International Journal of Operations Research Vol. 1, No. 1, 59−60 (2004)

A Parallel Algorithm for Power Matrix Computation

JrJung Lyu[1,*], Ming-Chang Lee[2]

[1]Department of Industrial and Information Management, National Cheng Kung University, Tainan, Taiwan, ROC

[2]Department of Information Management, Fooyin University, Kaohsiung, Taiwan, ROC

Abstract—We present a parallel algorithm for power matrix A^n in $O(\log^2 n)$ time using $O(n^{2.807}/\log n)$ number of processors. It is shown that the growth rate of the proposed algorithm is the same as the parallel arithmetic complexity of matrix computations, including matrix inversion and solving systems of linear equations.

Keywords—matrix computations, parallel algorithms, computational complexity

1. INTRODUCTION

Matrix computations, such as inversion of a specific matrix, power matrix, solving a linear system of equations, are very important to scientific and engineering computational practice (Padmini, Madan and Jain (1999), Pudlak (2000), Rojo, Soto and Rojo (2000)). Let A be an order n matrix and let $T_I(n)$, $T_E(n)$, $T_D(n)$, $T_P(n)$ and $T_{PW}(n)$ denote the parallel arithmetic complexity of inverting order n matrices, solving a system of n linear equations in n unknowns, finding order n determinants, calculating the characteristic polynomials of order n matrices, and finding the nth order matrix, respectively. In the previous research Csanky (1976), it has shown that $T_I(n)$, $T_D(n)$, $T_E(n)$ and $T_P(n)$ have the same growth rate (order of magnitude) in time $O(\log^2 n)$ using $O(n^4)$ processors.

The purpose of this note is to develop a parallel power matrix algorithm running in $O(\log^2 n)$ time using $O(n^{2.807}/\log n)$ number of processors. In addition, we show that the growth rate of the proposed parallel A^n algorithm, $T_{PW}(n)$, is the same as $T_I(n)$, $T_D(n)$, $T_E(n)$ and $T_P(n)$.

2. THE ALGORITHM

Assume r_i, $i = 1, \ldots, n$, are the n roots of r, where r is any complex number. Since $A^n = A^n - rI + rI = [(A^n - rI)^{-1}]^{-1} + rI$, we can prove that

$$A^n - rI = \prod_{i=1}^{n}(A - r_i I)$$

From Kung (1976), it is shown

$$\prod_{i=1}^{n}(A - r_i I)^{-1} = \sum_{i=1}^{n}\frac{r_i}{nr}(A - r_i I)^{-1} \tag{1}$$

and we thus have

$$\begin{aligned} A^n &= \left[\prod_{i=1}^{n}(A - r_i I)^{-1}\right]^{-1} + rI \\ &= \left[\sum_{i=1}^{n}\frac{r_i}{nr}(A - r_i I)^{-1}\right]^{-1} + rI \end{aligned} \tag{2}$$

We can therefore develop the power matrix algorithm based on its relationship with the matrix inversion algorithm, shown in (2). That is, by improving the complexity of matrix inversion algorithm, we can further reduce the complexity of the power matrix algorithm.

2.1 Parallel algorithm for A^{-1}

The proposed matrix inversion algorithm is based on the combination of Leverrier's method Lakshmivarahan and Dhall (1990) and parallel Strassen's algorithm Paprzycki and Cyphers (1996). Let the characteristic polynomials $\Phi(\lambda)$ of matrix A be denoted by $\det[I - A] = (\lambda^n + c_1\lambda^{n-1} + \ldots + c_{n-1}\lambda + c_n = 0$. Assume $\lambda_1, \lambda_2, \ldots, \lambda_n$ are the roots of $\Phi(\lambda)$. From the theory of similar matrices, it is known that the trace of A, denoted by

$$s_k = \sum_{i=1}^{n}\lambda_i^k ,$$

and

$$(-1)^n c_n = \det[A]$$

and

$$c_n = (-1)^n \prod_{i=1}^{n}\lambda_i .$$

We also can derive the following equations:

$$\begin{pmatrix} 1 & 0 & \cdots & \cdots & \cdots & 0 & 0 \\ s_1 & 2 & 0 & & & & \\ s_2 & s_1 & & & & & \\ \vdots & & & & & & \\ s_{k-1} & & & & & & \\ \vdots & & & & & & \\ s_n & s_{n-1} & \cdots & \cdots & \cdots & s_1 & n \end{pmatrix}\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ c_n \end{bmatrix} = -\begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ s_n \end{bmatrix} \tag{3}$$

We can therefore obtain A^{-1} from the Cayley-Hamilton theorem, i.e.

$$A^{-1} = -\frac{(A^{n-1} + c_1 A^{n-2} + \ldots + c_{n-1} I)}{c_n} \tag{4}$$

* Corresponding author's email: jlyu@mail.ncku.edu.tw

Parallel algorithm A^{-1}

Step 1. Compute $A \sim A^{n-1}$. The parallel Strassen's algorithm performs block matrix multiplication in O(log n) time when using O($n^{2.807}$/log n) processors Grayson and Van De Geijn (1996). Adopting the concept of fan-in algorithm Romine and Ortega (1988), we can calculate $A \sim A^{n-1}$ running in O(log n) O(log n) = O(log^2 n) time
Using

$$O\left(\frac{n}{\log n}\right)O\left(\frac{n^{2.807}}{\log n}\right)=O\left(\frac{n^{3.807}}{\log^2 n}\right)$$

processors.
Step 2. Compute s_i. The traces s_i of A^i, i = 1, 2, ..., n are obtained in O(log n) time using O(n^2) processors.
Step 3. Compute c_i. Using the lower triangular equations (3) to solve the coefficients of characteristic polynomials c_i, i = 1, 2, …, n. This step can be executed in parallel in O(log^2 n) time using O(n^3) processors Sameh and Brent (1977).
Step 4. Calculate A^{-1}. Once c_i and A^i are found, this step can be executed in parallel in O(log n) time using O(n^3) processor Lakshmivarahan and Dhall (1990).

Theorem 1. The parallel algorithm for computation of the inversion of A in O(log^2 n) time using O($n^{3.807}$/log^2 n) processors bound when n >128.

Proof. We establish the theorem by exhibiting an algorithm.

2.2 Parallel algorithm for power matrix

We can then develop the parallel power matrix algorithm.

Parallel algorithm A^n

Step 1. Calculate A - r_iI, i = 1, 2, ..., n. This step can be executed in parallel in O(1) time using O(n^2) processors.
Step 2. Calculate $(A - r_iI)^{-1}$, i = 1, 2, ..., n. The complexity of matrix inversion is $T_I(n)$ as defined before.
Step 3. Calculate $D = \frac{r_i}{nr}(A-r_iI)^{-1}$. This step can be executed in parallel in O(2) time using O(n^3) processors.
Step 4. Calculate $E = \sum_{i=1}^{n} D$. This is the sum of n matrices and can be executed in parallel in O(log n) time using O(n^3) processors.
Step 5. Parallel computation of $F = E^{-1}$. The complexity of matrix inversion is $T_I(n)$ as defined before.
Step 6. Calculate $A^n = F + rI$. The sum of two matrices can be executed in parallel in O(1) time using O(n^2) processors.

3. COMPLEXITY AND GROWTH RATE

We now derive the complexity of $T_E(n)$, $T_D(n)$, $T_P(n)$ and $T_{PW}(n)$.

Theorem 2. The computing time $T_E(n)$, $T_D(n)$, and $T_P(n)$ of A are O(log^2 n) with O(n$^{3.807}$/log^2 n) number of processors bound when n >128.

Proof.

(a) Using finite steps of row operation and O(n^2) processors, we can transfer $Ax = b$ to $A'x = I$. It takes $T_I(n)$ steps to calculate $(A')^{-1}$, and $T_E(n)$ is O(log^2 n) using O($n^{3.807}$/log^2 n) processors bound when n >128.
(b) From $(-1)^n c_n = \det[A]$ and theorem 1, we can obtain $T_D(n)$ in O(log^2 n) time using O($n^{3.807}$/log^2 n) processors bound when n >128.
(c) While $\phi(\lambda) = \det[\lambda I - A]$
$= (\lambda^n + c_1\lambda^{n-1} + c_2\lambda^{n-2} + ... + c_{n-1}\lambda + c_n)$, by theorem 1, we obtain $T_P(n)$ is O(log^2 n) time using O($n^{3.807}$/log^2 n) processors bound when n >128.

Theorem 3. The computing time of the parallel algorithm for A^n is $T_{PW}(n)$ is O(log^2 n) with O($n^{2.807}$/log n) processors bound when n >128.

Proof. We establish the theorem by exhibiting an algorithm.

Theorem 4. $T_I(n)$, $T_D(n)$, $T_E(n)$, $T_P(n)$, and $T_{PW}(n)$ are all of the same growth rate

Proof. From the step 1 through step 6 of the proposed parallel power matrix algorithm, we have $T_{PW}(n) \le 2T_I(n)$ + O(log n)+O(4). By the definition of equivalence theorems in Csanky (1976), we yield the result of the theorem.

REFERENCES

1. Csanky, L. (1976). Fast parallel matrix inversion algorithms. *SIAM Journal of Computing*, 5: 618-623.
2. Grayson, B. and Van De Geijn, R. (1996). A high performance parallel Strassen implementation. *Parallel Processing Letters*, 6(1): 3-12.
3. Kung, H.T. (1976). New algorithms and lower bounds for the parallel evaluation of certain rational expressions and recurrences. *Journal of the Association for Computing Machinery*, 23(2): 252-261.
4. Lakshmivarahan, S. and Dhall, S.K. (1990). *Analysis and Design of Parallel Algorithm*, McGraw-Hill, New York.
5. Padmini, M.V., Madan, B.B., and Jain, B.N. (1999). A linear array for large sparse matrix operations - Triangular system solvers and matrix multiplication, *Parallel Algorithms and Applications*, 13: 217-237.
6. Paprzycki, M. and Cyphers, C. (1996). Using Strassen's matrix multiplication in high performance solution of linear systems, *Computers and Mathematical Applications*, 31: 55-61.
7. Pudlak, P. (2000). A note on the use of determinant for proving lower bounds on the size of linear circuits, *Information Processing Letters*, 74: 197-201.
8. Rojo, O., Soto, R., and Rojo, H. (2000). Bounds for sums of eigenvalues and applications, *Computers and Mathematics with Applications*, 39: 1-15.
9. Romine, C.H. and Ortega, J.M. (1988). Parallel solution of triangular systems of equations, *Parallel Computing*, 6: 109-114.
10. Sameh, A.H. and Brent, R.P. (1977). Solving triangular systems on a parallel computer, *SIAM Journal on Numerical Analysis*, 14: 1101-1113.