A FAST MATRIX INVERSION ALGORITHM

by

L. Csanky

Memorandum No. ERL-M521

28 May 1975

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A FAST MATRIX INVERSION ALGORITHM[†]

L. Csanky

Computer Science Division
Department of Electrical Engineering and Computer Sciences
and the Electronics Research Laboratory
University of California, Berkeley, California 94720

May 1974

Abstract

In this paper, an investigation of the parallel arithmetic complexity of matrix inversion, solving systems of linear equations, computing determinants and computing the characteristic polynomial of a matrix is reported. The parallel arithmetic complexity of solving equations has been an open question for several years. The gap between the complexity of the best algorithms $(2n+0(1)$, where n is the number of unknowns/equations) and the only proved lower bound $(2 \log n$ (All logarithms in this paper are of base two.)) was huge. The first breakthrough came when Csanky [1] reported that the parallel arithmetic complexity of all these four problems has the same growth rate and exhibited an algorithm that computes these problems in $2n - 0(\log^2 n)$ steps. It will be shown in the sequel that the parallel arithmetic complexity of all these four problems is upper bounded by $0(\log^2 n)$ and the algorithms that establish this bound use a number of processors polynomial in n.

[†]This work was supported in part by National Science Foundation Grant DCR72-03725-A02.

1. INTRODUCTION

The Model and Complexity. The intuitive definition of the model of parallel computation and its complexity is the following. The model has an arbitrary number of identical processors and an arbitrarily large memory with unrestricted access. Each processor is capable of performing any one of the binary operations +, -, *, / in unit time. This unit time is called a step. (The processors do not necessarily perform the same operation in any step.) It is assumed that no other operation takes time (i.e. only the arithmetic operations count the overhead is ignored). The processors always take their operands from the memory and after a step they store the result in the memory. Before starting the computation the input data is stored in the memory. Then the computation starts. The parallel arithmetic complexity of the computation is the least number of steps necessary to produce the result in the memory.

If P is a computational problem of size n, then the parallel arithmetic computational complexity P(n) of P is the least number of steps necessary to compute P for any possible input.

Let A be an order n matrix. Let det(A), adj(A), tr(A) denote the determinant of A, the adjoint of A and the trace of A respectively. Unless specified otherwise capital letters denote matrices, lower case letters denote scalars.

Let I(n), E(n), D(n), P(n) denote the parallel arithmetic complexity of inverting order n matrices, solving a system of n linear equations in n unknowns, computing order n determinants and finding the characteristic polynomials of order n matrices respectively.

2. RESULTS

Theorem 1. $I(n) \leq O(\log^2 n)$ and the number of processors used in the algorithm is a polynomial in n.

Proof. Let A be the order n matrix to be inverted. Let the characteristic polynomial $f(\lambda)$ of A be

$$f(\lambda) = \det(\lambda I - A) = \lambda^n + c_1\lambda^{n-1} + \cdots + c_{n-1}\lambda + c_n .$$

Let

$$\operatorname{adj}(\lambda I - A) = I\lambda^{n-1} + B_2\lambda^{n-2} + \cdots + B_{n-1}\lambda + B_n .$$

Using the idea of one of the simplest proofs of the Cayley-Hamilton theorem (Marcus-Ming [2]) the following formulas were obtained:

$$B_1 = I \tag{1}$$

$$B_k = AB_{k-1} - \frac{I}{k-1}\operatorname{tr}(AB_{k-1}) \tag{2}$$

$$c_i = -\frac{1}{i}\operatorname{tr}(AB_i) \tag{3}$$

and

$$A^{-1} = n\frac{B_n}{\operatorname{tr}(AB_n)} . \tag{4}$$

(It was pointed out to the author that a number of people derived essentially the same formulas in recent years. Frame [3] seems to have been the first.)

Define operator T as follows:

$$TA = \operatorname{tr}(A)$$

$$(I+MT)A = A + MTA = A + M\operatorname{tr}(A) .$$

Then

$$B_k = (I - \frac{I}{k-1}T)AB_{k-1}$$

and

$$B_n = (I - \frac{I}{n-1}T)\{A[(I - \frac{I}{n-2}T)\{A[\cdots(I-IT)\{A[I]\}\cdots]\}]\} \quad . \qquad (5)$$

In this formula A can be thought of as a second operator, its action being multipication on the left.

Observe that

$$T(AT) = (TA)T \quad , \qquad (6)$$

that is, operators T and A associate. This implies that the factors in the expression for B_n associate, i.e.

$$B_n = (A - \frac{I}{n-1}TA)(A - \frac{I}{n-2}TA)\cdots(A - \frac{I}{2}TA)(A - ITA) \quad . \qquad (7)$$

Thus B_n can be computed in roughly $\log n$ stages by the well known binary tree method.

Stage 1. For all $i+1 = 2t$ compute from $(A - \frac{I}{i+1}TA)(A - \frac{I}{i}TA)$ the form

$$A^2 - M^{(1)}_{1\ 2t}TA^2 - M^{(2)}_{1\ 2t}TA \quad .$$

As a consequence of (6) matrices $M^{(h)}_{1\ 2t}$ can be computed independently. The powers of A can also be computed independently.

$\vdots$

Stage j. For all $2^j t$ compute from

$$(A^{2^{j-1}} - M^{(1)}_{j-1\ 2^j t}TA^{2^{j-1}} - \cdots - M^{(2^{j-1})}_{j-1\ 2^j t}TA)$$
$$\cdot (A^{2^{j-1}} - M^{(1)}_{j-1\ 2^j(t-1)}TA^{2^{j-1}} - \cdots - M^{(2^{j-1})}_{j-1\ 2^j(t-1)}TA)$$

the form

$$A^{2^j} - M^{(1)}_{j\,2^j t}\, TA^{2^j} - M^{(2)}_{j\,2^j t}\, TA^{2^j-1} - \cdots - M^{(2^j)}_{j\,2^j t}\, TA \ .$$

As a consequence of (6) matrices $M^{(h)}_{j\,2^j t}$ can be computed independently. The powers of A can also be computed independently.

The number of steps stage j takes is roughly

$$\log n + \log n + 1 + \log 2^j = \log n + \log n + 1 + j$$

where the first term is the number of steps the multiplication on the right by $A^{2^{j-1}}$ and matrices $M^{(h)}$ takes, the second term is the number of steps computing the traces takes, multiplication by the trace takes 1 step and addition of the matrices takes j steps. Thus the total number of steps for all stages is roughly

$$\log n(2 \log n+1) + 1 + 2 + 3 + \cdots + \log n = 2.5 \log^2 n + 1.5 \log n \ .$$

Since $n \frac{B_n}{TAB_n}$ ($TAB_n \neq 0 \Leftrightarrow A$ is invertible) can be computed in $0(\log n)$ additional steps, we have

$$I(n) \leq 2.5 \log^2 n + 0(\log n) \ .$$

A very crude upper bound on the number of processors can be obtained by observing that in any stage the multiplication phase takes the maximum number of processors.

In stage j the number of matrix multiplications is roughly

$$2^{j-1} + \frac{n}{2^j} 2^{2j-2} \ .$$

Thus for the number of processors p we obtain

$$p \leq \max_{(j)}\{n^3 2^{j-1} + \frac{n^4}{2^j} 2^{2j-2}\} \approx \frac{1}{4}n^5 + \frac{1}{2}n^3 \ . \qquad \text{Q.E.D.}$$

Careful counting of the processors and application of Leverrier's method yields $O(\log^2 n)$ steps with $O(n^4)$ processors.

Corollary 2. $E(n), D(n), P(n) \leq O(\log^2 n)$ and the number of processors used in the algorithms is a polynomial in n.

Proof. Solve $Ax = b$ where b is a column vector by inverting A, then $x = A^{-1}b$. Compute $\det(A)$ from $\det(A) = -c_n = \frac{TAB_n}{n}$. Compute $f(\lambda)$ from $c_i = -\frac{TAB_i}{i}$. Q.E.D.

3. CONCLUSIONS

This work has decreased the gap between the lower and upper bounds on the parallel arithmetic complexity of problems I, E, D, P. Indeed we have

$$O(\log n) \leq I(n), E(n), D(n), P(n) \leq O(\log^2 n) \ .$$

It is our belief that if $I(n) = O(\log n)$ then the algorithm which establishes this will have to use some new, surprising and fundamental result.

ACKNOWLEDGMENT

I wish to thank Professor Manuel Blum for his encouragement and interest in this work.

REFERENCES

[1] Csanky, L., "On the parallel complexity of some computational problems," Ph.D. dissertation, Computer Science Division, University of California, Berkeley, September 1974.

[2] Marcus, M. and Ming, H., A Survey of Matrix Theory and Matrix Inequalities, Allyn and Bacon, Inc., 1964.

[3] Frame, J.S., "A simple recurrent formula for inverting a matrix," Bull. Amer. Math. Soc. 55 (1949), p. 1045.