第23卷 第4期 上海师范大学学报(自然科学版) Vol. 23, No. 4
1994年12月 J. of Shanghai Teachers Univ. (Natural Sciences) Dec., 1994

PCR Algorithm for the Parallel Computation of the Solution of a Class of Singular Equations*

Wang Guorong *Wei Yimin*
(Department of Mathematics)

Abstract This paper presents a new highly parallel algorithm for computing the solution of a class of singular equations $Ax = b (A \in R^{n\times n}, \mathrm{Ind}(A) = k, b \in R(A^k))$. By this algorithm the solution $x = A_d b$ is obtained in $T = (1 + n + \lceil \log_2 k \rceil n)(1 + \log_2 n) + n(\log_2(n - r + 1) + 4)$ steps with $p = 2n(n - 1)$ processors.

Keywords parallel algorithm; singular equations; index; Drazin inverse; time complexity

1 Introduction

Let $A \in R_n^{n\times n}, b \in R^n$. Then the unique solution $x = A^{-1}b$ of the nonsingular equation

$$Ax = b \tag{1.1}$$

is given, componentwise, by

$$x_i = \det A_i / \det A_{n+1} (i = 1, 2, \ldots, n) \tag{1.2}$$

where

$$A_{n+1} = A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, A_i = \begin{bmatrix} a_{11} & \cdots & a_{1,i-1} & b_1 & a_{1,i+1} & \cdots & a_{1n} \\ a_{21} & \cdots & a_{2,i-1} & b_2 & a_{2,i+1} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n1} & \cdots & a_{n,i-1} & b_n & a_{n,i+1} & \cdots & a_{nn} \end{bmatrix} \tag{1.3}$$

and $A_i (i = 1, 2, \ldots, n + 1)$ are all identical, except in one column. (1.2) is called Cramer's Rule.

Cramer's Rule is abandoned due to its inefficiency on serial processors. A highly parallel algorithm for the solution $x = A^{-1}b$ of the nonsingular equation (1.1) is presented in [1]. This algorithm is called the Parallel Cramer's Rule (PCR). An elimination method akin to that used in Gaussian Elimination (GE) is used in the PCR Algorithm.

M. K. Sridhar[4] shows that PCR Algorthm obtains solution in n steps with no more than $2n(n-1)$ processors. The parallelism in the algorithm is analysed under the assumption of an unbounded parallel computational model and issues relating to interprocessor communication and

Received: Sept., 12, 1993

* This project is supported by the National Natural Science Foundation of China.

task scheduling have not been considered.

The PCR Algorithm was tested by a number of systems of linear equations and was found to exhibit stability and accuracy identical to Parallel Gaussian Elimination (PGE)[5,7]. Since the PCR Algorithm provides approximately twice the speedup over the PGE with only twice the number of processors, it offers exciting possibilities for VLSI implementation as well as MIMD parallel processing structures.

In [9], G. R. Wang gave a PCR algorithm for the parallel computation of the minimum-norm least-squares solution of inconsistent linear equations $Ax = b, A \in R_r^{m\times n}, b \notin R(A)$.

Let $A \in R^{n\times n}$. The smallest nonnegative integer k such that

$$\text{rank } A^k = \text{rank } A^{k+1} \tag{1.4}$$

is called the index of A, and is denoted by $\text{Ind}(A)$.

The concept of the Drazin inverse is defined as follows:

Let $A \in R^{n\times n}$ with $\text{Ind}(A) = k$, and $X \in R^{n\times n}$ be such that

$$A^{l+1}X = A^l, \qquad XAX = X, \qquad AX = XA. \tag{1.5}$$

Then X is called the Drazin inverse of A, and is denoted by $X = A_d$. In particular, when $\text{Ind}(A) = 1$, the matrix X satisfying (1.5) is called the group inverse of A, and is denoted by $X = A^{\#}$.

A_d has the following property[1].

Lemma 1 Let $A \in R^{n\times n}$ with $\text{Ind}(A) = k, l \geqslant k$. Then

(1) $R(A_d) = R(A^l), N(A_d) = N(A^l)$;

(2) $R(A_d) \oplus N(A_d) = R^n$;

(3) $A_dA = AA_d = P_{R(A_d),N(A_d)} = P_{R(A^l),N(A^l)}$;

(4) $A_d = A^l(A^{2l+1})^{(1)}A^l, (A^{2l+1})^{(1)} \in (A^{2l+1})\{1\}$.

2 Algorithm

Let $A \in R^{n\times n}$. For a given $b \in R^k$, find a vector $x \in R(A^k)$ such that

$$Ax = b \qquad (\text{Ind}(A) = k). \tag{2.1}$$

We see that the above problem has a unique solution[8], which is

$$x = A_db. \tag{2.2}$$

For any $A \in R^{n\times n}, x \in R^n, A(i \to x)$ denotes the matrix obtained by replacing the *i-th* column of A with x. $R(A)$ and $N(A)$ denote the range and the null space of A.

Cramer's Rule for computing the unique solution of (2.1) is given in [8].

Theorem 1 Let $A \in R^{n\times n}$ with $\text{Ind}(A) = k$, $\text{rank}A^k = r < n$, and let $U, V^T \in R_{n-r}^{n\times(n-r)}$ be matrices whose columns form the basis for $N(A^k)$ and $N[(A^k)^T]$ respectively. Then the unique solution x of (2.1) is given, componentwise, by

$$x_i = \det D_i/\det D_{n+1}, \qquad (i = 1,2,\ldots,n) \tag{2.3}$$

where

$$D_{n+1} = D = \begin{bmatrix} A & U \\ V & O \end{bmatrix}, D_i = \begin{bmatrix} A(i \to b) & U \\ V(i \to O) & O \end{bmatrix} \tag{2.4}$$

and $x = A_db$ satisfies

$$x \in R(A^k), N(V) = R(A^k). \tag{2.5}$$

By (2.5), we have

$$Vx = 0. \tag{2.6}$$

It follows from (2.6) that the unique solution x of (2.1) satisfies

$$\begin{bmatrix} A & U \\ V & O \end{bmatrix} \begin{bmatrix} x \\ O \end{bmatrix} = \begin{bmatrix} b \\ O \end{bmatrix}, \tag{2.7}$$

This is a nonsignular linear equation. If we also use the PCR Algorithm to compute the solution of (2.7), then the steps and the number of the processors depend on the order $2n - r$ of the coefficient matrix of (2.7). When $n >> r$, the cost will be expensive. A condensed Cramer's Rule for the unique solution of (2.1) is presented as follows[7]

Theorem 2 Let $A \in R^{n\times n}$ with $\mathrm{Ind}(A) = k$, rank $A^k = r < n$, $b \in R^k$, and $V \in R^{n\times(n-r)}_{n-r}$ be a matrix whose columns form the basis for $N[(A^k)^T]$. We define

$$B = VV^T. \tag{2.8}$$

The solution x of the nonsingular equation

$$(A^TA + B)X = A^Tb \tag{2.9}$$

is the unique solution of (2.1). Let

$$C = A^TA + B \in R^{n\times n}, \qquad d = A^Tb \in R^n. \tag{2.10}$$

Then x is given, componentwise, by

$$x_i = \det C_i / \det C_{n+1} \qquad (i = 1,2,\ldots,n), \tag{2,11}$$

where

$$C_{n+1} = C = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix}, \quad C_i = \begin{bmatrix} c_{11} & \cdots & c_{1,i-1} & d_1 & c_{1,i+1} & \cdots & c_{1n} \\ c_{21} & \cdots & c_{2,i-1} & d_2 & c_{2,i+1} & \cdots & c_{2n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ c_{n1} & \cdots & c_{n,i-1} & d_n & c_{n,i+1} & \cdots & c_{nn} \end{bmatrix} \tag{2.12}$$

Proof See [7].

Using Cramer's Rule to (2.9), we obtain (2.11) immediately.

By Theorem 2, a highly parallel algorithm for the unique solution of (2.1) is given as follows.

We assume, for the convenience of exposition, that the orders of the linear equations can be expressed as $n = 2^l$, where l is an integer. Later, we shall see that the algorithm is valid for arbitrary n.

First of all, we need an algorithm for computing the basis of $N(A)$ [3]

A matrix $H \in R^{n\times n}$ is said to be in Hermite echelon form if its elements h_{ij} satisfy the following conditions:

(1) $h_{ij} = 0, i > j$.

(2) h_{ij} is either 0 or 1.

(3) if $h_{ii} = 0$ then $h_{ik} = 0$ for every k, $l \le k \le n$.

(4) if $h_{ii} = 1$ then $h_{ki} = 0$ for every $k \neq i$.

For a given matrix $A \in R^{n\times n}$, the Hermite echelon form H_A obtained by row reducing A is unique; $N(A) = N(H_A) = R(I - H_A)$ and a basis for $N(A)$ is the set of non-zero columns of $I - H_A$.

Algorithm 1 Let $A \in R_r^{n\times n}$. This algorithm computes $U \in R_{n-r}^{n\times(n-r)}$ whose columns form the basis for $N(A)$.

(1) Row reduce A to its Hermite echelon form H_A.

(2) Form $I - H_A$, and select the non-zero columns $u_1, u_2, \ldots, u_{n-r}$ from this matrix, $U = (u_1, u_2, \ldots, u_{n-r})$.

The following is the PCR Algorithm for computing the unique solution of (2.1). Notation. $\lceil x \rceil$ is the integer such that $x \leq \lceil x \rceil < x + 1$; $\lfloor x \rfloor$ is the integer such that $x - 1 < \lfloor x \rfloor \leq x$.

Algorithm 2 Let $A \in R^{n\times n}$ with $\mathrm{Ind}(A) = k$, $\mathrm{rank}A^k = r < n$, $b \in R^n$. This algorithm computes $x = A_d b$.

(1) Compute $d = A^T b$ in parallel.

(2) Compute $C = A^T A$ in parallel.

(3) Compute A^{2^w} in parallel. ($w = \lceil \log_2 k \rceil$)

(4) Use Algorithm 1 to compute $v = (v_1, v_2, \ldots, v_{n-r})$ whose colulms form the basis for $N[(A^{2^w})^T]$ in parallel. (see Lemma 1(1))

(5) Compute $C \leftarrow C + VV^T$ in parallel.

(6) Form L-form and R-form matrices

$$LC = (d \vdots C), \qquad RC = (C \vdots d)$$

LC and RC differ only in the position of the d vector, which appears on the left-hand in LC and on the right-hand side in RC.

We shall triangulate LC in parallel by subtracting fractions of the pivotal column from c_{nn} as the first pivot, until all rows to the left of pivot $c_{kk}(k = n/2 + 1)$ have been reduced to zero.

$$LC \rightarrow \begin{bmatrix} d_1 & c_{11} & \cdots & c_{1,n/2} & c_{1,n/2+1} & \cdots & c_{1,n} \\ \cdots & \cdots & & \cdots & \cdots & \cdots & \\ d_{n/2} & c_{n/2,1} & \cdots & c_{n/2,n/2} & c_{n/2,n/2+1} & \cdots & c_{n/2,n} \\ 0 & 0 & \cdots & 0 & c_{n/2+1,n/2+1} & \cdots & c_{n/2+1,n} \\ \cdots & \cdots & & & \cdots & & \\ 0 & 0 & \cdots & 0 & 0 & \cdots & c_{n,n} \end{bmatrix}$$

And we shall traiangulate RC in parallel by subtracting fractions of the pivotal row from rows below this row, starting from c_{11} as the first pivot until pivot $c_{kk}(k = n/2)$ is reached, leaving a submatrix of order $(n/2) \times (n/2 + 1)$ below this pivotal row.

$$RC \rightarrow \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1,n/2} & c_{1,n/2+1} & \cdots & c_{1,n} & d_1 \\ 0 & c_{22} & \cdots & c_{2,n/2} & c_{2,n/2+1} & \cdots & c_{2,n} & d_2 \\ 0 & & \cdots & \cdots & \cdots & \cdots & & \\ \cdots & & \cdots & c_{n/2,n/2} & c_{n/2,n/2+1} & \cdots & c_{n/2,n} & d_{n/2} \\ \cdots & & \cdots & 0 & c_{n/2+1,n/2+1} & \cdots & c_{n/2+1,n} & d_{n/2+1} \\ \cdots & & \cdots & \cdots & \cdots & \cdots & & \\ 0 & \cdots & \cdots & 0 & c_{n,n/2+1} & \cdots & c_{n,n} & d_n \end{bmatrix}$$

In either case these operations are of the general form

$$c_{ij} \leftarrow c_{ij} - c_{ik}c_{kj}/c_{kk}$$

(7) Form new L-form and R-form submatrices of order $(n/2) \times (n/2+1)$ in LC and RC that have not been triangulated.

$$LLC = \begin{bmatrix} d_1 & c_{11} & \cdots & c_{1,n/2} \\ \cdots & \cdots & \cdots & \cdots \\ d_{n/2} & c_{n/2,1} & \cdots & c_{n/2,n/2} \end{bmatrix}, \quad RRC = \begin{bmatrix} c_{n/2+1,n/2+1} & \cdots & c_{n/2+1,n} & d_{n/2+1} \\ \cdots & \cdots & \cdots & \\ c_{n,n/2+1} & \cdots & c_{n,n} & d_n \end{bmatrix}$$

and form two corresponding L-form and R-form matrices

$$RLC = \begin{bmatrix} c_{11} & \cdots & c_{1,n/2} & d_1 \\ \cdots & \cdots & \cdots & \cdots \\ c_{n/2,1} & \cdots & c_{n/2,n/2} & d_{n/2} \end{bmatrix}, \quad LRC = \begin{bmatrix} d_{n/2+1} & c_{n/2+1,n/2+1} & \cdots & c_{n/2+1,n} \\ \cdots & \cdots & \cdots & \\ d_n & c_{n,n/2+1} & \cdots & c_{n,n} \end{bmatrix}$$

As before, LLC, LRC, RLC and RRC are triangulated in parallel.

Form new L-form and R-form submatrices $LLLC$, $LLRC$, $RRLC$ and $RRRC$ of order $(n/4) \times (n/4+1)$ in LLC, LRC, RLC and RRC that have not been triangulated and form four corresponding L-form and R-form matrices $RLLC$, $RLRC$, $LRLC$ and $LRRC$.

The algoithm therefore recursively doubles the number of submatrices, halving their order in every step. Since, by our assumption; $n = 2^l$, at the l-1 step, we form a submatrices of order 2×3.

(8) n submatrices of order 2×3 are triangulated in parallel. We take c_{ii} and d_i from the above n submatrices of order 2×3. Then

$$x_i = d_i/c_{ii} \qquad (i = 1, 2, \ldots, n).$$

Example Let

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \in R_3^{4\times 4}, \mathrm{Ind}(A) = 2, \mathrm{rank}(A^2) = 2, b = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \in R(A^2),$$

(1) $d = A^T b = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$,

(2) $C = A^TA = \begin{bmatrix}1&0&0&0\\0&0&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}$,

(3) $A^2 = \begin{bmatrix}1&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&1\end{bmatrix}$,

(4) $(A^2)^T = A^2 = H_{A^2}, I - H_{A^2} = \begin{bmatrix}0&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&0\end{bmatrix}$, $V = \begin{bmatrix}0&0\\1&0\\0&1\\0&0\end{bmatrix}$,

(5) $C \leftarrow C + VV^T = \begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&2&0\\0&0&0&1\end{bmatrix}$,

(6) $LC = \begin{bmatrix}1&1&0&0&0\\0&0&1&0&0\\0&0&0&2&0\\1&0&0&0&1\end{bmatrix} \rightarrow \begin{bmatrix}1&1&0&0&0\\0&0&1&0&0\\0&0&0&2&0\\0&0&0&0&1\end{bmatrix}$,

$RC = \begin{bmatrix}1&0&0&0&1\\0&1&0&0&0\\0&0&2&0&0\\0&0&0&1&1\end{bmatrix}$,

(7) $LLC = \begin{bmatrix}1&1&0\\0&0&1\end{bmatrix}$, $LRC = \begin{bmatrix}0&2&0\\1&0&1\end{bmatrix}$,

$RLC = \begin{bmatrix}1&0&1\\0&1&0\end{bmatrix}$, $RRC = \begin{bmatrix}2&0&0\\0&1&1\end{bmatrix}$

(8) $x_1 = 1/1 = 1, x_2 = 0/1 = 0, x_3 = 0/2 = 0, x_4 = 1/1 = 1$.

3 Complexity

In this section, we discuss the parallel arithemtic complexity of the PCR Algorithm for computing the unique solution of (2.1).

Theorem 3 By the PCR Algorithm, the unique solution of (2.1) is obtained in $T = (1 + \log_2 n)(1 + n + n\lceil \log_2 k \rceil) + n(\log_2(n - r + 1) + 4)$ steps with $p = 2n(n - 1)$ processors.

Proof (1) Parallel computation of $d = A^Tb$ takes $T_1 = 1 + \log_2 n$ steps and $p_1 = n^2$ processors.

(2) Parallel computation of A^TA takes $T_2 = n(1 + \log_2 n)$ steps and $p_2 = n^2$ processors.

(3) Parallel computation of A^{2^w} takes $T_3 = \lceil \log_2 k \rceil n (1 + \log_2 n)$ steps and $p_3 = n^2$ processors. ($w = \lceil \log_2 k \rceil$)

(4) Parallel computation of V takes $T_4 = 2n$ steps and no more than $p_4 = (n - 1)^2$ proces-

sors.

(5) Parallel computation of $C \leftarrow C + VV^T$ takes $T_5 = n(1 + \log_2(n - r + 1))$ steps and $p_5 = n(n - r + 1)$ processors.

(6)—(7) Recursive parallel triangulation of L-form and R-form matrices takes $T_6 = n - 1$ steps and $p_6 = 2n(n - 1)$ processors[4].

(8) Parallel computation of $x_i = d_i/c_{ii} (i = 1, 2, \ldots, n)$ takes $T_7 = 1$ steps and $p_7 = n$ processors.

Thus $T = \sum_{(i=1)}^{7} T_i = (1 + \log_2 n)(1 + n + n\lceil \log_2 k \rceil) + n(\log_2(n - r + 1) + 4)$ steps and

$$p = \max p_i = 2n(n - 1).$$

Note. The recursive technique in steps (6)—(7) of the PCR Algorithm can be represented by a binary tree shown in Fig. 1.

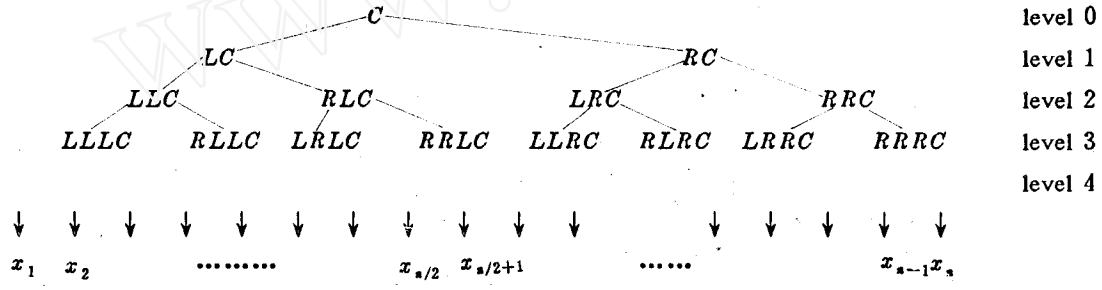

Fig. 1

Here each node represents the task unit defined in step (6)—(7). All tasks at one level can performed in parallel[2]. Since, by our assumption, $n = 2^l$, there are $\log_2 n = l$ levels in the tree.

If we relax the assumption that $n = 2^l$ and choose an arbitrary n, we observe that recursive doubling can still take place. However, the orders of the submatrices will have to be carefully taken into account due to the possibility of there being odd and even ordered matrices produced from an odd matrix. Naturally, the tree becomes unsymmetrical.

The case for an arbitrary n is illustrated with $n = 5$ in Fig. 2. Let LC and RC be 5×6 matrices, LLC and RLC be 3×4 matrix, RRC and LRC be 2×3 matrix. Then the tree becomes unsymmetrical.

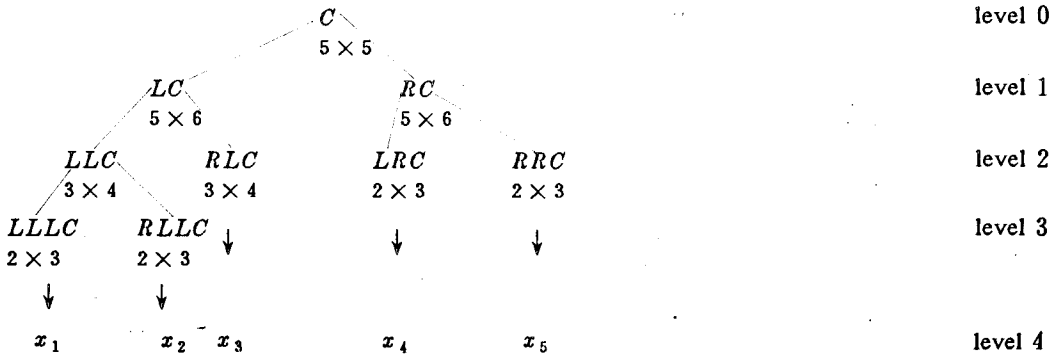

Fig. 2

In general, if there exists a submatrix of order 3×4 at the $\lceil \log_2 n \rceil - 1$ level, then triangulation requires one additional level where two 2×3 submatrices are formed out of the above submatrix. Each of these produces two solutions. Hence, when n is arbitrary, the number of steps remains n.

The problem relating to pivoting in the PCR Algorithm is discussed in [4]. It is omitted here.

Remark If A is nonsingular, the conclusion reduces to the results in [4].

References

[1] A. Ben-Isreal & T. N. E Greville, Generalized Inverses: Theory and Applications, Wiley-Interscience, New York, 1974

[2] D. Heller, A survey of parallel algorithms in numerical linear algebra, *SIAM Rev.*, 1978, 20: 740—777

[3] B. Noble, Applied Linear Algebra, Prentice-Hall, New Jersey, 1969

[4] M. K. Sridhar, A new algorithm for parallel solution of linear equations, Information Processing Letters, 1987, 24: 407—412

[5] M. A. Srinivas, Optimal parallel scheduling of Gaussing eliminations DAG's, *IEEE Trans. Comput.*, 1983, C-32: 109—117

[6] O. Wing & J. W. Huang, A Computational model for parallel solution of linear equations, *IEEE Trans. Comput.*, 1980, C-29: 632—638

[7] Chen Yonglin, Explicit expression and Cramer rule for solution of restricted linear equations, Applied Mathematics, *A Journal of Chinese Universities*, 1993, 8: 61—70 (in Chinese)

[8] Wang Guorong, A Cramer Rule for finding the solution of a class of singular equations, *Linear Algebra & Appl.*, 1989, 116: 27—34

[9] Wang Guorong, PCR algorithm for parallel computing minimum-norm least-squares solutions of inconsistent linear equations, Numer. Math., *A J. Chinese Universities.* (English Series), 1993, 1(2): 1—10

并行计算一类奇异方程组的 PCR 算法

王国荣 魏益民

(数学系)

提 要 本文给出了一个计算奇异方程组 $Ax = b(A \in R^{n\times n}, \mathrm{Ind}(A) = k, b \in R(A^k))$ 的新的高度并行算法. 通过该算法可以在时间 $T = (1 + n + \lceil \log_2 k \rceil n)(1 + \log_2 n) + n(\log_2(n - r + 1) + 4)$ 步内, 用 $p = 2n(n - 1)$ 台处理机得到方程组的解 $x = A_d b$.

关键词 并行算法; 奇异方程组; 指标; Drazin 逆; 时间复杂性

中图法分类号 O151.21