ELSEVIER

Information Processing Letters 65 (1998) 331–337

Information Processing Letters

A parallel algorithm for circulant tridiagonal linear systems [1]

Yaw-Wen Chang, Chang-Biau Yang *

Department of Applied Mathematics, National Sun Yat-sen University, Kaohsiung 804, Taiwan, ROC

Received 1 April 1997; revised 1 June 1997
Communicated by S.G. Akl

Abstract

In this paper, based upon the divide-and-conquer strategy, we propose a parallel algorithm for solving the circulant tridiagonal systems, which is simpler than the two previous algorithms proposed by Agui and Jimenez (1995) and Chung et al. (1995). Our algorithm can be easily generalized to solve the tridiagonal systems, the block-tridiagonal systems, and the circulant block-tridiagonal systems. © 1998 Elsevier Science B.V.

Keywords: Circulant tridiagonal system; Parallel algorithm; Divide-and-conquer

1. Introduction

Many scientific and engineering problems can be transformed into the problems of solving linear systems. Thus, solving linear systems is the kernel issue for solving scientific and engineering problems. Among them, especially, tridiagonal systems [1,7], block-tridiagonal systems [5], banded systems [6], circulant tridiagonal systems [4], and block-circulant systems [4] are most frequently discussed. For instance, finite difference approximations to certain elliptic partial differential equations can be transformed into a block-tridiagonal system [5,8]. Closed curve fitting is important in computer-aided design, graphics, pattern recognition and picture processing [2], and it can be transformed into a circulant tridiagonal system.

In this paper, based on the divide-and-conquer strategy, we shall propose a simple and efficient parallel solver of circulant tridiagonal systems, which can also be used to efficiently solve tridiagonal systems, circulant block-tridiagonal systems and block-tridiagonal systems. A complete circulant tridiagonal system of order n, $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{d}$, can be expressed as

$$
\begin{aligned}
&a_1x_1 + b_1x_2 + c_1x_n = d_1,\\
&c_ix_{i-1} + a_ix_i + b_ix_{i+1} = d_i, \quad \text{for } 2 \leqslant i \leqslant n-1,\\
&b_nx_1 + c_nx_{n-1} + a_nx_n = d_n.
\end{aligned}
$$

* Corresponding author. Email: cbyang@math.nsysu.edu.tw.

[1] This research work was partially supported by the National Science Council of the Republic of China under contract NSC 85-2121-M-110-014 C.

PII S0020-0190(98)00018-0

$$A = \begin{bmatrix} a_1 & b_1 & & & \cdots & c_1 \\ c_2 & a_2 & b_2 & & & \\ & c_3 & a_3 & b_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & & & b_{n-1} \\ b_n & & & & c_n & a_n \end{bmatrix}$$

Fig. 1. The form of matrix A.

$$A = \left[\begin{array}{cccccccccccc} a_1 & & & b_1 & & & & & & & & c_1 \\ & a_2 & & b_2 & & & & & & & & c_2 \\ & & a_3 & b_3 & & & & & & & & c_3 \\ & & & a_4 & b_4 & & & & & & & c_4 \\ \hline & & & c_5 & a_5 & & & b_5 & & & & \\ & & & c_6 & & a_6 & & b_6 & & & & \\ & & & c_7 & & & a_7 & b_7 & & & & \\ & & & c_8 & & & & a_8 & b_8 & & & \\ \hline & & & & & & & & \ddots & & & \\ \hline b_{21} & & & & & & & & & c_{21} & a_{21} & \\ b_{22} & & & & & & & & & c_{22} & & a_{22} \\ b_{23} & & & & & & & & & c_{23} & & & a_{23} \\ b_{24} & & & & & & & & & c_{24} & & & & a_{24} \end{array}\right]$$

Fig. 2. The state after executing Gaussian elimination in each processor.

The form of matrix A is shown in Fig. 1. Clearly, a tridiagonal system is a special case of the circulant tridiagonal systems. That is, in a tridiagonal system, $c_1 = 0$ and $b_n = 0$. In this paper, it is assumed that matrix A is nonsingular and all procedures in the following have no pivoting problem.

2. The previous results

In this section, we shall review two previous parallel solvers based on the divide-and-conquer strategy. Agui and Jimenez [1] proposed a parallel algorithm to solve the tridiagonal systems. And, Chung et al. [4] proposed a parallel algorithm to solve the circulant block-tridiagonal systems. In fact, Agui's algorithm can also solve the circulant tridiagonal systems.

Agui and Jimenez's algorithm [1] partitions the n equations into p groups, each forming a subsystem, is handled by a processor. The *N-diagonalization scheme*, used previously by Wang and Mou [9], is performed on each processor. In the N-diagonalization scheme, Gaussian elimination is performed on the coefficient submatrix to transform it into an N-shaped matrix on a processor. Then, after putting the first and the last equations, which are the interface equations between consecutive processors, together in each processor, a special reduction scheme is used to transform the reduced system to a smaller N-shaped system. Thus, the algorithm can be applied recursively.

Chung et al.'s algorithm [4] uses the same scheme of partition and Gaussian elimination. After putting the first and the last equations together in each processor, Chung reorders the corresponding columns to transform the matrix into a smaller circulant tridiagonal matrix. Thus, the algorithm can also be applied recursively.

Now, we show an example to illustrate the two algorithms. Suppose that $n = 24$ and $p = 6$, where p is the number of processors used. And let $m = n/p = 4$. Thus, there are 24 equations, denoted as g_i, $1 \leqslant i \leqslant 24$, in the system. Each g_i is assigned to processor $\lceil i/4 \rceil$. Note that the p processors are identified by $1, 2, \ldots, p$.

$$A = \begin{bmatrix}
a_1 & & b_1 & & & & & & & & & c_1 \\
 & a_4 & b_4 & & & & & & & & & c_4 \\
 & c_5 & a_5 & & b_5 & & & & & & & \\
 & c_8 & & a_8 & b_8 & & & & & & & \\
 & & & c_9 & a_9 & & b_9 & & & & & \\
 & & & c_{12} & & a_{12} & b_{12} & & & & & \\
 & & & & & c_{13} & a_{13} & & b_{13} & & & \\
 & & & & & c_{16} & & a_{16} & b_{16} & & & \\
 & & & & & & & c_{17} & a_{17} & & b_{17} & \\
 & & & & & & & c_{20} & & a_{20} & b_{20} & \\
b_{21} & & & & & & & & & c_{21} & a_{21} & \\
b_{24} & & & & & & & & & c_{24} & & a_{24}
\end{bmatrix}$$

Fig. 3. The reduced system by taking first and last equations in each processor.

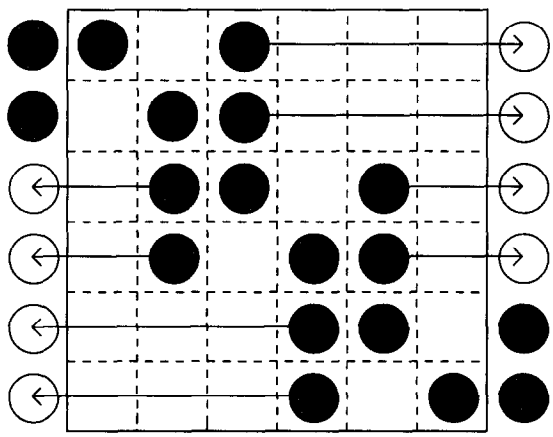

Fig. 4. The reduction scheme in each processor.

After Gaussian elimination is executed in each processor, the form in each processor is N-shaped, as shown in Fig. 2.

From the overall view, after the first and the last equations in each processor are put together, we shall get a subsystem, called the *reduced system*, as shown in Fig. 3. Agui and Jimenez implemented their algorithm on the binary tree machine. Thus, Agui and Jimenez's algorithm groups the equations in one node (processor) with its two sons (six equations) together, and uses the parent processor to handle the six equations. He then applies a special reduction scheme, as shown in Fig. 4, to the reduced system. From the overall view, a special system, consisting of several N-shaped subsystems, shown in Fig. 5, is obtained. Then, the algorithm is applied recursively.

On the other hand, after the reduced system shown in Fig. 3 is obtained, Chung et al.'s algorithm reorders these columns of the reduced system to a smaller circulant tridiagonal system, as shown in Fig. 6. Then, the algorithm is applied recursively.

3. The algorithms for the circulant tridiagonal systems

In the two previous algorithms, they have to perform a special reduction scheme or column reordering. In this paper, we propose a new parallel solver, based upon the same divide-and-conquer scheme, but simpler than their algorithms. Now, we present our algorithm for the circulant tridiagonal systems. We partition the n

$$A = \begin{bmatrix}
a_1 & & & & & & b_1 & & & & & c_1 \\
& a_4 & & & & & b_4 & & & & & c_4 \\
& & a_5 & & & & b_5 & & & & & c_5 \\
& & & a_8 & & & b_8 & & & & & c_8 \\
& & & & a_9 & & b_9 & & & & & c_9 \\
& & & & & a_{12} & b_{12} & & & & & c_{12} \\
b_{13} & & & & & c_{13} & a_{13} & & & & & \\
b_{16} & & & & & c_{16} & & a_{16} & & & & \\
b_{17} & & & & & c_{17} & & & a_{17} & & & \\
b_{20} & & & & & c_{20} & & & & a_{20} & & \\
b_{21} & & & & & c_{21} & & & & & a_{21} & \\
b_{24} & & & & & c_{24} & & & & & & a_{24}
\end{bmatrix}$$

Fig. 5. The special system after performing the reduction scheme.

$$A = \begin{bmatrix}
c_1 & b_1 & & & & & & & & & a_1 \\
c_4 & b_4 & a_4 & & & & & & & & \\
& a_5 & c_5 & b_5 & & & & & & & \\
& & c_8 & b_8 & a_8 & & & & & & \\
& & & a_9 & c_9 & b_9 & & & & & \\
& & & & c_{12} & b_{12} & a_{12} & & & & \\
& & & & & a_{13} & c_{13} & b_{13} & & & \\
& & & & & & c_{16} & b_{16} & a_{16} & & \\
& & & & & & & a_{17} & c_{17} & b_{17} & \\
& & & & & & & & c_{20} & b_{20} & a_{20} \\
& & & & & & & & & a_{21} & c_{21} & b_{21} \\
a_{24} & & & & & & & & & & c_{24} & b_{24}
\end{bmatrix}$$

Fig. 6. The reduced system after reordering the selected columns.

equations evenly into p groups. And, each processor works on n/p consecutive equations, where $m = n/p$ is assumed to be an integer. First, each processor performs the row operations similar to Gaussian elimination. Let $r_i^{(k)}$ denote the ith row in the kth processor, $1 \leqslant i \leqslant m$ and $1 \leqslant k \leqslant p$. Each processor performs the following row operations:

FOR $i = 2$ TO $(m-1)$ DO

$r_{i+1}^{(k)} = r_{i+1}^{(k)} - r_i^{(k)} * (c_{(k-1)m+i+1}/a_{(k-1)m+i})$

FOR $i = (m-1)$ TO 2 DO

$r_{i-1}^{(k)} = r_{i-1}^{(k)} - r_i^{(k)} * (b_{(k-1)m+i-1}/a_{(k-1)m+i})$

Thus, there are $2m-4$ row operations performed in each processor. And, all processors work in parallel. Then, in the first processor, the subsystem will become

$$a_1^* x_1 + b_1^* x_m + c_1^* x_n = d_1^*,$$
$$c_i^* x_1 + a_i^* x_i + b_i^* x_m = d_i^*, \quad \text{for } 2 \leqslant i \leqslant (m-1),$$
$$c_m^* x_1 + a_m^* x_m + b_m^* x_{m+1} = d_m^*.$$

Similarly, in the last processor, the subsystem will be

$$c_{n-m+1}^* x_{n-m} + a_{n-m+1}^* x_{n-m+1} + b_{n-m+1}^* x_n = d_{n-m+1}^*,$$
$$c_{n-m+i}^* x_{n-m+1} + a_{n-m+i}^* x_{n-m+i} + b_{n-m+i}^* x_n = d_{n-m+i}^*, \quad \text{for } 2 \leqslant i \leqslant (m-1),$$
$$b_n^* x_1 + c_n^* x_{n-m+1} + a_n^* x_n = d_n^*.$$

$$A = \left[\begin{array}{ccccccccccccccc}
a_1 & & & b_1 & & & & & & & & & & & c_1 \\
c_2 & a_2 & & b_2 & & & & & & & & & & & \\
c_3 & & a_3 & b_3 & & & & & & & & & & & \\
c_4 & & & a_4 & b_4 & & & & & & & & & & \\
\hline
 & & & c_5 & a_5 & & & b_5 & & & & & & & \\
 & & & & c_6 & a_6 & & b_6 & & & & & & & \\
 & & & & c_7 & & a_7 & b_7 & & & & & & & \\
 & & & & c_8 & & & a_8 & b_8 & & & & & & \\
\hline
 & & & & & & & & & & & & & & \\
\hline
 & & & & & & & & & & c_{21} & a_{21} & & & b_{21} \\
 & & & & & & & & & & & c_{22} & a_{22} & & b_{22} \\
 & & & & & & & & & & & c_{23} & & a_{23} & b_{23} \\
b_{24} & & & & & & & & & & & c_{24} & & & a_{24}
\end{array}\right]$$

Fig. 7. The state after executing our algorithm.

For the remaining $(p-2)$ processors, the kth, $2 \leqslant k \leqslant (p-1)$, subsystem in processor k will be

$$c^*_{(k-1)m+1}x_{(k-1)m} + a^*_{(k-1)m+1}x_{(k-1)m+1} + b^*_{(k-1)m+1}x_{km} = d^*_{(k-1)m+1},$$
$$c^*_{(k-1)m+i}x_{(k-1)m+1} + a^*_{(k-1)m+i}x_{(k-1)m+i} + b^*_{(k-1)m+i}x_{km} = d^*_{(k-1)m+i}, \quad \text{for } 2 \leqslant i \leqslant (m-1),$$
$$c^*_{km}x_{(k-1)m+1} + a^*_{km}x_{km} + b^*_{km}x_{km+1} = d^*_{km}.$$

As an example, let $n = 24$, $p = 6$ and $m = n/p = 4$. The result after the above row operations is shown in Fig. 7. Note that in Agui and Jimenez's algorithm and Chung et al.'s algorithm, the submatrix in each processor is N-shaped after Gaussian elimination. However, in our algorithm, the submatrix in each processor is of shape of smaller N, except that the two corner entries falls outside the shape N. Next, we put the first and the last equations together in each processor, then the reduced system consisting of these equations is still a circulant tridiagonal system with smaller size as follows:

$$a^*_1x_1 + b^*_1x_m + c^*_1x_n = d^*_1,$$
$$c^*_mx_1 + a^*_mx_m + b^*_mx_{m+1} = d^*_m,$$
$$c^*_{(k-1)m+1}x_{(k-1)m} + a^*_{(k-1)m+1}x_{(k-1)m+1} + b^*_{(k-1)m+1}x_{km} = d^*_{(k-1)m+1}, \quad \text{for } 2 \leqslant k \leqslant (p-1),$$
$$c^*_{km}x_{(k-1)m+1} + a^*_{km}x_{km} + b^*_{km}x_{km+1} = d^*_{km}, \quad \text{for } 2 \leqslant k \leqslant (p-1),$$
$$c^*_{n-m+1}x_{n-m} + a^*_{n-m+1}x_{n-m+1} + b^*_{n-m+1}x_n = d^*_{n-m+1},$$
$$b^*_nx_1 + c^*_nx_{n-m+1} + a^*x_n = d^*_n.$$

The reduced system obtained from Fig. 7 has the form as shown in Fig. 6. Since the reduced system, consisting of $2p$ equations, is still a circular tridiagonal system, it can be solved recursively. Once the values of the variables in the reduced system become available, those of the others in each processor can be easily found. The most important is that the reduced system in our algorithm is still a circulant tridiagonal system, thus we do not need to reorder the columns of the reduced system. Therefore, our algorithm is simpler than Chung et al.'s algorithm, in which column reordering is needed in the reduced system, and Agui and Jimenez's algorithm, in which a special reduction scheme is applied.

Since our algorithm is a divide-and-conquer scheme, similar to Agui and Jimenez's algorithm and Chung et al.'s algorithm, our algorithm can also be implemented easily on the hypercube, the tree structure or other interconnected multiprocessor systems. For the detailed parallel implementation of the divide-and-conquer scheme on the circulant tridiagonal system, see [1].

When our algorithm is implemented on the hypercube, the time complexity of our algorithm is the complexity for solving the reduced system plus that of the two initial do-loops and that of the back-substitution process. The two initial do-loops and the back-substitution need $O(n/p)$ time. And, the time complexity of the reduced system can be described by the following recursive formula:

$$T(2p) = T(p) + c,$$

where c is a constant. And, we obtain $T(2p) = O(\log p)$. Thus, the time complexity of our algorithm is $O(n/p) + T(2p) = O(n/p + \log p)$, where n is the number of equations in the circular tridiagonal system and p is the number of processors used. If $p \ll n$, then our algorithm requires $O(n/p)$ time.

4. Tridiagonal systems

In this section, we shall slightly modify our algorithm to solve the tridiagonal systems. In fact, a tridiagonal system is a special case of the circulant tridiagonal systems, that is, $c_1 = 0$ and $b_n = 0$. Our modified algorithm will reduce the total number of equations in the reduced system and increase slightly the number of row operations in the first processor and the last processor. In the first processor, the following row operations are performed:

FOR $i = 1$ TO $(m-1)$ DO
$r_{i+1}^{(1)} = r_{i+1}^{(1)} - r_i^{(1)} * (c_{i+1}/a_i)$
FOR $i = (m-1)$ TO 2 DO
$r_{i-1}^{(1)} = r_{i-1}^{(1)} - r_i^{(1)} (b_{i-1}/a_i)$

And, in the last processor, the following row operations are done:

FOR $i = 2$ TO $(m-1)$ DO
$r_{i+1}^{(m)} = r_{i+1}^{(m)} - r_i^{(m)} * (c_{(n-1)m+i+1}/a_{(n-1)m+i})$
FOR $i = m$ TO 2 DO
$r_{i-1}^{(m)} = r_{i-1}^{(m)} - r_i^{(m)} * (b_{(n-1)m+i-1}/a_{(n-1)m+i})$

And, for the kth processor, $2 \leqslant k \leqslant (p-1)$, it performs the following operations:

FOR $i = 2$ TO $(m-1)$ DO
$r_{i+1}^{(k)} = r_{i+1}^{(k)} - r_i^{(k)} * (c_{(k-1)m+i+1}/a_{(k-1)m+i})$
FOR $i = (m-1)$ TO 2 DO
$r_{i-1}^{(k)} = r_{i-1}^{(k)} - r_i^{(k)} * (b_{(k-1)m+i-1}/a_{(k-1)m+i})$

Thus, the subsystem in the first processor will become

$$a_i^* x_i + b_i^* x_m = d_i^*, \quad \text{for } 1 \leqslant i \leqslant m-1,$$
$$a_m^* x_m + b_m^* x_{m+1} = d_m^*.$$

And, the subsystem in the last processor will become

$$c_{n-m+1}^* x_{n-m} + a_{n-m+1}^* x_{n-m+1} = d_{n-m}^*,$$
$$c_i^* x_{n-m+1} + a_i^* x_i = d_i^*, \quad \text{for } (n-m+2) \leqslant i \leqslant n.$$

Similarly, for the remaining $(p-2)$ processors, the kth, $2 \leqslant k \leqslant (p-1)$, subsystem in processor k is

$$
\begin{aligned}
&c^*_{(k-1)m+1}x_{(k-1)m} + a^*_{(k-1)m+1}x_{(k-1)m+1} + b^*_{(k-1)m+1}x_{km} = d^*_{(k-1)m+1},\\
&c^*_{(k-1)m+i}x_{(k-1)m+1} + a^*_{(k-1)m+i}x_{(k-1)m+i} + b^*_{(k-1)m+i}x_{km} = d^*_{(k-1)m+i}, \quad \text{for } 2 \leqslant i \leqslant (m-1),\\
&c^*_{km}x_{(k-1)m+1} + a^*_{km}x_{km} + b^*_{km}x_{km+1} = d^*_{km}.
\end{aligned}
$$

Therefore, if we put together the last equation in the first processor, the first equation in the last processor, and the first and the last equations in the remaining $(p-2)$ processors, then the reduced system, consisting of the $(2p-2)$ equations, is still a tridiagonal system as follows:

$$
\begin{aligned}
&a^*_m x_m + b^*_m x_{m+1} = d^*_m,\\
&c^*_{(k-1)m+1}x_{(k-1)m} + a^*_{(k-1)m+1}x_{(k-1)m+1} + b^*_{(k-1)m+1}x_{km} = d^*_{(k-1)m+1}, \quad \text{for } 2 \leqslant k \leqslant (p-1),\\
&c^*_{km}x_{(k-1)m+1} + a^*_{km}x_{km} + b^*_{km}x_{km+1} = d^*_{km}, \quad \text{for } 2 \leqslant k \leqslant (p-1),\\
&c^*_{n-m+1}x_{n-m} + a^*_{n-m+1}x_{n-m+1} = d^*_{n-m+1}.
\end{aligned}
$$

And, the reduced system still can be solved recursively.

5. Conclusion

Our algorithm can be easily generalized to solve the circulant block-tridiagonal systems and block-tridiagonal systems [3]. A circulant block-tridiagonal system $\boldsymbol{Ax} = \boldsymbol{d}$ is of the form shown in Fig. 1, except that each entry a_i, b_i, and c_i is replaced by a $q \times q$ submatrix A_i, B_i, and C_i respectively.

For solving tridiagonal systems in parallel, there are two previous algorithms based upon the divide-and-conquer strategy. Agui and Jimenez [1] proposed a parallel algorithm to solve the tridiagonal systems. In their algorithm, a special reduction scheme is applied. And, Chung et al. [4] proposed a parallel algorithm to solve the circulant block-tridiagonal systems. However, the algorithm must reorder the columns in the reduced system in order to apply his algorithm recursively. And, our algorithm is also based upon the divide-and-conquer strategy. In the reducing process, our algorithm can obtain the same form of the system without any other efforts, such the special reduction scheme or reordering columns. Thus, our algorithm is simpler than the previous algorithms.

References

[1] J.C. Agui, J. Jimenez, A binary tree implementation of a parallel distributed tridiagonal solver, Parallel Computing 21 (1995) 233–241.

[2] R.H. Bartels, J.C. Beatty, B.A. Barsky, An Introduction to Splines for Use in Computer Graphics and Geometric Modeling, Morgan Kaufmann, San Mateo, CA, 1987.

[3] Y.W. Chang, C.B. Yang, A parallel algorithm for circulant tridiagonal linear systems, in: Proc. Internat. Conf. on Algorithms, Kaohsiung, Taiwan, 1996.

[4] K.L. Chung, Y.H. Tsai, W.M. Yan, A parallel solver for circulant block-tridiagonal systems, Comput. Math. Appl. 29 (1) (1995) 109–113.

[5] E. Galligani, V. Ruggiero, A polynomial preconditioner for block tridiagonal matrices, Parallel Algorithms Appl. 3 (1994) 227–237.

[6] U. Meier, A parallel partition method for solving banded systems of linear equations, Parallel Computing 2 (1985) 33–43.

[7] P.H. Michielse, H.A. van der Vorst, Data transport in Wang's partition method, Parallel Computing 7 (1988) 87–95.

[8] R.A. Sweet, A cyclic reduction algorithm for solving block tridiagonal systems of arbitrary dimension, SIAM J. Numer. Anal. 14 (4) (1977) 706–719.

[9] X. Wang, Z.G. Mou, A divide-and-conquer method of solving tridiagonal systems on hypercube massively parallel computers, IEEE Comput. Soc. (1991) 0810–817.