JOURNAL OF ALGORITHMS **21,** 71–79 (1996)
ARTICLE NO. 0037

Very Fast Approximation of the Matrix Chain Product Problem*

Artur Czumaj†

Institute of Informatics, University of Warsaw, Banacha 2, 02-097 Warsaw, Poland

Received December 14, 1992

This paper considers the matrix chain product problem. This problem can be solved in $O(n \log n)$ sequential time, while the best known parallel NC algorithm runs in $O(\log^2 n)$ time using $n^6/\log^6 n$ processors and in $O(\log^3 n)$ time with $O(n^2)$ time–processor product. This paper presents a very fast parallel algorithm for approximately solving the matrix chain product problem and for the problem for finding a near-optimal triangulation of a convex polygon. It runs in $O(\log n)$ time on a CREW PRAM and in $O(\log \log n)$ time on a COMMON CRCW PRAM. If the dimensions of matrices are integers drawn from a domain $[k, \ldots, k + s]$, we can speed up our algorithm to run in $O(\log \log \log(n + s))$ time on a COMMON CRCW PRAM. In all cases the total time–processor product is linear. The algorithm produces solutions for the above problems that are at most $1/(2\sqrt{3} + 3)$ (≈ 0.1547) times the optimal solutions.

1. INTRODUCTION

Consider the evaluation of the product of n matrices $M = M_1 \times M_2 \times \cdots \times M_n$, where M_i is a $d_{i-1} \times d_i$ $(d_i \geq 1)$ matrix. Since matrix multiplication satisfies the associative law, the final result is the same for all orders of multiplying. However, the order of multiplication greatly affects the total number of operations used to evaluate M. The *matrix chain product* problem consists of finding an optimal order of multiplying the matrices such that the total number of operations is minimized. In this paper, any order of multiplying n matrices will be described by a method of putting $n - 1$ pairs of nested parentheses that defines the order of

* This work was supported by Grant KBN 2-1190-91-01.

† Current address: Heinz Nixdorf Institut, Universität-GH-Paderborn, D-33095 Paderborn, Germany. E-mail: artur@uni-paderborn.de.

0196-6774/96 $18.00

multiplication. We also assume that the number of operations used to multiply a $p \times q$ matrix by a $q \times r$ matrix is pqr.

In order to solve this problem, Hu and Shing [10] derived the problem of finding an *optimal triangulation of a convex polygon*. Given a convex polygon $P = (v_0, v_1, \ldots, v_n)$ with positive weights associated with each vertex, divide it into triangles so that the total cost of partitioning is the smallest possible. The cost of a triangulation is the sum of the costs of all triangles in this partition. The cost of a triangle is the product of the weights at each vertex of the triangle.

Hu and Shing [10] showed that the matrix chain product problem and the problem of finding an optimal triangulation of a convex polygon are equivalent under linear time reduction. We show that the transformation can be performed in constant time using n processors on an EREW PRAM.

An optimal algorithm for both these problems was given by Hu and Shing [10] and it runs in $O(n \log n)$ serial time. This and all other known sequential algorithms are hardly parallelizable, and the best previous known approach to design parallel algorithms for these problems is based on dynamic programming. It gives $\mathcal{NC}$ algorithms which run in $O(\log^2 n)$ time using $n^6/\log^k n$ processors on a CREW PRAM for some constant k [8, 9, 12]. The large number of processors used in these parallel algorithms makes them impractical for real applications. This suggests designing algorithms that find approximate solutions and have better performance.

Chandra [5] showed that an arbitrary order of matrix multiplications may be as bad as $\Omega(T_{\text{opt}}^3)$, where T_{opt} is the minimum number of operations required to compute the matrix chain product. In 1978 Chin [6] described a sequential algorithm for finding a near-optimal order of matrix multiplications. This algorithm was later improved, analyzed, and transformed to the problem of finding a near-optimal triangulation of a convex polygon by Hu and Shing [11]. Their algorithm runs in linear time and has an error ratio of at most 15.47% (i.e., it produces an order of matrices which requires at most $1.1547 \times T_{\text{opt}}$ operations to compute the matrix chain product).

In this paper, we describe a parallel $\mathcal{NC}$ algorithm that solves the two problems approximately. It runs in $O(\log n)$ time using $n/\log n$ processors on a CREW PRAM and in $O(\log \log n)$ time using $n/\log \log n$ processors on a COMMON CRCW PRAM. In designing parallel algorithms, an important goal is to minimize the total work done by the algorithm, where the work of a parallel algorithm is defined to be the product of the number of processors used and the time taken. Our algorithm uses linear work, which is the best possible. Thus it is work optimal. Furthermore, we can improve its running time if the domain of the input is restricted. The values of the matrix dimensions in the matrix chain product problem are

always integer. We shall show that if they are drawn from a domain $[k, \ldots, k + s]$ for some k and s, we can implement the algorithm to run in $O(\log \log \log(n + s))$ time with optimal linear work on a COMMON CRCW PRAM. Our algorithm produces the order of matrix multiplications and the triangulation of a convex polygon with an error ratio of at most 15.47% —the same as that in the Chin–Hu–Shing algorithm. In most cases the solution found is actually within a few percent of the best possible solution. Furthermore, the error ratio decreases as n grows.

Very recently Bradford [4] independently reported similar results for both above problems. Also, Czumaj [7] gave an $O(\log^3 n)$-time algorithm using $n^2/\log^3 n$ processors on a CREW PRAM for the matrix chain product problem.

Throughout this paper we will use $v_0, v_1, \ldots, v_n$ to denote vertices as well as their weights in a convex polygon. For simplicity we assume that all weights are distinct. If there are some vertices with the same weight then we assume that a particular ordering is chosen and remains fixed. The distinct weights of the vertices induce a total order on the vertices. Hence if $v_i < v_j$ we will say v_i is *smaller* than v_j.

1.1. The All Nearest Smaller Values Problem

The parallel algorithm we shall describe makes use of the following problem.

The All Nearest Smaller Values Problem (*ANSV*). Given an array $A = (a_0, a_1, \ldots, a_n)$, for each a_i $(0 \le i \le n)$, find the nearest element to its left (and the nearest element to its right) that is less than a_i, if such an element exists. That is, for each i, $1 \le i \le n$, find the maximal j $(0 \le j < i)$ and the minimal k $(i < k \le n)$ such that $a_j < a_i$ and $a_k < a_i$.

Fact 1.1 [1, 2]. The ANSV problem can be solved in $O(\log n)$ time on a CREW PRAM and in $O(\log \log n)$ time on a COMMON CRCW PRAM. If the entries in array A are integers drawn from $[k, \ldots, k + s]$ for some k and s, it can be solved in $O(\log \log \log(n + s))$ time. In all cases, the total work is linear.

2. REDUCTION TO THE TRIANGULATION OF BASIC POLYGONS

Define a *basic polygon* to be a polygon such that the second and the third smallest vertices are neighbors of the smallest one. The following fact allows us to reduce the triangulation problem to the triangulation of basic polygons.

Fact 2.1 [10]. There exists an optimal triangulation of a convex polygon containing arcs or sides connecting the smallest vertex to both the second and the third smallest vertices.

Hence in finding an optimal triangulation of a polygon, we can divide it into subpolygons by joining the smallest vertex with the second smallest and the third smallest vertices repeatedly, until each subpolygon has the smallest vertex adjacent to the second smallest and the third smallest vertices by sides. This partition divides the polygon into basic subpolygons. It can be found in the following way.

Divide the polygon into two parts by the arc joining the smallest vertex, say v_0, with the second smallest one, say v_k. Consider two obtained subpolygons $P_1 = (v_0, v_1, \ldots, v_k)$ and $P_2 = (v_k, v_{k+1}, \ldots, v_n, v_0)$ independently. If v_t is the third smallest vertex in P_1, then P_1 is divided into polygons $P_{1,1} = (v_0, \ldots, v_t)$ and $P_{1,2} = (v_0, v_t, v_{t+1}, \ldots, v_k)$, $P_{1,2}$ is already a basic polygon. $P_{1,1}$ and P_2 will be divided further. Consider $P_{1,1}$. A vertex v_t can be the third smallest one if and only if there is no smaller vertex between v_0 and v_t. To find all such vertices v_t, we solve the ANSV problem with respect to the weights $(v_0, v_1, \ldots, v_k)$. Then we join v_t and v_0 if v_0 is the nearest smaller vertex to the left of v_t. Partition of polygon P_2 can be done similarly. Hence Fact 1.1 implies the following.

Fact 2.2. Partition of the polygon into nonintersecting basic subpolygons can be done in $O(\log n)$ time on a CREW PRAM and in $O(\log\log n)$ time on a COMMON CRCW PRAM. If all weights of the vertices in the polygon are integers drawn from $[k, \ldots, k + s]$ for some k and s, it can be done in $O(\log\log\log(n + s))$ time. In all cases, the total work is linear.

3. CHIN–HU–SHING APPROXIMATING ALGORITHM

In this section, we briefly describe the approximate sequential algorithm due to Chin [6] and Hu and Shing [10].

Let v_m be the smallest vertex of a convex polygon and let v_t be a vertex with v_k and v_c as its two neighbors. Define vertex v_t to be *large* if

$$\frac{1}{v_k} + \frac{1}{v_c} > \frac{1}{v_t} + \frac{1}{v_m}.$$

The Cin–Hu–Shing algorithm runs in the following two steps:

1. While there is a large vertex v_t, cut off v_t by the arc connecting its two neighbors and remove v_t from the polygon (see Fig. 1a); the arc connecting the neighbors of v_t is called an *h-arc*.

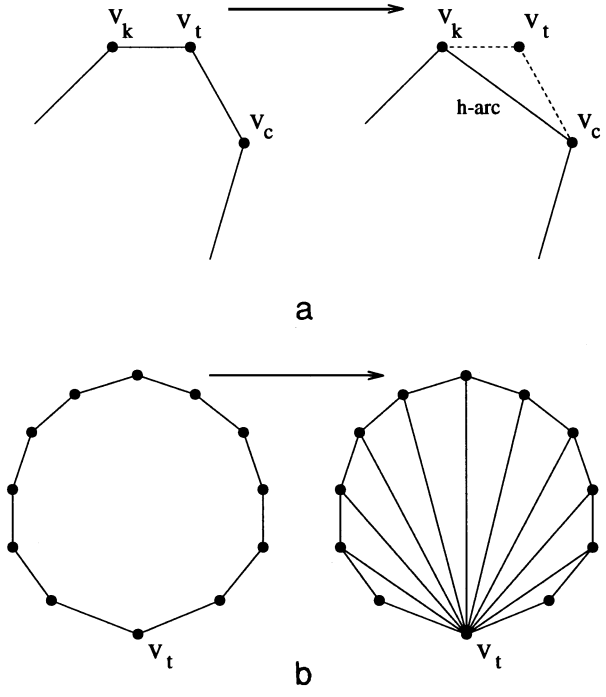

FIG. 1. Two steps of the algorithm CHS.

2. If none of vertices is large, then the smallest vertex is connected to all other vertices (see Fig. 1b); obtained arcs are called *fan-arcs*.

In the sequel, this algorithm will be called *algorithm CHS*.

In most cases the algorithm CHS yields the optimal solution or a solution which is only a few percent worse than the optimal one (less than 1% on the average; see simulation results in [6]). For a given $n \geq 5$, Hu and Shing [11] calculated the worst case error ratio, which is $1/(2\sqrt{n-2}+3)$ and is maximum when $n = 5$.

Fact 3.1 [6, 10, 11]. The algorithm CHS can be implemented to run in $O(n)$ time. It finds a near-optimal triangulation of a polygon with the maximum error ratio $1/(2\sqrt{3}+3) \approx 0.1547$.

4. APPROXIMATE PARALLEL ALGORITHM

Our parallel algorithm produces the same triangulation as that of Chin–Hu–Shing but using a different approach. The input is a basic

polygon $(v_0, v_1, \ldots, v_n)$, where v_0, v_1, and v_n are the smallest, second smallest, and third smallest vertices.

Define a *candidate* to be an arc (v_i, v_j) that is not a side (i.e., $0 \le i < j - 1 < n$) and for each r, $i < r < j$, the inequality $v_r > \max\{v_i, v_j\}$ holds. We will store candidate (v_i, v_j) with the smallest vertex v_r such that $i < r < j$. Since $v_p > v_r > \max\{v_i, v_j\}$ for each p ($i < p < r$ or $r < p < j$), vertices v_i and v_j are the nearest smaller values to the left and to the right of v_r. Hence each vertex stores at most one candidate. Since $v_r > v_n > v_1$ for each r, $1 < r < n$, vertex v_r stores a candidate which we denote by (v_{i_r}, v_{j_r}). Vertices v_0, v_1, and v_n store no candidates. Hence there are $n - 2$ candidates in the basic polygon with $n + 1$ vertices and we can find all of them using the algorithm for the ANSV problem.

Suppose two candidates intersect. Let (v_i, v_j) and (v_p, v_q) be intersecting candidates. Without loss of generality, we may assume $i < p < j < q$. Then $v_p > \max\{v_i, v_j\} \ge v_j > \max\{v_p, v_q\}$, i.e., $v_p > v_p$, a contradiction. Thus, no candidates intersect (except possibly at the endpoints) and they triangulate the basic polygon.

Let vertices v_k and v_c be neighbors of a vertex v_t in certain stage of the algorithm CHS. It follows directly from the definition that if v_t is large then $v_t > \max\{v_k, v_c\}$. Since v_k and v_c are the neighbors of v_t, each vertex v_r ($k < r < t$ or $t < r < c$) has had to be cut out in a former step of the algorithm CHS. Hence v_r was large with respect to its neighbors, which implies $v_r > v_t > \max\{v_k, v_c\}$. This gives us the following lemma.

LEMMA 4.1. *If a vertex v_t is cut off by an h-arc (v_k, v_c), then (v_k, v_c) is the candidate stored with v_t.*

From the discussion above a candidate (v_k, v_c) is an h-arc if and only if for each r, $k < r < c$, vertex v_r is large with respect to v_{i_r} and v_{j_r}.

Let $C = (c_1, \ldots, c_n)$ be an array of bits such that $c_1 = c_n = 0$, and for $2 \le p \le n - 1$, $c_p = 1$ if vertex v_p is large with respect to v_{i_p} and v_{j_p}. Hence a candidate (v_{i_r}, v_{j_r}) is an h-arc if and only if $c_q = 1$ for each q, $i_r < q < j_r$. To verify this condition we solve the ANSV problem for array C. Let for $2 \le r \le n - 1$, L_r and R_r be the nearest smaller values to the left and to the right of c_r. Then $c_{L_r} = c_{R_r} = 0$ and $c_p = 1$ for each p, $L_r < p < r$ or $r < p < R_r$. Thus we can find all h-arcs in the same time and work that is needed to solve the ANSV problem. Candidate (v_{i_r}, v_{j_r}) is an h-arc only if $c_r = 1$, $L_r \le i_r$ and $j_r \le R_r$.

The remaining problem is to find all fan-arcs of the algorithm CHS. There is a fan-arc (v_0, v_r) if and only if vertex v_r is not cut off by the h-arc (v_{i_r}, v_{j_r}). Hence using the information on the h-arcs, we can easily verify this condition in constant time with n processors.

We summarize the algorithm with a specification of the output. The output is an array $\text{ARCS} = \text{ARC}_1, \ldots, \text{ARC}_{n-2}$) of the arcs in the triangula-

tion. For each r, $2 \le r \le n-1$, if (v_{i_r}, v_{j_r}) is an h-arc then $\text{ARC}_{r-1} = (v_{i_r}, v_{j_r})$; otherwise $\text{ARC}_{r-1} = (v_0, v_r)$. This completes the proof of the following theorem.

THEOREM 4.2. *The problem of finding a near-optimal triangulation of a convex polygon can be solved in $O(\log n)$ time on a CREW PRAM and in $O(\log \log n)$ time on a COMMON CRCW PRAM. If the weights in the polygon are integers drawn from $[k, \ldots, k+s]$ for some k and s, then it can be solved in $O(\log \log \log(n+s))$ time on a COMMON CRCW PRAM. In all cases the total work is linear.*

5. APPROXIMATING THE MATRIX CHAIN PRODUCT PROBLEM

In this section we show that the approximation algorithm presented in the previous section can be applied to approximate a solution of the matrix chain product problem.

First we formally specify the input and the output of the matrix chain product problem. The input is an integer-valued sequence $(d_0, d_1, \ldots, d_n)$ of matrix dimensions. An order of multiplication corresponds to a method of putting $n-1$ nested pairs of parentheses around n matrices. We define the output to be the array $\text{BRACKETS} = (B_1, \ldots, B_{n-1})$ which contains the positions of all the pairs of parentheses that appear in an optimal order of multiplying the matrices. Here for each i, $1 \le i \le n-1$, $B_i = (j, t)$ only if the expression $(M_{j+1} \times M_{j+2} \times \cdots \times M_t)$ is present in this optimal order; that is, there is a pair of parentheses around M_{j+1} and M_t.

Hu and Shing [10] showed the following correspondence between the matrix chain product problem and the problem of finding an optimal triangulation of a convex polygon. Let $M = M_1 \times \cdots \times M_n$, where M_i is a $d_{i-1} \times d_i$ matrix, and let $P = (v_0, v_1, \ldots, v_n)$ be a convex polygon such that the weight assigned to vertex v_i is d_i. Given an order of multiplying matrices $M_1, \ldots, m_n$ defined by the array $\text{BRACKETS} = (B_1, \ldots, B_{n-1})$ with $B_{n-1} = (0, n)$, there is the corresponding triangulation of P defined by the array $\text{ARCS} = (\text{ARC}_1, \ldots, \text{ARC}_{n-2})$ such that for $1 \le i \le n-2$, $0 \le j < t \le n$, $\text{ARC}_i = (v_j, v_t)$ if and only if $B_i = (j, t)$. This transformation can be obtained in constant time with n processors on an EREW PRAM. Hu and Shing [10] showed that the costs of corresponding instances are the same. Since the transformation above is one-to-one, we get the following theorem.

THEOREM 5.1. *The problem of finding a near-optimal order of matrix chain product can be solved in $O(\log n)$ time on a CREW PRAM and in $O(\log \log n)$ time on a COMMON CRCW PRAM. If the dimensions of the*

matrices are integers drawn from $[k, \ldots, k+s]$ *for some* k *and* s*, then it can be solved in* $O(\log\log\log(n+s))$ *time on a COMMON CRCW PRAM. In all cases the total work is linear.*

6. CONCLUSIONS

We have given the very fast optimal parallel algorithm for finding a near-optimal order of matrix chain product and a near-optimal triangulation of a convex polygon. It runs in $O(\log n)$ time on a CREW PRAM and in $O(\log\log n)$ time on a COMMON CRCW PRAM. The time–processor product of the algorithm is linear. One can improve these bounds if the domain of the input values is restricted. The values of the matrix dimensions in the matrix chain products problem are always integer. Therefore if dimensions are drawn from a domain $[k, \ldots, k+s]$ for some k and s, one can find a near-optimal order of matrix multiplications in $O(\log\log\log(n+s))$ time with linear work on a COMMON CRCW PRAM. However, one would require a stronger condition. We can speed-up our algorithm to run in $O(\alpha(n))$ time on a COMMON CRCW PRAM if the difference between two successive elements (i.e., dimensions or weights of vertices in a polygon) is bounded by a constant. This follows from the result for the ANSV problem given in [3].

Note that the arcs of the near-optimal triangulation of basic polygons from Section 4 are given in inorder numbering. That is, if $\text{ARC}_i = (v_x, v_y)$, $\text{ARC}_j = (v_y, v_z)$, and $\text{ARC}_k = (v_x, v_z)$ for $x < y < z$, then we have $i < k < j$. Since to obtain the triangulation of a convex polygon we divide it into basic polygons with possible shifting or reversing the number of vertices, one can extend (in constant time with n processors) the inorder numbering from basic polygons to the convex polygon. Using the transformation presented in Section 5, we can obtain also a near-optimal order of matrix multiplication whose parenthesization is given in the inorder form. That is, if there is a parenthesization of the form $((M_i \times \cdots \times M_k) \times (M_{k+1} \times \cdots \times M_j))$, then the inorder number of a pair of parentheses around M_i and M_j is larger than the one around M_i and M_k, and is smaller than the one around M_{k+1} and M_j. This representation seems to be more readable and practical than the one given in Section 5.

As in [5] the algorithm can be generalized to a larger class of functions by assuming that the multiplication of a $p \times q$ matrix and a $q \times r$ matrix takes $\tau(p, q, r)$ operations. Most of the results (except the error ratio) will stay true as long as $\tau(p, q, r)$ is nonnegative and *reasonable*; i.e., $\tau(p, q, r)$ is monotonically nondecreasing in p, q, r and $\tau(p, q, r) = \tau(r, p, q)$.

ACKNOWLEDGMENTS

The author thanks B. Chlebus, K. Diks, and W. Rytter for carefully reading a draft of the paper and for many helpful comments. The author is also very much indebted to the anonymous referees for making suggestions that improved the paper.

REFERENCES

1. O. Berkman, D. Breslauer, Z. Galil, B. Schieber, and U. Vishkin, Highly parallelizable problems, *in* "Proceedings of the 21st Annual ACM Symposium on Theory of Computing, 1989," pp. 309–319.
2. O. Berkman and Y. Matias, personal communication, 1992.
3. O. Berkman and U. Vishkin, "Almost Fully-Parallel Parentheses Matching," UMIACS-TR-91-103, Institute for Advanced Computer Studies, Univ. of Maryland, 1991.
4. P. G. Bradford, Efficient parallel dynamic programming, manuscript, July 1, 1992.
5. A. K. Chandra, "Computing Matrix Chain Products in Near-Optimal Time," IBM Research Report RC 5625(#24393), IBM T. J. Watson Res. Ctr., Yorktown Heights, NY, 1975.
6. F. Y. Chin, An $O(n)$ algorithm for determining a near-optimal computation order of matrix chain products, *Comm. ACM* **21** (1978), 544–549.
7. A. Czumaj, Parallel algorithm for the matrix chain product and the optimal triangulation problem, manuscript, May 1992.
8. Z. Galil and K. Park, Parallel algorithms for dynamic programming recurrences with more than 0(1) dependency, *J. Parallel Distrib. Comput.* **21** (1994), 213–222.
9. S-H. S. Huang, H. Liu, and V. Viswanathan, Parallel dynamic programming, *in* "Proceedings of the 2nd IEEE Symposium on Parallel and Distributed Processing, 1990," pp. 497–500.
10. T. C. Hu and M. T. Shing, Some theorems about matrix multiplications, *in* "Proceedings of the 21st Annual IEEE Symposium on the Foundations of Computer Science, 1980," pp. 28–35.
11 T. C. Hu and M. T. Shing, An $O(n)$ algorithm to find a near-optimum partition of a convex polygon, *J. Algorithms* **2** (1981), 122–138.
12 W. Rytter, On efficient parallel computations for some dynamic programming problems, *Theoret. Comput. Sci.* **59**, (1988), 297–307.