Research Report 224

An optimal parallel algorithm for computing a near-optimal order of matrix multiplications

Czumaj A

RR224

This paper considers the computation of matrix chain products of the form $M_1 \times M_2 \times \ldots M_{n-1}$. The order in which the matrices are multiplied affects the number of operations. The best sequential algorithm for computing an optimal order of matrix multiplication runs in $O(n \log n)$ time while the best known parallel NC algorithm runs in $O(\log^2 n)$ time using $n^6/\log^6 n$ processors. This paper presents the first approximating optimal parallel algorithm for this problem and for the problem of finding a near-optimal triangulation of a convex polygon. The algorithm runs in $O(\log n)$ time using $n/\log n$ processors on a CREW PRAM, and $O(\log \log n)$ time using $n/\log \log n$ processors on a weak CRCW PRAM. It produces an order of matrix multiplications and a partition of polygon which differ from the optimal ones at most 0.1547 times.

*This work was supported by the grant KBN 2-11-90-91-01.

Department of Computer Science
University of Warwick
Coventry CV4 7AL
United Kingdom

February 1992

An optimal parallel algorithm for computing a near-optimal order of matrix multiplications*

Artur Czumaj

Warsaw University
and
University of Warwick

February, 1992

Abstract

This paper considers the computation of matrix chain products of the form $M_1 \times M_2 \times \cdots \times M_{n-1}$. The order in which the matrices are multiplied affects the number of operations. The best sequential algorithm for computing an optimal order of matrix multiplication runs in $O(n \log n)$ time while the best known parallel NC algorithm runs in $O(\log^2 n)$ time using $n^6/\log^6 n$ processors. This paper presents the first approximating optimal parallel algorithm for this problem and for the problem of finding a near-optimal triangulation of a convex polygon. The algorithm 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 weak CRCW PRAM. It produces an order of matrix multiplications and a partition of polygon which differ from the optimal ones at most 0.1547 times.

1 Introduction

The problem of computing an *optimal order of matrix multiplication* (*the matrix chain product problem*) is defined as follows. Consider the evaluation of the product of $n-1$ matrices

$$M = M_1 \times M_2 \times \cdots \times M_{n-1}$$

where M_i is a $w_{i-1} \times w_i$ ($w_i \geq 1$) 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 to evaluate M. The problem is to find an optimal order of multiplying the matrices, such that the total number of operations is minimized. Here, we assume that the number of operations to multiply a $p \times q$ matrix by a $q \times r$ matrix is pqr. It was shown in [Cha-75] that an

*This work was supported by the grant KBN 2-11-90-91-01.

arbitrary order of matrix multiplications may be as bad as $O(T_{opt}^3)$, where T_{opt} is the minimal number of operations required to compute matrix chain products.

One can show that this problem is equivalent to the problem of finding an *optimal triangulation of a convex polygon* (see [HS-80]). Given a convex polygon $(v_0, v_1, \ldots, v_n)$. Divide it into triangles, such that the total cost of partitioning is the smallest possible. By the total cost of triangulation we mean the sum of costs of all triangles in this partitioning. The cost of a triangle is the product of weights in each vertex of triangle.

Both these problems can be solved sequentially in $O(n \log n)$ time [HS-80]. The best known approach to design parallel algorithms is based on dynamic programming. It gives us NC algorithms which run in $O(\log^2 n)$ time using $O(n^6/\log^k n)$ processors on a CREW PRAM for some k ([Ryt-88], [HLV-90] and [GP-92]). Algorithms which increase the total time-processor product from $O(n \log n)$ to $O(n^6/\log^k n)$ are not of much of practical value. This suggests designing approximating algorithms for these problems. They would run fast and use few processors what would fill the gap between the total work in their serial versions and parallel ones.

[Chin-78] described a sequential approximating algorithm for finding a near optimal order of matrix multiplications. This algorithm was later improved, analysed and transformed to the problem of finding a near-optimal triangulation of a convex n-gon in [HS-81]. The algorithm runs in linear time and has an error at most 15.47% (i.e., its cost is at most 1.1547 × the cost of an optimal order of product of matrices). Especially interesting is the fact that this error is decreasing when n is growing.

In this paper we describe the first optimal parallel algorithm which solves these two problems approximatelly. 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 CRCW PRAM. Moreover we can improve these running times if the domain of input is restricted. In the matrix chain product problem there are always integer values of matrix dimensions. We will show that if dimensions are drawn from a domain $[k \ldots k+s]$ we can implement our algorithm to run in $O(\alpha(n) + \log \log \log s)$ time with linear time-processor product on a priority CRCW PRAM[1]. Our algorithm produces the order of matrix multiplications and the triangulation of a convex polygon with an error ratio at most 0.1547 - the same as in Chin-Hu-Shing algorithm.

The models of parallel computation that are used in this paper are the concurrent-read exclusive-write (CREW) and the concurrent-read concurrent-write (CRCW) parallel random access machines (PRAM). A PRAM employs synchronous processors all having access to a common memory. A CREW PRAM allows several processors to read the same entry of memory simultaneously, but forbids multiple concurrent writes to a cell. A CRCW PRAM allows simultaneous access by more than one processor to the same memory cell for both reads and writes. In this paper we mainly focus on the weakest model of a CRCW PRAM - *weak CRCW*, in which the only concurrent writes allowed are of the value 1.

This paper is organized as follows. Section 2 contains some basic definitions and

[1] $\alpha(n)$ denotes very slowly growing the inverse-Ackerman function, for definition see e.g. [BV-89]

notions concerning the algorithm. Section 3 recalls Chin-Hu-Shing sequential algorithm. Section 4 shows reduction of the initial problem to the one of finding an optimal triangulation in a basic polygon. Section 5 describes the new algorithm which is then implemented on a CREW PRAM and on a CRCW PRAM in section 6. In the last section we give some extensions of our algorithm.

2 Basic notions and definitions

Firstly we introduce some basic notions and facts.

The fact stated below converts the matrix chain product problem to the problem of finding an optimal triangulation of a convex polygon, see [HS-80].

Fact 2.1 *Any order of multiplying $n-1$ matrices corresponds to a partition of a n-sided convex polygon.*

Corollary 2.2 *The problem of finding an optimal order of multiplying a chain of matrices is equivalent to the problem of finding an optimal triangulation of a convex polygon. Here by the cost of partitioning of a polygon we mean the sum of costs of all triangles in a given partition, and by the cost of a triangle we mean the product of values of three vertices in this triangle.*

This equivalence and transformation are clear, so we give the following observation without a proof.

Observation 2.3 *Any partition of a convex polygon can be transformed to an order of multiplication of chain of matrices in constant time with n processors on a CREW PRAM. Also any order of multiplication of chain of matrices can be transformed to a partition of a convex polygon in constant time with n processors on a CREW PRAM.*

From now on, only the partitioning problem will be discussed.

Throughout this paper we will use $w_0, w_1, \ldots, w_{n-1}$ and $v_0, v_1, \ldots, v_{n-1}$ 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 weights then we assume that a particular ordering is chosen and remains fixed. We can choose e.g. lexicographically ordering[2].

Define a vertex v_i to be the *smallest* (minimum) one if for each other vertex v_j we have $v_i < v_j$. Similarly we define the kth *smallest* vertex v_i if there are exactly $k-1$ vertices smaller than v_i.

We will need the following problems.

The all nearest smaller values problem
Given an array $A = (a_0, a_1, \cdots, a_n)$. For each a_i $(0 \leq i \leq n)$, 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, $1 \leq i \leq n$, find the maximal j $(0 \leq j < i)$ and the minimal k $(i < k \leq n)$ such that $a_j < a_i$ and $a_k < a_i$.

[2]That is, $v_i \prec v_j$ iff $v_i < v_j$, or $v_i = v_j$ and $i < j$

Fact 2.4 **([BSV-88], [BBGSV-89])** *The all nearest smaller values problem can be solved 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 weak CRCW PRAM.*

The prefix minima problem

Given an array $A = (a_0, a_1, \cdots, a_n)$. For each a_i $(0 \le i \le n)$, find the minimum among $a_0, \ldots, a_i$.

Fact 2.5 **([Sch-87], [BSV-88])** *The prefix minima problem can be solved in $O(\log n)$ time with $n/\log n$ processors on a CREW PRAM and in $O(\log\log n)$ time with $n/\log\log n$ processors on a weak CRCW PRAM.*

The all nearest zero problem

Let $A = (a_0, a_1, \ldots, a_n)$ be an array of bits. Find for each bit a_i the nearest zero bit both to its left and right.

Fact 2.6 *The all nearest zero problem can be solved in $O(\log n)$ time using $n/\log n$ processors on a CREW PRAM and in $O(\alpha(n))$ time using $n/\alpha(n)$ processors on a weak CRCW PRAM ([BV-89]).*

Evaluation of logical 'AND' in a tree

Given a binary tree T (with possible chains) of n nodes. Let each vertex has assigned a logical value *false* or *true*. Evaluate for each internal vertex v logical 'AND' of v and all its descendants.

Remark. This problem can be transformed to the problem of evaluation of expression with logical operation 'AND' in each internal vertex.

Lemma 1

Suppose we are given an infix order in a tree T. That is, for each node v_i we know its infix rank i. Assume also that for each node there is given the number of all its descendants in a tree. Then we can evaluate logical 'AND' for all vertices in a tree T in $O(\log n)$ time using $n/\log n$ processors on a CREW PRAM and in $O(\alpha(n))$ time using $n/\alpha(n)$ processors on a weak CRCW PRAM.

Proof: We show how to reduce this problem to the all nearest zero problem. Let $A = (a_1, \ldots, a_n)$ be an array of bits such that $a_i = 1$ iff the ith (w.r.t. to an infix order) vertex v_i in a tree T has assigned value *true*. Because of infix order we know that for given vertex v_i (with ith number) all its descendants are in the consecutive subarrays to its left (*descendants of its left son*) and right (*descendants of its right son*). Therefore we start with finding for each a_i (that is, for each vertex v_i) the nearest zero bit to its left and rigth. Then we have to check if these zeros are in a subtree rooted at v_i. Because we know the number of its descendants we can check this in constant time with one processor for each node v. □

Remark. These results can be extended to the case when we are given a prefix or postfix order and the number of all descendants or if we are given in the array the Euler tour (see e.g. [GR-88]) of the input tree.

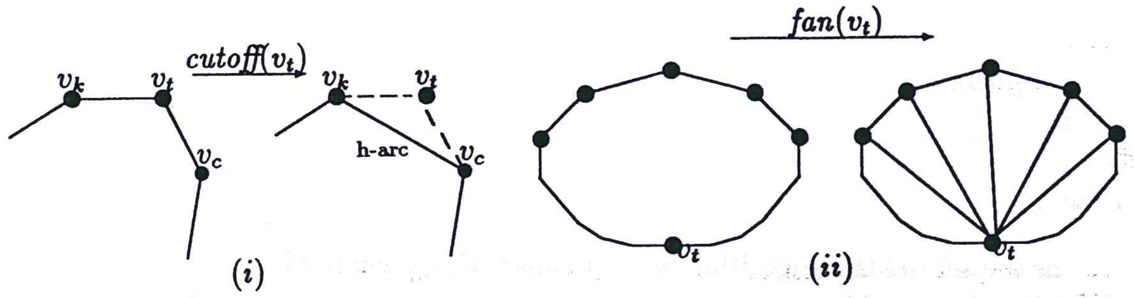

Figure 1: Procedures (*i*) *cutoff*(v_t) and (*ii*) *fan*(v_t).

3 Chin-Hu-Shing approximating algorithm

The Chin-Hu-Shing ([Chin-78] and [HS-81]) algorithm is based on two intuitions:

- If a vertex has a very large weight, then it should be cut in the optimal partition.
- If none of vertices has a very large weight, then we should join the smallest vertex with all other vertices.

Let v_m be the smallest vertex of a convex polygon, and v_t be a vertex with v_k and v_c as its two neighbours. Define vertex v_t to be *large* iff

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

In Chin-Hu-Shing algorithm the following procedures are used.

Procedure *cutoff* (v_t)::
Let v_k and v_c be two neighbours of vertex v_t. Join v_k with v_c (i.e., cut off v_t) and call this arc *h-arc*. Later we consider a polygon without vertex v_t.

Procedure *fan* (v_t)::
Join v_t with all vertices in a polygon. These arcs are called *fan-arcs*.

Additionally there are three procedures. They pop a given vertex off the stack ($pop(v_i)$), push onto the stack ($push(v_i)$) and return the top of the stack (top).

Chin, Hu and Shing described the following heuristic linear-time algorithm.

Chin-Hu-Shing Algorithm

Input : - a sequence of weights of vertices in a polygon - $w_0, w_1, \ldots, w_{n-1}$
Find the smallest vertex w_i and shift indices of vertices such that w_i is the first element of a sequence; that is, $v_0 = w_i, v_1 = w_{(i+1)\bmod n}, \cdots, v_j = w_{(i+j)\bmod n}, \cdots$.

```
push($v_0$);
$i := 1$;
while $i \le n-1$ do
    if top $\ne v_0$ and top is large then
        cutoff(top); pop(top);
    else
        push($v_i$);
    fi;
od;
fan($v_0$);
```

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

What is especially interesting, in most cases algorithm CHS yields the optimal solution or a solution which takes only a few percent worse than the optimal one[3]. As was mentioned in the introduction the (worst case) error ratio is inversely proportional to n and is maximum when $n = 5$. In general, if value $t = \max_i\{\frac{v_i}{v_0}\}$ has upper bound, then Hu and Shing [HS-81] calculated the maximum error ratio for any given input n. This ratio is given by the following formula:

$$\text{maximum error ratio R } = \frac{t-1}{t^2+t+(n-4)}$$

For example, if $t = 2$, then $R = 1/(n+2)$, and if $t \cong \sqrt{n}$, and n approaches ∞, then $R = \frac{1}{2\sqrt{n}}$.

Fact 3.1 ([HS-80], [HS-81]) *Algorithm CHS finds a near-optimal partitioning of a polygon and a near-optimal order of computing the matrix chain products with the maximum error ratio* $\simeq 0.1547$ *(exactly it is* $\frac{\sqrt{3}}{6+3\sqrt{3}}$*).*

4 Reduction to the problem of finding an optimal triangulation in a basic polygon

The first step of our algorithm is a partition of a convex polygon into smaller nonintersecting *basic polygons.*

Fact 4.1 ([HS-80]) *There exists an optimal triangulation of a convex polygon containing arcs or sides between the smallest vertex and the second and the third smallest ones*

Define a *basic polygon* to be a polygon containing sides between the smallest vertex (v_0) and the second (v_1) and the third (v_{n-1}) ones. Fact 4.1 implies a partition of a convex n-gon into smaller nonintersecting basic subpolygons which are in an optimal triangulation. Such a partition can be found in the following way.

First divide a polygon into two parts by joining the smallest vertex with the second smallest one. Consider two obtained subpolygons independently. Assume that v_0, v_{n-1} are two smallest vertices and v_0 is the smallest one.

Observation 4.2 *Vertex* v_t *is the third smallest one if and only if* v_t *is a vertex with the greatest index between* v_0 *and* v_{n-1}*, such that there is no vertex less than* v_t *in the sequence* $v_1, v_2, \ldots, v_{t-1}$.

[3]less than 1 percent on the average, see simulation results in [Chin-78]

Using the observation above we can easy design parallel algorithm which divides a convex polygon into basic polygons.

Let $v_0, v_1, \dots v_{n-1}$ be the weights of a polygon with v_0 as the smallest vertex and v_{n-1} as the second smallest one. Solve the prefix minima problem with respect to weights $(v_1, \dots, v_{n-1})$. Join v_i with v_0 iff this vertex is the minimal in the range $v_1, \dots, v_i$. Hence we obtain the following fact.

Fact 4.3 *Partitioning of the polygon into nonintersecting basic subpolygons can be done 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 weak CRCW PRAM.*

5 Outline of a new algorithm

In this section we describe an overview of a new algorithm which produces the same partitioning as that of Chin-Hu-Shing. An input of this new algorithm is a basic polygon with v_0 as the smallest vertex, v_1 as the second smallest one, and v_{n-1} as the third smallest one.

First, our algorithm finds the set of candidates for h-arcs. We show a necessary condition for an arc to be an h-arc. Moreover this condition gives us the set of edges in a polygon which do not intersect, what implies that this set of candidates is not too big (it counts exactly n-3 arcs). Then from the set of all candidates for h-arc, we eliminate these edges which are not chosen by algorithm CHS. At the end we find all the fan-arcs.

Let us define a *candidate* to be an arc (v_i, v_j) such that for each k, $i < k < j$, the inequality $v_i < v_k$, $v_j < v_k$ hold. We show that arcs with this property are candidates for h-arcs (i.e., this is a necessary condition to be h-arc).

Observation 5.1 *No candidates intersect.*

Observation 5.2 *Let v_k, v_c be neighbours of vertex v_t. If v_t is* large *then the following condition holds*

$$v_c < v_t, \quad \text{and} \quad v_k < v_t.$$

Now we give some observations concerning algorithm CHS. We can see that during an execution of the procedure *cutoff*(v_t), with v_k and v_c as neighbours of v_t, two invariants hold:

1. $k < t < c$

2. for each i $(k < i < c,\ i \neq t)$, $v_t < v_i$

Hence we obtain the following crucial lemma.

Lemma 2
h-arc is always a candidate.

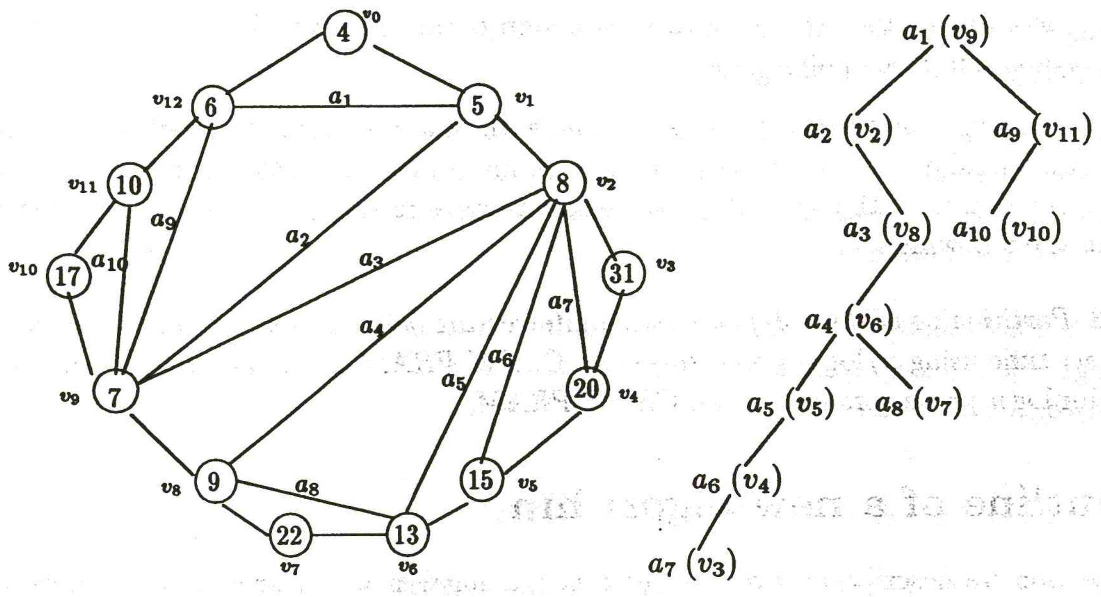

Figure 2: Partitioning of a convex polygon by candidates and the corresponding tree of candidates. In the tree each vertex is represented by the corresponding candidate and in brackets we present vertices which store given candidate.

Let us look at the polygon after finding all candidates. This polygon is completely divided into $n-2$ triangles. We introduce a tree of candidates. Triangle (v_i, v_j, v_l) (where $i < j < l$) is said to lie lower than arc (v_i, v_l). Let us take candidate (v_k, v_c). It is obvious that each such an arc (if $k < c-1$) has exactly one lower triangle. We can define the *tree of candidates* as follows.

For given candidate (v_k, v_c) we define its sons to be arcs (except sides of polygon) in the lower triangle for (v_k, v_c). It is easy to see that this tree is a binary tree (i.e., each vertex has at most two sons) with $n-3$ vertices.

Observation 5.3 *For each candidate (v_k, v_c), all its descendants are of the form (v_i, v_j), where $k \leq i < j \leq c$.*

Given a tree of candidates, we need to verify for all the candidates whether they are h-arcs.

Algorithm CHS begins by creating h-arcs and then non-divided subpolygon is partitioned by fan-arcs. Always when vertex v_t is cut new h-arc is added. Therefore during the exectuting the loop in algorithm CHS there is always only one non-divided convex polygon. Thus we obtain the following

Fact 5.4 *The loop in algorithm CHS finishes with exactly one convex polygon without internal arcs.*

This fact implies the following important observation.

Observation 5.5 *Let (v_k, v_c) be the h-arc obtained by algorithm CHS. Then all the candidates (v_i, v_j), for $k \leq i < j \leq c$, are generated by CHS algorithm as h-arcs.*

Proof: From the above fact, we see that there is no fan-arc in polygon $(v_k, v_{k+1}, \ldots, v_c)$. Hence this polygon must be triangulated by h-arcs. □

The following is a new algorithm for partitioning of a basic polygon.

Algorithm Par-CHS

```
{ Input :  basic polygon with vertices $v_0, v_1, \ldots, v_{n-1}$ }
  1. Create the tree of candidates.
  2. Verify all candidates - i.e., find all h-arcs.
  3. Find all fan arcs in a polygon.
```

Theorem 3
Algorithm Par-CHS produces the same partition of a convex polygon as CHS algorithm.

Proof: It follows from the previous comments and from the fact that all arcs divided a convex polygon into basic subpolygons are fan-arcs. □

6 An optimal parallel algorithm

In this section we show how to implement efficiently algorithm Par-CHS on CREW PRAM and CRCW PRAM machines.

Begin with building a tree of candidates. From previous observations we can establish condition when procedure *cutoff*(v_t) is called. Let v_k and v_c $(k < t < c)$ be two neighbours of v_t. Then a vertex v_t is poped off the stack if and only if all vertices between v_k and v_c (in the original convex polygon) are greater than v_t (and $v_k < v_t$ and $v_c < v_t$). That is, for each i, $k < i < c, i \neq t$, $v_t < v_i$.

Therefore, to find such a triple it is enough to find for each vertex v_t two vertices v_k and v_c which lie on the both sides of v_t and satisfy the above condition. It is clear that this is equivalent to the problem of finding all nearest smaller values. For each v_t choose v_k to be the nearest smaller vertex to the left side and v_c to be the nearest smaller vertex than v_t to the right side. And now create the set of pairs - (k, c) which are stored in the array *node(t)*.

Then we need to create the candidate's tree. That is, we have to find fathers for all nodes (i.e., for all pairs (k, c)). The root of the tree is a pair $(1, n-1)$ stored in *node(r)*, where r is this vertex which was cut off by arc (v_1, v_{n-1}). Definition of the father's relation implies that the father of candidate (v_i, v_j) is either $node(j) = (i, r)$, if $v_i < v_j$, or $node(i) = (l, j)$, if $v_j < v_i$, for some r or l. Hence we can find fathers in $O(1)$ time using n processors on a CREW PRAM.

From the previous section we know that a candidate is an h-arc iff

- it cuts off a large vertex (with respect to this arc) and
- all candidates below it (in the tree of candidates) are h-arcs too

One can easily show that if *node*$(t) = (k, c)$ and (v_k, v_c) is an h-arc, then (v_k, v_c) cuts off v_t as a large vertex. Hence to check whether given candidate is an h-arc we can solve straight-line program which evaluates boolean expressions. For each leaf *node*$(t) = (k, c)$ of the tree of candidates assign value *true* if vertex v_t is large with v_k and v_c as its two neighbours and *false* otherwise. For every internal vertex *node*$(t) = (k, c)$ assign the logical operation 'AND' if v_t is large with v_k and v_c as its two neighbours, otherwise set value *false*. Then compute a tree of expression. If *node*$(t) = (k, c)$ has computed value *true* then candidate (v_k, v_c) is an h-arc. But standard algorithms for expression evaluation (see e.g. [CV-88] or [Ryt-90]) run in at least $O(\log n)$ time on any of PRAM's models. Therefore we will need new ideas to solve this problem.

Let us look at the tree of candidates. Each candidate (v_k, v_c) is held in *node*(t). For each such a node we know the number of all its descendants (exactly c-k-2) and morever $t - 1$ is its infix number (see e.g. a tree from figure 2). Thus we can use the algorithm from lemma 1. So a node corresponding to a candidate (v_k, v_c) has value *true* if and only if it is an h-arc.

At the end we need only to find all other vertices in the polygon, i.e., all fan-arcs. From the previous step we have in the array *node*(i) some h-arcs, and all others candidates are not in the partitioning. Hence we can check whether node v_i was cut off during executing algorithm CHS, simply by checking if *node*(i) is an h-arc. If *node*(i) is not an h-arc, then join vertex v_i with a vertex v_0 and put this information into *node*(i). It can be done in constant time with n processors.

This completes the proof of the following result.

Theorem 4
The problem of finding a near-optimal order of matrix chain products and the problem of finding a near-optimal partition of a convex polygon can be solved

- *in $O(\log n)$ time using $n/\log n$ processors on a CREW PRAM*
- *in $O(\log\log n)$ time with $n/\log\log n$ processors on a weak CRCW PRAM*

7 Conclusion and extensions

We gave very fast optimal parallel algorithm for finding a near-optimal order of matrix chain product and for finding a near-optimal partitioning of a convex polygon. It runs in $O(\log n)$ time on a CREW PRAM and in $O(\log\log n)$ time on a weak CRCW PRAM. This algorithm has optimal linear time-processor product. These bounds seem to be optimal among linear-work algorithms on both CREW and CRCW PRAM.

One can improve these bounds if the domain of input values is restricted. In the matrix chain products problem there are always integer values of matrix dimension. Therefore if dimensions are drawn from a domain $[k \dots k+s]$ we can find a near-optimal order of matrix multiplications in $O(\alpha(n) + \log\log\log s)$ time with linear work on a priority CRCW PRAM. This result follows from the algorithm with the same bound for the nearest smaller values problem (and also the prefix minima problem) where the domain is restricted to the set $[0 \dots s]$ [Ber-92].

We can speed-up our optimal algorithm to $\alpha(n)$ time on a common CRCW PRAM if the difference between two succesive elements (i.e., dimension or weights of vertices of a polygon) is bounded by some constant k. It also follows from the result for the all nearest smaller values problem [BV-91].

As in [Cha-75] 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 $\tau(p, q, r) = \tau(r, p, q)$.

Acknowledgment

The author would like to thank B. Chlebus, K. Diks and W. Rytter for carefully reading draft of the paper and many helpful comments.

References

[Ber-92] O. Berkman, personal communication, 1992.

[BBGSV-89] O. Berkman, D. Breslauer, Z. Galil, B. Schieber, U. Vishkin, "Highly parallelizable problems", *Proceedings of the 21st Annual ACM Symposium on Theory of Computing* 1989, pp. 309–319.

[BSV-88] O. Berkman, B. Schieber, U. Vishkin, "Some doubly logarithmic optimal parallel algorithms based on finding all nearest smaller values", UMIACS-TR-88-79, University of Maryland, Institute for Advanced Computer Studies, 1988.

[BV-89] O. Berkman, U. Vishkin, "Recursive *-tree parallel data-structure", *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, IEEE Computer Society, 1989, pp. 196–202, *also UMIACS-TR-90-40, University of Maryland, Institute for Advanced Computer Studies, 1990*.

[BV-91] O. Berkman, U. Vishkin, "Almost fully-parallel parentheses matching", UMIACS-TR-91-103, University of Maryland, Institute for Advanced Computer Studies, 1991.

[Cha-75] 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, N.Y., 1975.

[Chin-78] F.Y. Chin, "An O(n) algorithm for determining a near-optimal computation order of matrix chain products", Communications of the ACM, Vol. 21, No. 7, 1978, pp. 544–549.

[CV-88] R. Cole, U. Vishkin, "Optimal parallel algorithms for expression tree evaluation and list ranking", *Proceedings of the 3rd Aegean Workshop Comput.*, 1988.

[GP-92] Z. Galil, K. Park, "Parallel dynamic programming", manuscript 1992.

[GR-88] A. Gibbons, W. Rytter, "Efficient parallel algorithms", Cambridge University Press, 1988.

[HLV-90] S-H.S. Huang, H. Liu, V. Viswanathan, "Parallel dynamic programming", *Proceedings of the 2nd IEEE Symposium on Parallel and Distibuted Processing*, 1990, pp. 497–500.

[HS-80] T.C. Hu, M.T. Shing, "Some theorems about matrix multiplication", *Proceedings of the 21st Annual Symposium on Foundations of Computer Science*, IEEE Computer Society, 1980, pp. 28–35.

[HS-81] T.C. Hu, M.T. Shing, "An O(n) algorithm to find a near-optimum partition of a convex polygon", Journal of Algorithms, Vol. 2, 1981, pp. 122–138.

[Ryt-88] W. Rytter, "On efficient parallel computations for some dynamic drogramming problems", Theoretical Computer Science, Vol. 59, 1988, pp. 297–307.

[Ryt-90] W. Rytter, "On parallel computation of expression and straight-line programs", Computer and Artificial Intelligence, Vol. 9, No. 5, 1990, pp. 427–429.

[Sch-87] B. Schieber, "Design and analysis of some parallel algorithms", PhD Thesis, Tel Aviv University, Tel Aviv, 1987.