + \Phi_k^{(j)} \tilde{z}_1^{(j+1)}, \end{aligned} \tag{3.3}$$

where

$$Q^{(j)} = \sum_{r=1}^{m_j} \Phi_r^{(j)} z_{r+1}^{(j)},$$

$$G^{(j)} = \sum_{r=1}^{m_j} \Phi_r^{(j)} \psi_r^{(j)},$$

and

$$m_j = \begin{cases} k-1, & \text{for } j = 1 \dots p-2, \\ k, & \text{fo } j = p-1. \end{cases}$$

From (3.2) and (3.3), by considering that $\tilde{z}_1^{(0)} = \tilde{z}_1^{(p)} = 0$, one obtains a *tridiagonal* set of $p-1$ equations, in the $p-1$ unknowns $\tilde{z}_1^{(j)}$, $j = 1 \dots p-1$. Once this "small" linear system (usually it is $p \ll n$) is solved, one obtains the solution, by using (3.1): this is done in parallel on all the processors (the same is true for the computation of (2.1.1), (2.1.2) and (2.1.3)).

Before discussing the expected performance of the parallel algorithm, let us draw a scheme of it, for $p \geq 2$ processors, by using a programming-like language. The processors are numbered from 0 to $p-1$. We assume to have a distributed memory parallel computer (the algorithm for a shared memory parallel computer follows immediately).

Moreover we shall use the following three subroutines, which can be implemented quite easily on a parallel computer:

(1) input(proc, n, data_1, data_2,..., data_n),
this subroutine reads from processor proc the n datas data_1 ... data_n.
(2) output(proc, n, data_1, data_2,..., data_n),
this subroutine sends to processor proc the n datas data_1 ... data_n.
(3) concatenate(v1, v2, n1, n2, d_1,..., d_n1, e_1,..., e_n2),
this subroutine concatenates, for increasing processor number, the n1 datas d_1 ... d_n1 to the vector v1, and the n2 datas e_1 ... e_n2 to the vector v2.

In the following $T = (t_{ij})$ is an auxiliar tridiagonal matrix, while $\boldsymbol{d} = (d_i)$ and $zz = (zz_i)$ are vectors.

Processor 0
$x_1^{(0)} = 0$
$z_1^{(0)} = 0$
for $i = 1 \dots k-1$
  $\sigma_i^{(0)} = -(a_i + x_i^{(0)})^{-1}$
  $x_{i+1}^{(0)} = c_i \sigma_i^{(0)}$
  $z_{i+1}^{(0)} = (z_i^{(0)} - b_i)\sigma_i^{(0)}$
end
input (1, 2, $G^{(1)}$, $Q^{(1)}$)
$t_{11} = x_k^{(0)} + a_k + c_k G^{(1)}$
$d_1 = b_k - z_k^{(0)} - c_k Q^{(1)}$
concatenate (T, $\boldsymbol{d}$, 1, 1, t_{11}, d_1)
$zz = T^{-1}\boldsymbol{d}$
$y_k = zz_1$
for $i = k \dots 2$, **step** -1
  $y_{i-1} = x_i^{(0)} y_i + z_i^{(0)}$
end

Processor j, $j = 1 \ldots p-2$
$x_1^{(j)} = 0$
$z_1^{(j)} = 0$
for $i = 1 \ldots k$
$\sigma_i^{(j)} = -(a_{jk+i} + x_i^{(j)})^{-1}$
$x_{i+1}^{(j)} = c_{jk+i}\sigma_i^{(j)}$
$z_{i+1}^{(j)} = (z_i^{(j)} - b_{jk+i})\sigma_i^{(j)}$
end
$\Phi_1^{(j)} = 1$
$Q^{(j)} = 0$
$\Psi_1^{(j)} = \sigma_1^{(j)}$
$G^{(j)} = 0$
for $i = 2 \ldots k$
$Q^{(j)} = Q^{(j)} + \Phi_{i-1}^{(j)} z_i^{(j)}$
$G^{(j)} = G^{(j)} + \Phi_{i-1}^{(j)} \Psi_{i-1}^{(j)}$
$\Phi_i^{(j)} = \Phi_{i-1}^{(j)} x_i^{(j)}$
$\Psi_i^{(j)} = \Psi_{i-1}^{(j)} \sigma_i^{(j)}$
end
output ($j-1$, 2, $G^{(j)}$, $Q^{(j)}$)
input ($j+1$, 2, $G^{(j+1)}$, $Q^{(j+1)}$)
$t_{j+1,j} = \Psi_{k-1}^{(j)}$
$t_{j+1,j+1} = x_k^{(j)} + a_{(j+1)k} + c_{(j+1)k} G^{(j+1)}$
$t_{j,j+1} = c_{jk}\Phi_k^{(j)}$
$d_{j+1} = b_{(j+1)k} - z_k^{(j)} - c_{(j+1)k} Q^{(j+1)}$
concatenate (T, $\boldsymbol{d}$ 3, 1, $t_{j+1,j}$, $t_{j+1,j+1}$, $t_{j,j+1}$, d_{j+1})
$zz = T^{-1}\boldsymbol{d}$
$z_1^{(j)} = zz_j$
for $i = 1 \ldots k$
$z_{i+1}^{(j)} = \Psi_i^{(j)} zz_j + z_{i+1}^{(j)}$
end
$y_{(j+1)k} = zz_{j+1}$
for $i = k \ldots 2$, **step** -1
$y_{jk+i-1} = x_i^{(j)} y_{jk+i} + z_i^{(j)}$
end
Processor $p-1$
$x_1^{(p-1)} = 0$
$z_1^{(p-1)} = 0$
for $i = 1 \ldots k$
$\sigma_i^{(p-1)} = -(a_{(p-1)k+i} + x_i^{(p-1)})^{-1}$
$x_{i+1}^{(p-1)} = c_{(p-1)k+i}\sigma_i^{(p-1)}$
$z_{i+1}^{(p-1)} = (z_i^{(p-1)} - b_{(p-1)k+i})\sigma_i^{(p-1)}$
end
$\Phi_1^{(p-1)} = 1$
$Q^{(p-1)} = z_2^{(p-1)}$
$\Psi_1^{(p-1)} = \sigma_1^{(p-1)}$
$G^{(p-1)} = \Psi_1^{(p-1)}$
for $i = 2 \ldots k$
$\Phi_i^{(p-1)} = \Phi_{i-1}^{(p-1)} x_i^{(p-1)}$
$Q^{(p-1)} = Q^{(p-1)} + \Phi_i^{(p-1)} z_{i+1}^{(p-1)}$

$\Psi_i^{(p-1)} = \Psi_{i-1}^{(p-1)} \sigma_i^{(p-1)}$
$G^{(p-1)} = G^{(p-1)} + \Phi_i^{(p-1)} \Psi_i^{(p-1)}$
end
output $(p-2, 2, G^{(p-1)}, Q^{(p-1)})$
concatenate $(T, \boldsymbol{d}, 0, 0)$
$zz = T^{-1}\boldsymbol{d}$
$z_1^{(p-1)} = zz_{p-1}$
for $i = 1 \ldots k$
$z_{i+1}^{(p-1)} = \Psi_i^{(p-1)} zz_{p-1} + z_{i+1}^{(p-1)}$
end
$y_n = z_{k+1}^{(p-1)}$
for $i = k \ldots 2$, **step** -1
$y_{n-k+i-1} = x_i^{(p-1)} y_{n-k+i} + z_i^{(p-1)}$
end

4. Numerical tests

We have said (see Remark 1.1) that algorithm (1.3.1) $\cdots$ (1.3.4) has the same efficiency as the standard LU algorithm. It follows that there will be no distinction between the speed-up of the algorithm and the speed-up of the problem. The cost of this sequential algorithm, for problem (1.1), is of about $7n$ operations, while the cost of the parallel variant (see (2.1.1), (2.1.2), (2.1.3) and (3.1)) is of about $15n$ operations.

Moreover, we must take into account, for the parallel algorithm, the cost for data transmissions among the processors, if one uses a message passing parallel computers. This cost amounts, when using p processors working in parallel, to less than $4p$ data transmitted, for every node. In fact, by observing the scheme of the parallel algorithm, every processors makes at most 1 call to output (2 datas) and 1 call to input (2 datas) to the adjacent processors, moreover there is 1 call to concatenate (4 datas). This last routine is easily constructed, on a ring topology, with $(p-1)$ calls to output and $(p-1)$ calls to input to the adjacent nodes. It follows that we have at most p calls to output and p calls to input per processor (each call to the former routine is associated to a call to the latter): this number is important, because the overhead for synchronization, at every call, is usually very high. Nevertheless, if $n \gg p$, we could expect a maximum speed-up of $7p/15$.

The parallel computer used for the numerical tests is a Multiputer by MicroWay with 32 processors. Each processor is a transputer T800 by INMOS. Each transputer has four hardware links connected to an electronic cross bar: in this way it is a simple matter to configure the processors to form a ring.

The programming language used is Fortran, with the Express software environment [10]. We have found that the call to the function KXCONC of Express is not an efficient way to construct concatenate. The most efficient way we have found is to use $(p-1)$ calls the KXWRIT and KXREAD, that are the equivalent of output and input, respectively.

The test problem chosen has (see (1.2)) $a_i = 4$, $c_i = -1$. The right hand side is defined in order to have the following solution vector:

$$y_i = 1, \quad \text{for} \quad i = 1 \ldots n.$$

The found solution is very accurate, while the results, in terms of speed-up, are summarized in *Fig. 1*. In this figure the measured speed-ups on $p = 6$, 12 18, 24, 30 processors, versus the dimension n of the problem are shown.

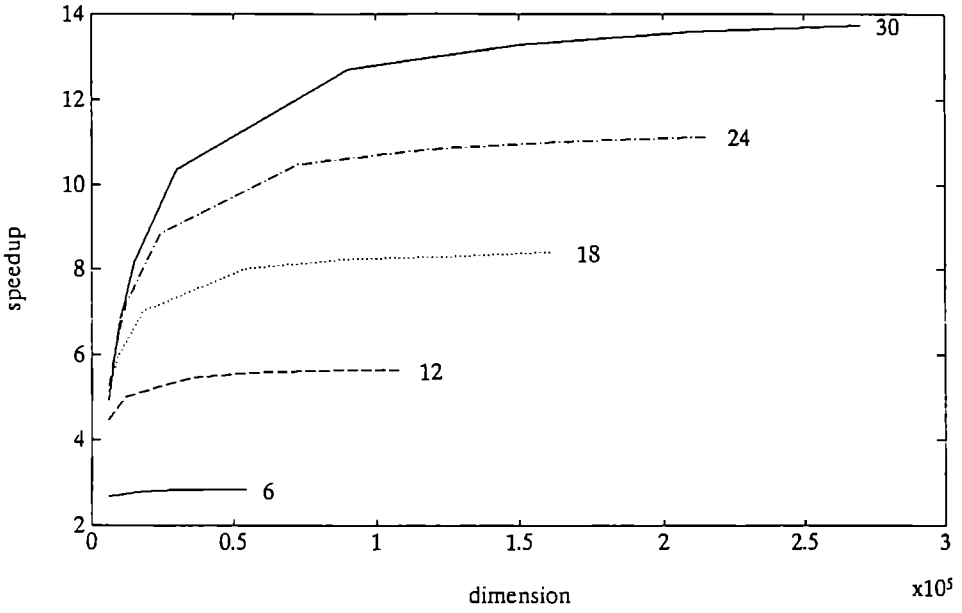

Fig. 1.

Table 1

Dimension	T_1	T_{30}
30000	0.90	0.09
90000	2.71	0.21
150000	4.51	0.34
210000	6.31	0.46
270000	8.11	0.59

At last, in *Table 1*, there are the execution times (in sec), for the LU algorithm on 1 processor, and the parallel algorithm, on 30 processors, for various dimensions of the problem.

References

[1] B.L. Buzbee, G.H. Golub and C.W. Nielson, On direct methods for solving Poisson's equations, *SIAM J. Num. Anal.* 7 (4) (1970).

[2] S. Godounov and V. Riabenki, *Schémas aux Différences* Chaps. 2 and 3 (Éditions MIR, Moscou, 1977).

[3] D. Heller, Some aspects of the cyclic reduction algorithm for block tridiagonal linear systems, *SIAM J. Num. Anal.* 13 (4) (1976) 484–496.

[4] D. Kershaw, Solution of single tridiagonal linear systems and vectorization of the ICCG algorithm on the Cray-1, *Parallel Computations* (Acedemic Press, 1982) pp. 85–99.

[5] V. Lakshmikantham and D. Trigiante, *Theory of Difference Equations: Numerical Methods and Applications, Series Math. in Sci. and Engin.* Vol. 181 Chap. 5 (Academic Press, 1988).

[6] H.S. Stone, Parallel tridiagonal equation solver, NASA Report TMX-62, 370 (1974).

[7] H. Van der Vorst, Large tridiagonal and block tridiagonal linear systems on vector and parallel computers, *Parallel Comput.* 5 (1987) 45–54.

[8] H. Van der Vorst, Analysis of a parallel solution method for tridiagonal linear systems, *Parallel Comput.* 5 (1987) 303–311.

[9] H.H. Wang, A parallel Method for tridiagonal equations, *ACM Trans. Math. Softw.* 7 (2) (1981) 170–183.

[10] ParaSoft Corporation, Express. Reference for 3L Fortran (1990).