Joint Conference of 1996 International Computer Symposium
December 19~21, Kaohsiung, Taiwan, R.O.C.

A Parallel Algorithm for Circulant Tridiagonal Linear Systems *

Yaw-Wen Chang and Chang-Biau Yang
Department of Applied Mathematics
National Sun Yat-sen University
Kaohsiung, Taiwan 80424
cbyang@math.nsysu.edu.tw

Abstract

In this paper, we propose a parallel algorithm for solving the tridiagonal systems, the circulant tridiagonal systems, the block-tridiagonal systems, the circulant block-tridiagonal systems and the banded systems, which is an improvement of Agui's and Chung's algorithms. Our algorithm is based on the divide-and-conquer strategy and is suitable for implementation on the hypercube or the binary tree. The applications of our algorithm include solving partial differential equations, the closed B-spline curve fitting problem, the dominant eigenvalues of a matrix and many other scientific, physical and engineering problems.

Key words. circulant tridiagonal systems, parallel algorithm, divide-and-conquer.

1 Introduction

Many scientific and engineering problems can be transformed into the problems of linear systems. Thus, solving the linear systems is the kernel issue for solving scientific and engineering problems. Among them, especially, tridiagonal systems [1, 13], block-tridiagonal systems [9], banded systems [12], circulant tridiagonal systems [6], and block-circulant systems [6] are most frequently discussed. For instance, finite difference approximations to certain elliptic partial differential equations can be transformed into a block-tridiagonal system [9,15]. Closed curve fitting is important in computer-aided design, graphics, pattern recognition and picture processing [4], and it can be transformed into a circulant tridiagonal system. In the past, the algorithms solving linear systems include Gaussian elimination [3], the LU factorization [2], the power method [5], the cyclic reduction method [15], the iterative method [7], the Jacobi method [11], Wang's partition [13], the implicit elimination [8], Given's transformation [14], the preconditioned conjugate gradient method [9], the divide-and-conquer method [1, 6], and so on.

In this paper, we shall propose a fast and efficient parallel solver based on the divide-and-conquer strategy. In Section 2, we shall present two conventional algorithms proposed by Chung [6] and Agui [1]. In Section 3, we propose a parallel algorithm for solving the circulant tridiagonal systems , which is an improvement of the divide-and-conquer method proposed by Chung [6]. And, we shall slightly modify our algorithm to solve the tridiagonal systems to reduce the total number of the equations in the reduced system. This will be given in Section 4. In Section 5, we shall generalize our algorithm to solve the circulant block-tridiagonal systems. Since the circulant tridiagonal systems and the banded systems are also special cases of the circulant block-tridiagonal systems, they can also be solved by our generalized algorithm. In Section 6 , we shall show that our algorithm is superior to Agui's and Chung's. Finally, the conclusion will be given in Section 7.

2 The Previous Results

In this section, we shall review two previous parallel solvers based on the divide-and-conquer strategy for the tridiagonal systems and the circulant tridiagonal systems.

Agui [1] proposed a parallel algorithm to solve the tridiagonal systems. And, Chung [6] proposed a parallel algorithm to solve the circulant block-tridiagonal systems. In fact, Agui's algorithm can also solve the circulant tridiagonal systems.

A complete circulant tridiagonal system of order n, $\mathbf{Ax=d}$, can be expressed as

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

*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.

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

Figure 1: The form of matrix A.

The form of matrix A is shown in Figure 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. It is assumed that matrix A is non-singular and all algorithms in the following have no pivoting problem.

Agui's algorithm [1] partitions the n equations into p groups, each forming a subsystem. Then the *N-diagonalization scheme*, used previously by Wang and Mou [16] and [10], is performed. In the *N-diagonalization scheme*, Gaussian elimination is performed on the coefficient matrix in each node to transform the coefficient matrix into an n-shaped matrix. Then, after putting the first and the last equations together in each processor, the *N-diagonalization scheme* is used again to transform the reduced system to a special system (not a circulant tridiagonal system). Thus, his algorithm can be applied recursively.

Chung' algorithm [6] uses the same partition and Gaussian elimination. After putting the first and the last equations together in each processor, Chung first reorders the corresponding columns to transform the matrix into a smaller circulant tridiagonal matrix. Thus, his algorithm can also be applied recursively.

Now, we present an example for illustrating the two algorithms proposed by Chung [6] and Agui [1]. Suppose that n=18 and p=6, where p is the number of processors used. And let $m = \frac{n}{p}$=3. Thus, there are eighteen equations, denoted as $g_i, 1 \leq i \leq 18$, in the system. Each g_i is assigned to processor $\lceil \frac{i}{3} \rceil$, as shown in Figure 2. Note that the n processors are identified by $1, 2, \cdots, n$. After the *N-diagonalization scheme* is executed in each processor, the result is shown in Figure 3.

Next, if we put the first and the last equations together in each processor, we shall get a subsystem, called the *reduced system*, as shown in Figure 4. Note that Agui partitions the reduced system two groups, but Chung partitions it three groups. Agui [1] repeat-

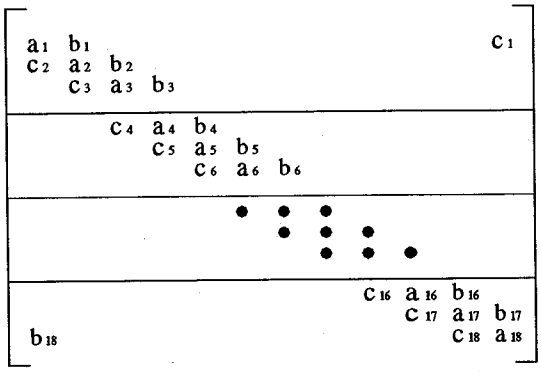

Figure 2: An example of Chung's algorithm.

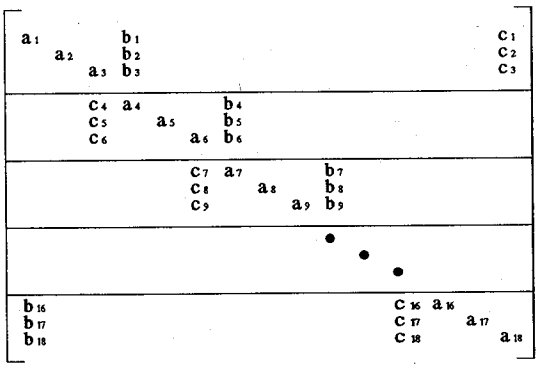

Figure 3: The state after executing Gaussian elimination in each processor.

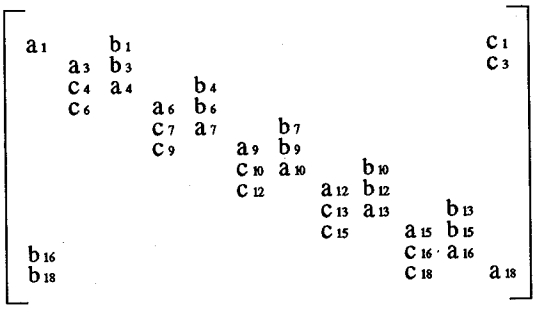

Figure 4: The reduced system before reordering the selected columns.

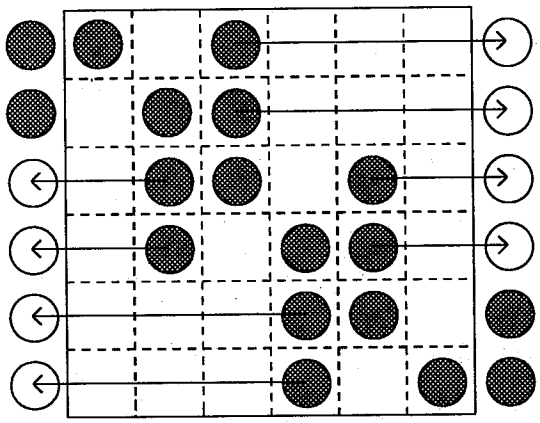

Figure 5: The N-diagonalization process in each processor.

edly applies the *N-diagonalization scheme*, shown in Figure 5, to the reduced system. And a special system, shown in Figure 6, is obtained. Thus, his algorithm can be applied recursively after putting the first and the last equations together in each processor.

On the other hand, after the reduce system shown in Figure 4 is obtained, Chung [6] rearranged these columns of the reduced system to a smaller circulant tridiagonal system, as shown in Figure 7. Thus, Chung's algorithm can also be applied recursively.

3 The Algorithms for the Circulant Tridiagonal Systems and the Tridiagonal Systems

Now, we propose our algorithm for the circulant tridiagonal systems. We partition the n equations evenly into p groups. And, each processor works on $\frac{n}{p}$ consecutive equations, where $m = \frac{n}{p}$ is assumed to be an integer. First, each processor performs the row operations similar to Gaussian elimination. Let $r_i^{(k)}$ be the ith row in the kth processor, $1 \leq i \leq m$ and

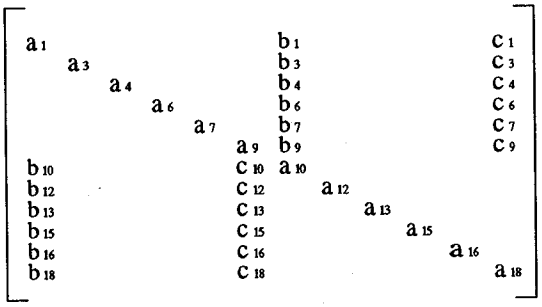

Figure 6: The special system after performing the N-diagonalization scheme.

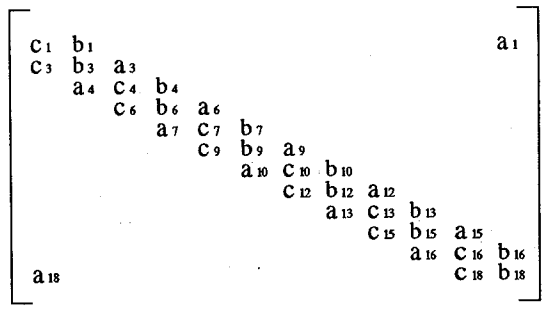

Figure 7: The reduced system after reordering the selected columns.

$1 \leq k \leq p$. 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, for the first processor, the subsystem will become

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

Similarly, for the last processor, the subsystem will be

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

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

$c_{(k-1)m+1}^* x_{(k-1)m} + a_{(k-1)m+1}^* x_{(k-1)m+1}$
$+b_{(k-1)m+1}^* x_{km} = d_{(k-1)m+1}^*$,
$c_{(k-1)m+i}^* x_{(k-1)m+1} + a_{(k-1)m+i}^* x_{(k-1)m+i}$
$+b_{(k-1)m+i}^* x_{km} = d_{(k-1)m+i}^*$, for $2 \leq i \leq (m-1)$,

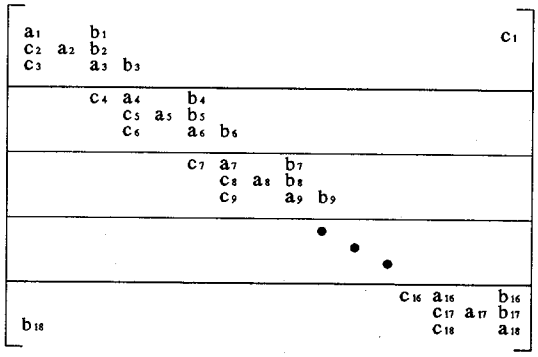

Figure 8: The state after executing our algorithm.

$$c^*_{km}x_{(k-1)m+1} + a^*_{km}x_{km} + b^*_{km}x_{km+1} = d^*_{km}.$$

As an example, let $n = 18$, $p = 6$ and $m = \frac{n}{p} = 3$. The result after the above row operations is shown in Figure 8. Next, we put the first and the last equations together in each processor, then the subsystem consisting of these equations is still a circulant tridiagonal system with smaller size, called the reduced system, as follows:

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

The reduced system, consisting of $2p$ equations, 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 reduced system of the above example is shown in Figure 9. Note that the reduced system in our algorithm is still a circulant a tridiagonal system, thus we do not need to reorder the columns of the reduced system. Therefore, our algorithm is simpler than Chung's algorithm, in which column reordering is needed in the reduced system.

In fact, when we implement our algorithm to solve a circulant system on the hypercube, processor $2k$ sends its first and last equations to processor $2k-1$, for $1 \le k \le \frac{q}{2}$. Thus, we can implement our algorithm recursively on the hypercube (See Figure 10).

Further, we can also implement our algorithm on the binary tree structure (See Figure 11). The nodes are labelled by the preorder sequence. We partition the n equations evenly into p groups. And, we assign group

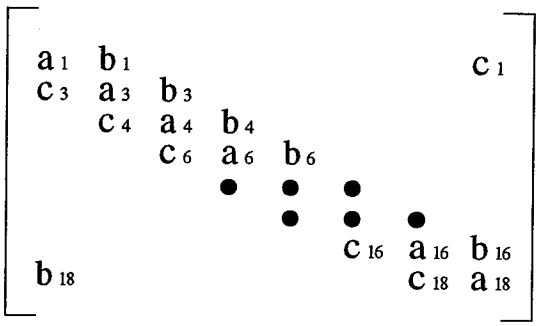

Figure 9: The reduced system in our algorithm.

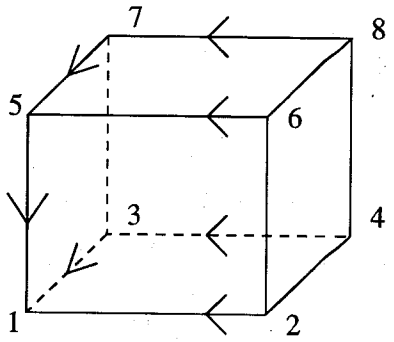

Figure 10: Performing our algorithm on the 3-cube.

i to processor i. Thus, in the Figure 11, at the step 1, processors 4 and 5 send their first and last equations to their parent, processor 3. And, at the same time, processors 7 and 8, processors 11 and 12 processors 14 and 15 also do the the same action. At the step 2, processors 3 and 6 send their first and last equations to processor 2 and, at the same time, processors 10 and 13 send their first and last equations to processor 9. At the step 3, processors 2 and 9 send their first and last equations to processor 1. Thus, our algorithm can be implemented on the binary tree.

The time complexity of our algorithm executed on the hypercube is the complexity for solving the reduced

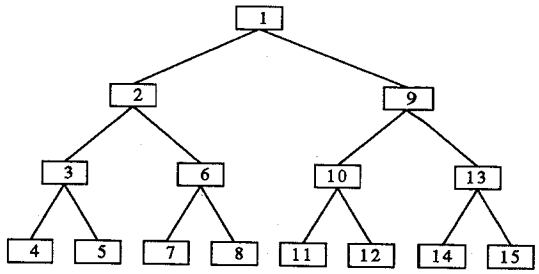

Figure 11: The numbers in this figure representing the nodes labels.

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 $O(\frac{n}{p})$ time. And, the time complexity of the reduced system can be obtained by the following recursive formula:

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

where c is a constant. We can obtain the time complexity of the reduced system is $T(2p) = O(\log p)$. Thus, the time complexity of our algorithm is $O(\frac{n}{p}) + T(2p) = O(\frac{n}{p} + \log p)$. If $p \ll n$, then our algorithm requires $O(\frac{n}{p})$ 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 the 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, it performs 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
$r_i^{(m)} = r_i^{(m)} - (b_{(n-1)m+i-1}/a_{(n-1)m+i})$

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

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

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

And, the subsystem in the last processor will become

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

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

$c_{(k-1)m+1}^* x_{(k-1)m} + a_{(k-1)m+1}^* x_{(k-1)m+1}$
$+b_{(k-1)m+1}^* x_{km} = d_{(k-1)m+1}^*$,
$c_{(k-1)m+i}^* x_{(k-1)m+1} + a_{(k-1)m+i}^* x_{(k-1)m+i}$
$+b_{(k-1)m+i}^* x_{km} = d_{(k-1)m+i}^*$, for $2 \le i \le (m-1)$,
$c_{km}^* x_{(k-1)m+1} + a_{km}^* x_{km} + b_{km}^* x_{km+1} = d_{km}^*$.

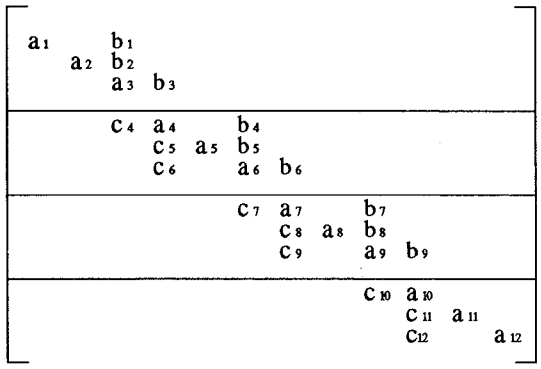

Figure 12: The matrix after performing row operations.

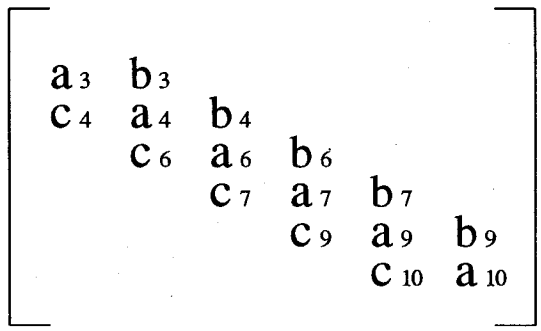

Figure 13: The reduced system which is still a tridiagonal system.

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 subsystem, consisting of the $(2p-2)$ equations, is still a tridiagonal system. The reduced system is as follows:

$a_m^* x_m + b_m^* x_{m+1} = d_m^*$,
$c_{(k-1)m+1}^* x_{(k-1)m} + a_{(k-1)m+1}^* x_{(k-1)m+1}$
$+b_{(k-1)m+1}^* x_{km} = d_{(k-1)m+1}^*$, for $2 \le k \le (p-1)$,
$c_{km}^* x_{(k-1)m+1} + a_{km}^* x_{km} + b_{km}^* x_{km+1}$
$= d_{km}^*$, for $2 \le k \le (p-1)$,
$c_{n-m+1}^* x_{n-m} + a_{n-m+1}^* x_{n-m+1} = d_{n-m+1}^*$.

The reduced system still can be solved recursively. For example, suppose $n = 12$, $p = 4$ and $m = \frac{n}{p} = 3$. After performing Gaussian elimination in each processor, the matrix becomes the form shown in Figure 12. And, the final reduced system is shown in Figure 13.

5 Circulant Block-tridiagonal Systems and Block-tridiagonal Systems

In this section, we present how to generalize our solver for circulant block-tridiagonal systems and block-tridiagonal systems. A circulant block-

$$A=\begin{bmatrix} A_1 & B_1 & & & & C_1 \\ C_2 & A_2 & B_2 & & & \\ & C_3 & A_3 & B_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & & & B_{n-1} \\ B_n & & & & C_n & A_n \end{bmatrix}$$

Figure 14: The form of the circulant block-tridiagonal matrix.

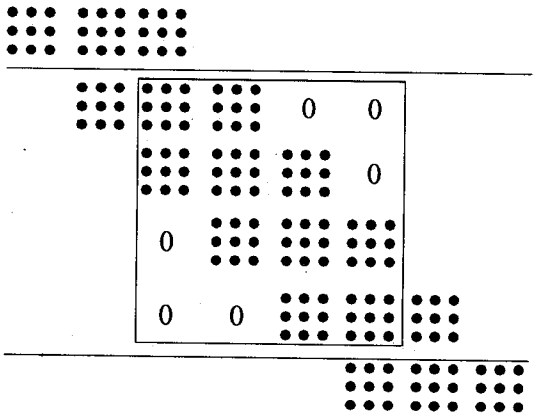

Figure 15: The state in each processor.

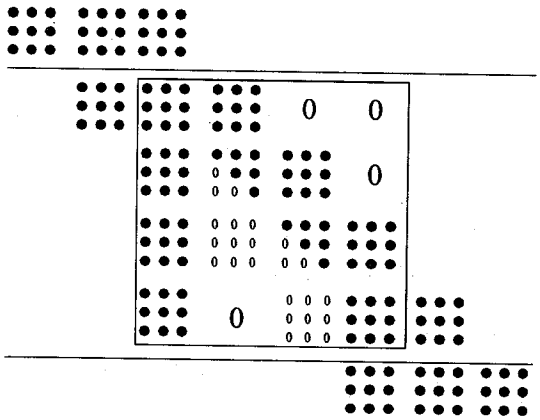

Figure 16: The state after performing the first do-loop.

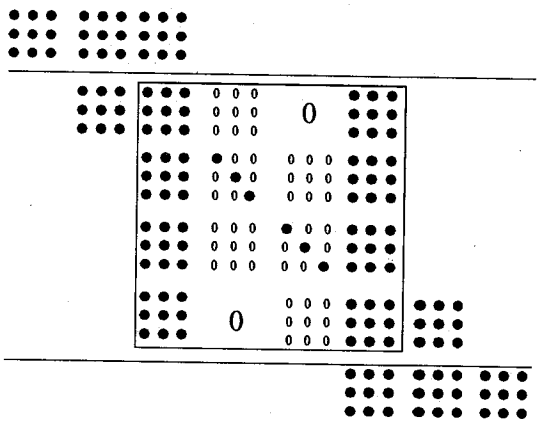

Figure 17: The state after performing the second do-loop.

tridiagonal system **Ax=d** is of the form shown in Figure 14. And, A_i, B_i, and C_i, $1 \leq i \leq n$, denote $q \times q$ submatrices. When our algorithm for circulant tridiagonal systems performs Gaussian elimination, the index i starts at 2 in the first do-loop and starts at $m-1$ in the second do-loop. Similarly, if each block is a $q \times q$ submatrix, the index i starts at $q+1$ in the first do-loop and starts at $(m-1)q$ in the second do-loop. For example, suppose $q = 3$ and $m = 4$. And, the state in each processor is shown in Figure 15. Then, after performing the first do-loop, the state is shown in Figure 16. And, after performing the second do-loop, the state is shown in Figure 17. If we put together the first q and the last q equations in each processor, the reduced system is still a circulant block-tridiagonal system, with smaller size, and can be solved recursively. Therefore, our algorithm can also solve circulant block-tridiagonal systems. Similar to the previous section, with slight modification, our algorithm for solving the circulant block-tridiagonal systems decreases the size of the reduced system when it solves block-tridiagonal systems.

6 Performance of Our Algorithm

In this section, we shall show that Chung's algorithm [6] is superior to Agui's [1] and our algorithm is superior to Chung's. Agui's parallel algorithm only solves the tridiagonal systems. After performing the N-diagonalization scheme, Agui's algorithm still uses the N-diagonalization scheme to maintain the reduced system a special system so that his algorithm can be performed recursively. However, the N-diagonalization scheme is time-consuming. In fact, if the size of the matrix is n, performing the N-diagonalization needs $O(n^2)$ row operations. Thus, the time complexity of Agui's algorithm is $O(\frac{n^2}{p^2} + \log p)$. On the other hand, based on the same partitioning scheme and the same Gaussian elimination, Chung's algorithm maintains the reduced system to be a circulant tridiagonal system by reordering the corresponding columns and uses Gaussian elimination instead of the N-diagonalization scheme. Note that doing the two do-loops needs only $O(n)$ row operations. Because Chung's algorithm needs only two do-loops, Chung's algorithm is simpler than Agui's. And, the time complexity of Chung's algorithm is $O(\frac{n}{p} + \log p)$. Clearly, Chung's algorithm is superior to Agui's. Next, the row operations invoked in our algorithm is somewhat different from Chung's and Agui's. Our algorithm performs fewer operations than Chung's and Agui's do. And, in this way, we can keep the reduced system to be a circulant tridiagonal system (without the N-diagonalization scheme or columns reordering). Besides, in the back-substitution process, our algorithm performs the same operations as the Chung's to find the final answer. The analysis of the time complexity is as follows:

For our algorithm,

Gaussian elimination: $\frac{2n}{p} - 4$ row operations

reduced system: $T(2p) = T(p) + 6$ and $T(4) = 1$

back-substitution: $\frac{n}{p} - 2$ row operations

Thus, the number of row operations required for our algorithm executed in parallel is

$\frac{3n}{p} + 6\lg p - 11$

For Chung's algorithm,

Gaussian elimination: $\frac{2n}{p} - 2$

reduced system: $T(2p) = T(p) + 11$ and $T(4) = 1$

back-substitution: $\frac{n}{p} - 2$

Thus, the Chung's is

$\frac{3n}{p} + 11\lg p - 14$

Clearly, our algorithm is superior to Chung's. In addition, our algorithm can also solve the circulant block-tridiagonal systems. It reduces more operations in Gaussian Elimination. In fact, the time complexity of our algorithm is $O(\frac{nq}{p} + q^2 \log p)$, where q is the size of the block. And, the time complexity of Chung's algorithm is $O(\frac{n^2}{p^2} + q^2 \log p)$. Thus, our algorithm for the circulant block-tridiagonal systems is further superior to Chung's.

7 Conclusion

We have presented two algorithms proposed by Agui [1] and Chung [6]. Agui [1] proposed a parallel algorithm to solve the tridiagonal systems. And, Chung [6] proposed a parallel algorithm to solve the circulant block-tridiagonal systems. However, the *N-diagonalization scheme* used by Agui's algorithm needs $O(n^2)$ row operations. And, although Chung's algorithm only does two do-loops, which needs $O(n)$ row operations, his algorithm must reorder the *reduced system* in order to apply his algorithm recursively. And, though our algorithm also performs two do-loops, our algorithm not only has fewer row operations than Agui's and Chung's but also needs not reorder the columns of the matrix. Thus, our algorithm is superior to Agui's and Chung's. Besides, our algorithm not only can solve the circulant tridiagonal systems but also can be used to solve the tridiagonal systems with slight modification to decrease the order of the reduced system. Further, we generalize them to solve the circulant block-tridiagonal systems and the block-tridiagonal systems. And, our algorithm for the circulant block-tridiagonal systems is further superior to Chung [6]. Since a banded system is a special case of the block-tridiagonal systems, it can also be solved by using our generalized solver for solving the block-tridiagonal systems. For the above solvers, since they are based on the divide-and-conquer method, they are appropriate for executing on the hypercube and on the binary tree.

References

[1] J. C. Agui and J. Jimenez, "A binary tree implementation of a parallelf distributed tridiagonal solver," *Parallel Computing*, Vol. 21, pp. 233–241, 1995.

[2] M. Angelaccio and M. Colajanni, "Subcube matrix decomposition: A unifying view for LU factorization on multicomputers," *Parallel Computing*, Vol. 20, pp. 257–270, 1994.

[3] E. Bampis and J. C. Konig, "Impact of communications on the complexity of the parallel gaussian elimination," *Parallel Computing*, Vol. 17, pp. 55–61, 1991.

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

[5] J. Carl D. Meyer and R. J. Plemmons, "Convergent powers of a matrix with applications to iterative methods for singular linear systems," *SIAM Journal on Numerical Analysis*, Vol. 14, No. 4, pp. 699–705, Sep. 1977.

[6] K. L. Chung, Y. H. Tsai, and W. M. Yan, "A parallel solver for circulant block-tridiagonal systems," *Computers & Mathematics with Applications*, Vol. 29, No. 1, pp. 109–113, 1995.

[7] D. J. Evans, "Parallel S.O.R. iterative methods," *Parallel Computing*, Vol. 1, pp. 3–18, 1984.

[8] D. J. Evans and R. Abdullah, "The parallel implicit elimination(pie) method for the solution of linear systems," *Parallel Algorithms and Applications*, Vol. 4, pp. 153–162, 1994.

[9] E. Galligani and V. Ruggiero, "A polynomial preconditioner for block tridiagonal metrices," *Parallel Algorithms and Applications*, Vol. 3, pp. 227–237, 1994.

[10] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing.* New York, USA: McGraw-Hill Book Company, 1984.

[11] A. H. Karp and J. Greenstadt, "An improved parallel Jacobi method for diagonalizing a symmetric matrix," *Parallel Computing*, Vol. 5, pp. 281–294, 1987.

[12] U. Meier, "A parallel partition method for solving banded systems of linear equations," *Parallel Computing*, Vol. 2, pp. 33–43, 1985.

[13] P. H. Michielse and H. A. V. der Vorst, "Data transport in Wang's partition method," *Parallel Computing*, Vol. 7, pp. 87–95, 1988.

[14] K. S. Rajasethupathy, G.-M. Thio, S. K. Dhall, and S. Lakshmivarahan, "Tridiagonalizing a real symmetric matrix: A parallel direct approach using Given's transformation," *Parallel Algorithms and Applications*, Vol. 2, pp. 305–313, 1994.

[15] R. A. Sweet, "A cyclic reduction algorithm for solving block tridiagonal systems of arbitrary dimension," *SIAM Journal on Numerical Analysis*, Vol. 14, No. 4, pp. 706–719, Sep. 1977.

[16] X. Wang and Z. G. Mou, "A divide-and-conquer method of solving tridiagonal systems on hypercube massively parallel computers," *IEEE Computer Society*, pp. 810–817, 1991.