MAX-PLANCK-INSTITUT FÜR INFORMATIK

Prefix Graphs and Their Applications

Shiva Chaudhuri Torben Hagerup

MPI–I–94–145 August 1994

Im Stadtwald
66123 Saarbrücken
Germany

Prefix Graphs and Their Applications

Shiva Chaudhuri Torben Hagerup

MPI-I-94-145 August 1994

Prefix Graphs and Their Applications *

Shiva Chaudhuri and Torben Hagerup

Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany
shiva@mpi-sb.mpg.de torben@mpi-sb.mpg.de

Abstract. The *range product problem* is, for a given set S equipped with an associative operator $\circ$, to preprocess a sequence $a_1, \ldots, a_n$ of elements from S so as to enable efficient subsequent processing of queries of the form: Given a pair (s,t) of integers with $1 \leq s \leq t \leq n$, return $a_s \circ a_{s+1} \circ \cdots \circ a_t$. The generic range product problem and special cases thereof, usually with $\circ$ computing the maximum of its arguments according to some linear order on S, have been extensively studied. We show that a large number of previous sequential and parallel algorithms for these problems can be unified and simplified by means of prefix graphs.

1 Introduction

In 1983 Chandra, Fortune and Lipton introduced a computational paradigm closely related to the Ackermann function and used it to study the computation of semigroup products on unbounded-fanin circuits [6, 7]. Since then the paradigm was rediscovered several times, under different names and in different guises, and exploited in the design of sequential and parallel algorithms. In particular, Berkman and Vishkin developed the "recursive star-tree data structure" [4] and used it in a series of papers. We unify much of the previous research by observing that it is concerned simply with solving special cases of the range product problem, and by giving a generic algorithm for the range product problem in terms of graphs known as *prefix graphs*. We prove few new results; however, the machinery developed here allows us, with a modest effort, to obtain simpler proofs of the results of several previous papers, and to exhibit the intimate connection between such seemingly disparate problems as binary addition on a circuit and linear-range merging (i.e., merging sequences of length n with elements in $\{1, \ldots, n\}$) on a PRAM. We hope that our effort at unification and simplification will contribute to a wider understanding and appreciation of the underlying paradigm.

2 Definition of Prefix Graphs

Informally, a prefix graph has n vertices arranged in a column on the left and n vertices arranged in a column on the right, some vertices in between, and enough

* Supported by the ESPRIT Basic Research Actions Program of the EU under contract No. 7141 (project ALCOM II).

edges to allow us to go from left to right, provided that we also go down by at least one level.

Definition 1. For all $n \in \mathbb{N}$, a *prefix graph* of *width* n is a directed acyclic graph $G = (V, E)$ with n distinguished *input vertices* $x_1, \ldots, x_n$ of indegree zero and n distinguished *output vertices* $y_1, \ldots, y_n$ of outdegree zero and with the following properties, where the *span* of a vertex $v \in V$, span(v), is defined as $\{i : 1 \leq i \leq n$ and G contains a path from x_i to $v\}$.

(1) For $i = 1, \ldots, n$, span$(y_i) = \{1, \ldots, i-1\}$ (for $i = 1$ this is $\emptyset$);

(2) For all $v \in V$, span(v) is either empty or an "interval" of the form $\{s, \ldots, t\}$, for some integers s and t with $1 \leq s \leq t \leq n$;

(3) Any two vertices in V with a common successor have disjoint spans.

The *depth* of a vertex v in G is the length of a longest path in G from an input vertex to v, and the depth of G is the maximum depth of any of its vertices.

Note that there is a natural linear order on the set of edges entering a vertex v in a prefix graph. If $e = (u, v)$ and $e' = (u', v)$ are two such edges, e precedes e' if and only if the elements of the span of u are smaller than the elements of the span of u' (if either span is empty, the order is undefined, but irrelevant). We call this order the *canonical ordering* of the edges entering v.

3 Generic Prefix and Range Product Algorithms

In the context of a fixed semigroup $(S, \circ)$, the *composition problem* defined by n elements $a_1, \ldots, a_n$ of S is to compute $a_1 \circ \cdots \circ a_n$. The corresponding *prefix product problem* is to compute all the prefix products $a_1, a_1 \circ a_2, \ldots, a_1 \circ \cdots \circ a_n$, and the corresponding *range product problem* is to preprocess the sequence $a_1, \ldots, a_n$ so that in response to a *range query* $[s, t]$, where $1 \leq s \leq t \leq n$, one can quickly compute $a_s \circ \cdots \circ a_t$.

Prefix graphs suggest very natural and simple reductions of the prefix product problem to the composition problem, and of the range product problem to the prefix product problem. In order to solve the prefix product problem defined by n elements $a_1, \ldots, a_n$ of a semigroup $(S, \circ)$, take a prefix graph $G = (V, E)$ of width n, apply $a_1, \ldots, a_n$ to the inputs of G and let values of S "percolate" through G from the inputs to the outputs, each vertex composing the values reaching it over its incoming edges, in the order given by the canonical ordering of these edges, and sending the resulting product over all of its outgoing edges (we say that the vertex solves its *local composition problem*). It is easy to see by induction on the vertex depth that each vertex with span $\{s, \ldots, t\}$ computes $a_s \circ \cdots \circ a_t$; in particular, the solution to the prefix product problem can be read off the output vertices.

In order to solve the range product problem for $a_1, \ldots, a_n$, we begin by carrying out the same computation. Further, each vertex in V computes all suffix products of the sequence of values that reached it (it solves its *local suffix problem*) and saves these, which ends the preprocessing.

For any set J of the form $J = [s,t] = \{s, s+1, \ldots, t\}$ with $1 \leq s \leq t \leq n$, let $P(J) = a_s \circ a_{s+1} \circ \cdots \circ a_t$. We claim that if $v \in V$ is of depth $d \geq 1$ in G and $\text{span}(v) = [s,t]$, then for any integer m with $s \leq m \leq t$, $P([m,t])$ is the product of at most d values computed during the preprocessing. The proof is by induction on d, and the claim is obvious if $d = 1$. Otherwise choose (the unique) $(u,v) \in E$ such that $m \in \text{span}(u)$, apply the induction hypothesis to compute $P(\text{span}(u) \cap [m,t])$ and note that if $J = [m,t] \backslash (\text{span}(u) \cap [m,t]) \neq \emptyset$, then $P(J)$ is one of the suffix products stored at v.

Applying the claim above to the output vertices, we see that if G is of depth $d \geq 1$, then the answer to any range query can be obtained as the composition of at most d of the values computed in the preprocessing phase. The relevant values can be found in $O(d)$ sequential time during a backwards scan in G, as in the proof above; we omit the details.

The algorithms that we will describe always just simulate the generic prefix and range product algorithms. The only variable parameters will be the semigroup $(S, \circ)$ under consideration and the model of computation.

4 Existence of Prefix Graphs

Define $I_0 : \mathbb{N} = \{1, 2, \ldots\} \to \mathbb{N}$ by $I_0(n) = \lceil n/2 \rceil$, for all $n \in \mathbb{N}$. Inductively, for $k = 1, 2, \ldots$, define $I_k : \mathbb{N} \to \mathbb{N}$ by $I_k(n) = \min\{i \in \mathbb{N} : I_{k-1}^{(i)}(n) = 1\}$, for all $n \in \mathbb{N}$, where superscript (i) denotes i-fold repeated application. Finally, for all $n \in \mathbb{N}$, take $\alpha(n) = \min\{k \in \mathbb{N} : I_k(n) \leq k\}$. It can be shown that for all $n, k \in \mathbb{N}$, $I_{k+1}(n) \leq I_k(n) \leq I_k(n+1)$ and, if $n \geq 2$, $I_k(n) < n$.

An important fact about prefix graphs is that for all $n, k \in \mathbb{N}$, there is a prefix graph $G_{n,k}$ of width n, depth at most $2k$ and $O(nkI_k(n))$ edges. The simplest demonstration of this fact proceeds by simultaneous induction on n and k. We hence describe $G_{n,k}$ in terms of graphs $G_{n',k'}$, where (n',k') precedes (n,k) lexicographically. Let $m = I_{k-1}(n)$. We will assume that m divides n. The construction of $G_{n,k}$ is shown in Fig. 1. We take n input vertices and n output vertices and partition both input and output vertices into groups of size m. The vertices in corresponding groups are connected via copies of $G_{m,k}$. Furthermore we create a new vertex for each input or output group and add edges from each input to the new vertex associated with its group and from each vertex associated with an output group to all vertices in its group. Finally, if $k > 1$, we connect the n/m new vertices on the left with the n/m new vertices on the right via a copy of $G_{n/m,k-1}$. If $k = 1$, we have $n/m = 2$, and we instead identify the upper new vertex on the left with the lower new vertex on the right, i.e., we connect the new vertices via a prefix graph of depth zero and with no edges.

It is easy to see that the resulting graph is a prefix graph of depth at most $2k$. Assume that for all (n',k') that precede (n,k) lexicographically, $G_{n',k'}$ contains at most $2n'k'I_{k'}(n')$ edges. Then the copies of $G_{m,k}$ contribute a total of at most $2nkI_k(m) = 2nk(I_k(n) - 1)$ edges, the copy of $G_{n/m,k-1}$, if present, contributes at most $2(n/m)(k-1)I_{k-1}(n/m) \leq 2n(k-1)$ edges, and with the remaining $2n$ edges this yields a grand total of at most $2nkI_k(n)$ edges.

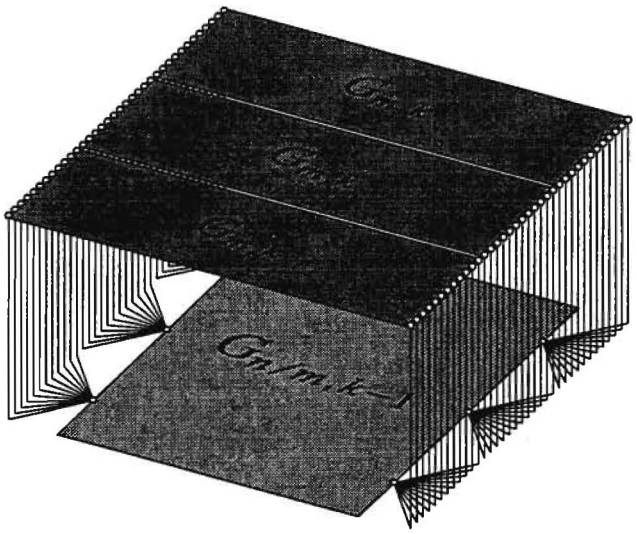

Fig. 1. A doubly-recursive construction of the prefix graph $G_{n,k}$.

The construction above, suitably modified if m does not divide n, suffices for all our applications, except those in Section 5.5. We now describe an alternative, but similar construction that suffices for all our applications, incorporates rounding and leads more directly to a fast parallel construction algorithm, one recursion of depth $\Theta(\log n)$ having been converted to iteration. The vertices in the graph will be classified as either *front* or *back vertices* as they are introduced; input vertices are always front vertices, and output vertices that are not also input vertices are back vertices.

The induction now is only on k. Without loss of generality we will assume that n is a power of 2. Take $l = I_k(n) - 1$ and let $m_0 \geq m_1 \geq \cdots \geq m_l \geq m_{l+1}$ be a sequence of powers of 2 with $m_0 = n$, $m_l = O(1)$ and $m_{l+1} = 1$; the exact values will be specified later. $G_{n,k}$ consists of $l+1$ "layers", illustrated in Fig. 2. Layer i, for $i = 0, \ldots, l$, partitions both input and output vertices of $G_{n,k}$ into groups of m_{i+1} consecutive vertices, joins all input vertices in each group to a new front vertex, joins one new back vertex for each group to all output vertices in the group, and finally connects every set of m_i/m_{i+1} consecutive front vertices to the corresponding back vertices via a copy of $G_{m_i/m_{i+1},k-1}$.

It is easy to see that the graph $G_{n,k}$ thus constructed is indeed a prefix graph (cf. Fig. 2): For every input vertex x and every higher-numbered output vertex y, exactly one of the layers contains a path from x to y (for this it is essential that the groups in layer l are trivial, i.e., of size 1). We still need to choose $m_1, \ldots, m_l$, which will depend on a constant parameter $q \in \mathbb{N}$. If $k = 1$, take $m_i = I_0^{(i)}(n) = n/2^i$, for $i = 1, \ldots, l$. The recursive construction then depends only on $G_{2,0}$, which we take to be the prefix graph of width 2 with no edges. If $k \geq 2$, we will assume for convenience that $((2q+1)I_1(n))^{2q} \leq n$; this

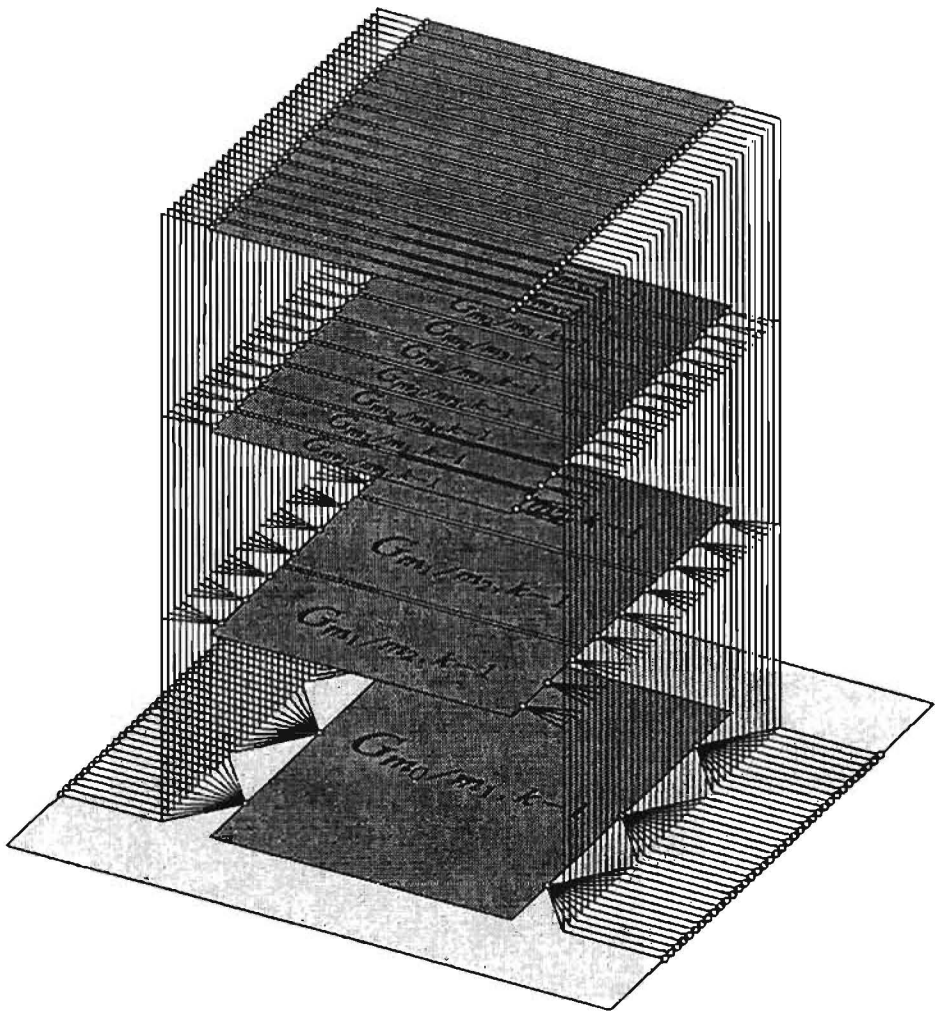

Fig. 2. An iterative-recursive construction of the prefix graph $G_{n,k}$.

excludes only a finite set of values of n. We then take m_i as the smallest power of 2 no smaller than $((2q+1)I_{k-1}^{(i)}(n))^{2q}$, for $i = 1, \ldots, l$; one can show that $I_{k-1}^{(I_k(n)-1)}(n) = 2$ for all $n, k \geq 2$, so that indeed $m_l = O(1)$, as required.

Let V_F and V_B be the sets of front and back vertices, respectively, in $G_{n,k}$, and denote by $\deg(v)$ the indegree of each vertex v. We are interested in the quantity

$$R_q(n,k) = \sum_{v \in V_F} \sqrt{\deg(v)|\mathrm{span}(v)|} + \sum_{v \in V_B} (\deg(v))^q,$$

called the *order-q root-span* of $G_{n,k}$. We will show that $R_q(n,k) = O(nk(I_k(n))^q)$.

If $k \geq 2$, assume by induction that $R_q(m, k-1) \leq rm(k-1)(I_{k-1}(n))^q$ for some constant r and for all $m \in \mathbb{N}$, and reconsider the construction of $G_{n,k}$. Input vertices of $G_{n,k}$ contribute nothing to $R_q(n,k)$, whereas the output (back) vertices have indegree $l+1 = I_k(n)$ and thus contribute a total of $n(I_k(n))^q$. For $i = 0, \ldots, l$, layer i contains n/m_i copies of $G_{m_i/m_{i+1},k-1}$. The (front) input vertices of these copies contribute exactly n to $R_q(n,k)$. We will determine the remaining contribution of layer i to $R_q(n,k)$. Consider three cases. (1) $k = 1$: The remaining contribution is zero. (2) $k \geq 2$ and $i = l$: Since $m_l = O(1)$, the remaining contribution is $O(n)$. (3) $k \geq 2$ and $i < l$: Since the size of the span of every vertex in the copies of $G_{m_i/m_{i+1},k-1}$ increases by a factor of m_{i+1} when the copies are incorporated into $G_{n,k}$, the remaining contribution of layer i to $R_q(n,k)$ is at most

$$\frac{n}{m_i}\sqrt{m_{i+1}}R_q(m_i/m_{i+1}, k-1) \leq \frac{rn(k-1)(I_{k-1}(m_i/m_{i+1}))^q}{\sqrt{m_{i+1}}}.$$

We have $m_i/m_{i+1} \leq 2(I_{k-1}^{(i)}(n))^{2q}$ (the factor of 2 is due to the rounding to the nearest power of 2). One can show that for all $m \in \mathbb{N}$, $I_{k-1}(2m^{2q}) \leq (2q+1)I_{k-1}(m)$. Hence $I_{k-1}(m_i/m_{i+1}) \leq (2q+1)I_{k-1}(I_{k-1}^{(i)}(n)) = (2q+1)I_{k-1}^{(i+1)}(n)$. But then the contribution of layer i to $R_q(n,k)$ is at most

$$\frac{rn(k-1)((2q+1)I_{k-1}^{(i+1)}(n))^q}{\sqrt{((2q+1)I_{k-1}^{(i+1)}(n))^{2q}}} = rn(k-1).$$

Putting the cases together and observing that the number of layers is $l+1 = I_k(n)$, we altogether obtain $R_q(n,k) \leq n(I_k(n))^q + O(n) + I_k(n)(n + rn(k-1))$, which, for the constant r chosen sufficiently large, is bounded by $rnk(I_k(n))^q$.

We summarize the results of this section as follows.

Theorem 2. *For all $n, k \in \mathbb{N}$ and all fixed $q \in \mathbb{N}$, there is a prefix graph of width n, depth at most $2k$ and order-q root-span $O(nk(I_k(n))^q)$. In particular, for all $n \in \mathbb{N}$ and all fixed $q \in \mathbb{N}$, there is a prefix graph of width n, depth $O(\alpha(n))$ and order-q root-span $O(n(\alpha(n))^{q+1})$.*

The lesson to be learned from Theorem 2 is that if we allocate resources (such as time or processors) proportional to $\sqrt{\deg(v)|\mathrm{span}(v)|}$ to each front vertex v of a prefix graph, and proportional to any fixed power of $\deg(v)$ to each back vertex v, then the total amount of resources used will be only slightly superlinear. In most cases it suffices to allocate resources proportional to $\deg(v)$ to each vertex v (if v is a front vertex, $\deg(v) \leq |\mathrm{span}(v)|$ and hence $\deg(v) \leq \sqrt{\deg(v)|\mathrm{span}(v)|}$); this can be viewed as placing a constant amount of resources (e.g., one processor) at each edge of the prefix graph.

Assuming constant-time access to a few tables, most notably one that gives the value of $I_k^{(i)}(m)$ for all positive integers m, k and i with $m \leq n$, $k \leq \alpha(n)$ and $i \leq \log n$, a suitable representation of a prefix graph with the properties described in Theorem 2 can be computed in constant time on a CREW PRAM

with $nk(I_k(n))^q$ processors, and hence in $O(nk(I_k(n))^q)$ sequential time. Suitable tables can be constructed in constant time on an n-processor CRCW PRAM, and hence in linear sequential time, whereas the best construction known for the CREW PRAM needs $O(\log(\log^* n))$ time and $O(n)$ operations. We omit the details due to lack of space, referring the reader to [4] and [5] for some of the arguments.

5 Applications of Prefix Graphs

In this section we describe a number of concrete instances of the generic algorithms introduced in Section 3. In each case Theorem 2 is used to bound the resource requirements of the resulting algorithms. We give algorithms with slightly superlinear processor-time products; simple arguments presented in Section 6 reduce this quantity to $O(n)$ in each case. Our examples span sequential computation, unbounded-fanin circuits and PRAMs.

5.1 Sequential Range Products

Assuming that the operator $\circ$ of the semigroup $(S, \circ)$ can be evaluated in constant sequential time, it is trivial to verify that each vertex v of a prefix graph can solve its local composition and suffix problems in linear time $O(\deg(v))$. By the generic prefix and range product algorithms and Theorem 2, we obtain the following result of Alon and Schieber [2].

Theorem 3. *For all $n, k \in \mathbb{N}$, instances of size n of the range product problem can be solved with preprocessing time $O(nkI_k(n))$ and query time $O(k)$.*

5.2 Addition on Unbounded-Fanin Circuits

In this section we consider unbounded-fanin circuits with AND, OR and NOT gates. The size of a circuit is defined to be the total number of gates and wires in the circuit, and the depth is the longest path from an input to an output. The *find-first problem* of size n is, given n bits $u_1, \ldots, u_n$, to compute the n bits $v_1, \ldots, v_n$ with $v_i = 1$ iff $u_i = 1$ and $u_1 = u_2 = \cdots = u_{i-1} = 0$, for $i = 1, \ldots, n$. As shown in [6, Theorem 3.4], find-first problems of size n can be solved by a circuit of constant depth and size $O(n)$.

Consider the problem of computing the $(n+1)$-bit sum $z_{n+1} \cdots z_1$ of two n-bit numbers $x_n \cdots x_1$ and $y_n \cdots y_1$. There is a well-known reduction of this problem to that of computing prefix sums of elements of the semigroup with three elements, S, R and P, and the operator $\circ$ defined as follows: For all $u, v \in \{\mathrm{S}, \mathrm{R}, \mathrm{P}\}$, $u \circ v = v$ if $v \in \{\mathrm{S}, \mathrm{R}\}$, and $u \circ \mathrm{P} = u$. For $i = 1, \ldots, n$, let $a_i = \mathrm{S}$ if $x_i = y_i = 1$, $a_i = \mathrm{R}$ if $x_i = y_i = 0$ and $a_i = \mathrm{P}$ otherwise. S, R and P are "set", "reset" and "propagate" indicators for the carry bit. It is easy to see that there is a carry into bit position $i+1$ iff $c_{i+1} = \mathrm{S}$, for $i = 1, \ldots, n$, where $c_{i+1} = a_1 \circ \ldots \circ a_i$, so that adding $x_n \cdots x_1$ and $y_n \cdots y_1$ essentially boils down

to computing $c_2, \ldots, c_{n+1}$. For this, imagine that the edges of a prefix graph can carry the values S, R and P and that the vertices can compute the product (with operator $\circ$) of the values on their incoming edges. Then the outputs of the prefix graph on input $a_1, \ldots, a_n$ are $c_2, \ldots, c_{n+1}$. The computation at a vertex of the prefix graph is essentially a find-first problem, since the value obtained by composing the values in its input sequence is simply the rightmost non-P value in the sequence, or P if there is no such value. Using the circuit of [6] mentioned above, we can simulate the computation at v with a boolean circuit of constant depth and size $O(\deg(v))$. An edge in the prefix graph can be simulated with two parallel wires, so that we altogether obtain a circuit of size and depth within constant factors of the size and depth of the prefix graph. Applying Theorem 2, we get the following result of [7, Corollary 3.5].

Theorem 4. *For all $n, k \in \mathbb{N}$, two n-bit numbers can be added with a circuit of depth $O(k)$ and size $O(nkI_k(n))$.*

As an alternative to the theorem above, two n-bit numbers can also be added with a circuit of depth $O(\alpha(n))$ and size $O(n)$. This follows by choosing $k = \alpha(n)$, using prefix-product circuits of [10] of logarithmic depth and linear size to reduce the problem size by a factor of $\Theta((\alpha(n))^2)$ and using Theorem 4 to solve the reduced problem, much in the spirit of Section 6.

In the remainder of Section 5 we describe PRAM algorithms in which the local composition and suffix problems are solved in parallel at each vertex of a prefix graph.

5.3 Segmented Broadcasting

The *segmented broadcasting problem* of size n is, given an array A of n cells, some of which contain *significant* objects, while others do not, to replace each insignificant object by the nearest significant object to its left, if any. If we let S be the set of objects and denote an insignificant object by 0, the segmented broadcasting problem is the prefix product problem associated with the semigroup $(S, \circ)$, where for all $a, b \in S$, $a \circ 0 = a$ and $a \circ b = b$ if $b \neq 0$. Instances of size n of the corresponding composition problem can be solved in constant time on an n-processor CRCW PRAM by a straightforward simulation of the find-first circuits of [6] mentioned in the previous subsection. By the generic prefix product algorithm and Theorem 2, we therefore obtain the following result of Berkman and Vishkin [4] and Ragde [11, Theorem 4], who solved versions of segmented broadcasting called *all nearest zero bits* and *ordered chaining*, respectively.

Theorem 5. *For all $n, k \in \mathbb{N}$, segmented broadcasting problems of size n can be solved in time $O(k)$ on a CRCW PRAM with $nkI_k(n)$ processors.*

5.4 Linear-Range Merging

An instance of the *linear-range merging problem* of size n is to merge two sorted sequences, each containing n elements in the range $1..n$. A linear-range merging

problem of size n reduces to a segmented broadcasting problem of size $2n$. To see this, multiply each input element by 2 and subtract 1 from each element in one input sequence only. This ensures that no value occurs in both input sequences. It now suffices to determine for each input element x its rank in the opposite input sequence, i.e., the number of elements $\leq x$ in that sequence, since the position of the element in the output sequence can be taken as its rank in the opposite input sequence plus its position in its own sequence.

The remaining task therefore is, given a sorted sequence $X = (x_1, \ldots, x_n)$ of n integers in the range $1..2n$, to compute a table $Rank[1..2n]$ such that for $j = 1, \ldots, 2n$, $Rank[j]$ is the rank of j in X (this task is carried out twice, once for each input sequence). Suppose that we begin by computing preliminary ranks as follows: Initialize $Rank[j]$ to 0, for $j = 1, \ldots, 2n$, and take $x_{n+1} = \infty$, where ∞ is an integer larger than $2n$. Then, for $i = 1, \ldots, n$, if $x_i \neq x_{i+1}$, then set $Rank[x_i] = i$. This computes the correct rank of every value that occurs in X. Furthermore, the correct value to be stored in each entry in the rank table is the maximum of the preliminary ranks stored in the entries to its left (including the entry under consideration), so that the problem at hand is the segmented broadcasting problem defined by the preliminary ranks.

By Theorem 5 and the discussion above, linear-range merging problems can be solved very efficiently on a CRCW PRAM. Exploiting additional information furnished by the reduction from merging to segmented broadcasting, however, we can solve linear-range merging problems equally efficiently on the CREW PRAM: For $j = 1, \ldots, 2n$, initialize $Rank[j]$ to $(j, 0, 0)$, rather than to 0. Then, for $i = 1, \ldots, n$, instead of setting $Rank[x_i] = i$ if $x_i \neq x_{i+1}$, set $Rank[x_i] = (x_i, x_{i+1}, i)$ if $x_i \neq x_{i+1}$. The third component of each triple is the actual rank information, the second component is a pointer to the next triple with a nonzero third (or second) component, and the first component is just the position of the triple itself. Let S be the set of triples of integers and define the semigroup $(S, \circ)$ as follows: For all $(a_1, b_1, c_1), (a_2, b_2, c_2) \in S$, $(a_1, b_1, c_1) \circ (a_2, b_2, c_2) = (a_1, b_1, c_1)$ if $c_2 = 0$, and $(a_1, b_1, c_1) \circ (a_2, b_2, c_2) = (a_2, b_2, c_2)$ otherwise.

Let $(a_1, b_1, c_1), \ldots, (a_n, b_n, c_n)$ be the set of triples generated for an instance of the linear-range merging problem. For such a sequence, it is easy to see that for any integers s and t with $1 \leq s \leq t \leq n$, $(a_s, b_s, c_s) \circ \cdots \circ (a_t, b_t, c_t)$ is (a_i, b_i, c_i) for the unique $i \in \{s, \ldots, t\}$ with $b_i > a_t$, or (a_s, b_s, c_s) if there is no such i. As a consequence, if we execute the associated instance of the generic prefix product algorithm on a prefix graph G, the local composition problem occurring at each vertex v in G can be solved in constant time even on a CREW PRAM with $\deg(v)$ processors.

The final rank table can be obtained from the prefix products, computed above, of the preliminary rank table by replacing each triple by its third component, which is the actual rank information. Using Theorem 2, we therefore obtain the following result, due to Berkman and Vishkin [5].

Theorem 6. *For all $n, k \in \mathbb{N}$, linear-range merging problems of size n can be solved in $O(k)$ time on a CRCW PRAM or a CREW PRAM with $nkI_k(n)$ processors. In the case of the CREW PRAM, an appropriate prefix graph must*

be supplied as part of the input.

Using a slightly smaller prefix graph, we obtain the following new result.

Theorem 7. *For all $n, k \in \mathbb{N}$, two sorted sequences of length n each of integers drawn from a range of size $O(n/I_k(n))$ can be merged in $O(k)$ time using $O(n)$ operations on a CRCW PRAM or a CREW PRAM. In the case of the CREW PRAM, an appropriate prefix graph must be supplied as part of the input.*

5.5 Prefix and Range Maxima of c-Bounded Sequences

A sequence $a_1, \ldots, a_n$ of integers is called *c-bounded*, for $c \in \mathbb{N}$, if $|a_i - a_{i+1}| \leq c$, for $i = 1, \ldots, n-1$. In this section we develop algorithms for the prefix maxima and range maxima problems for c-bounded input sequences. These results were first proved by Berkman and Vishkin [4]. The following simple fact is a consequence of [8, Theorem 4(c)].

Lemma 8. *For all $n, m \in \mathbb{N}$, the maximum of n integers $a_1, \ldots, a_n$ with $\max_{1 \leq i \leq j \leq n} |a_i - a_j| < m$ can be computed in constant time on a CRCW PRAM with $O(n + \sqrt{m})$ processors.*

If a c-bounded sequence $a_1, \ldots, a_n$ is applied to the inputs of a prefix graph and each vertex computes the maximum of the values entering it, the maximum difference between two values entering a vertex v will be at most $c(|\text{span}(v)| - 1)$. A front vertex v can therefore execute the algorithm of Lemma 8 with $O(\deg(v) + \sqrt{c|\text{span}(v)|})$ processors; a back vertex v can use the trivial brute-force algorithm requiring $(\deg(v))^2$ processors. By Theorem 2, we obtain:

Theorem 9. *For all $n, k, c \in \mathbb{N}$, the prefix maxima of a c-bounded sequence of length n can be computed in $O(k)$ time on a CRCW PRAM with $O(\sqrt{cnk}(I_k(n))^2)$ processors.*

We will solve the range maxima problem for c-bounded sequences by executing the algorithm of Theorem 9 (or, rather, the corresponding suffix maxima algorithm) at each vertex of a prefix graph $G = (V, E)$. For this we must establish an upper bound $c(v)$ on the difference between successive values entering each vertex $v \in V$. Let $\Gamma^-(v) = \{u \in V : (u, v) \in E\}$ and note that $2\max\{c|\text{span}(u)| : u \in \Gamma^-(v)\}$ is such an upper bound (in fact, the factor of 2 is not needed). By induction, one can show that $|\text{span}(u)| = |\text{span}(u')|$ for all $u, u' \in \Gamma^-(v)$, provided that v is a front vertex, i.e., in this case we can take $c(v) = 2c|\text{span}(v)|/|\Gamma^-(v)| = 2c|\text{span}(v)|/\deg(v)$.

Plugging the value for $c(v)$ determined above into the processor count of Theorem 9, we see that the number of processors needed at a front vertex v is $O(\sqrt{2c|\text{span}(v)|/\deg(v)}\,\deg(v)k(I_k(n))^2) = O(\sqrt{c\deg(v)|\text{span}(v)|}k(I_k(n))^2)$. By Theorem 2, this sums over all front vertices to $O(\sqrt{cnk^2}(I_k(n))^3)$. Since suffix maxima problems are simple to solve in constant time with a cubic number of processors, the same number of processors suffices for the back vertices. Noting that once the prefix product problem has been solved, the local suffix problem can be solved for all vertices in parallel, we have:

Theorem 10. *For all $n, k, c \in \mathbb{N}$, the preprocessing of a c-bounded sequence of length n for subsequent sequential range queries in $O(k)$ time can be done in $O(k)$ time on a CRCW PRAM with $O(\sqrt{c}nk^2(I_k(n))^3)$ processors.*

5.6 Randomized Prefix and Range Maxima

The algorithm in the previous subsection was able to deal only with c-bounded sequences because maxima of general sequences cannot be computed deterministically in constant time. The situation changes if we allow randomization. However, since the randomized algorithm is applied to many small inputs, namely once at each vertex, we have to cope with the failure of the execution at some vertices. The number of affected vertices being small, we can subsequently allocate enough resources to each such vertex, by means of an algorithm for so-called *interval allocation* [9], to let it run a deterministic algorithm. We now provide the details.

Lemma 11. *Let $f, g, h : \mathbb{N} \to \mathbb{N}$ satisfy $n \leq f(n) \leq g(n)$ for all $n \in \mathbb{N}$ and suppose that instances of size n of a composition problem can be solved in constant time both with $nh(n)$ processors and failure probability at most $1/g(n)$ and deterministically with $f(n)$ processors. Then, for every $k \in \mathbb{N}$, instances of size n of the corresponding prefix product problem can be solved with probability at least $1 - 1/g(f^{-1}(\sqrt{n})) - 2^{-\sqrt{n}}$ using $O(k)$ time and $nh(n)$ processors plus the resources needed for solving k successive interval allocation problems of size $nI_k(n)$. Here $f^{-1}(\sqrt{n})$ is to be understood as $\min\{m \in \mathbb{N} : f(m) \geq \sqrt{n}\}$.*

Proof. We allocate $h(n)$ processors to each edge of a prefix graph of width n and depth $2k$ with at most $rnI_k(n)$ edges, where $r \geq 1$ is a constant, and proceed level by level from the input vertices to the output vertices. At each level, each vertex first carries out five attempts to solve its local composition problem using the given randomized algorithm. Subsequently each vertex v at which all trials failed requests $f(\deg(v))$ processors and applies the given deterministic algorithm, after which we proceed to the next level.

The analysis must bound the probability that too many processors are requested. Assume that n is large enough to make $rI_k(n) \leq n$. Take $n_0 = f^{-1}(\sqrt{n})$ and say that a vertex v is of *high degree* if $\deg(v) \geq n_0$. Since $g(n_0) \geq f(n_0) \geq \sqrt{n}$, the probability that all five trials fail for some high-degree vertex is at most $rnI_k(n)(1/g(n_0))^5 \leq 1/g(n_0)$. The number of processors requested by the low-degree vertices is a weighted sum $X = \sum_v w_v X_v$ of independent Bernoulli variables, where $w_v = f(\deg(v)) \leq f(n_0 - 1) \leq \sqrt{n}$ and $E(X_v) = 1/g(\deg(v))$ for all v. We have $E(X) = \sum_v w_v E(X_v) \leq \sum_v f(\deg(v))/g(\deg(v)) \leq rnI_k(n)$. By the Chernoff bound of Raghavan for weighted sums [12, Theorem 1], $\Pr(X \geq 6rnI_k(n)) \leq 2^{-6rnI_k(n)/\sqrt{n}} \leq 2^{-\sqrt{n}}$.

Abstractly speaking, the argument above shows how to solve an instance of a certain problem at each vertex of a prefix graph, given a randomized and a (more wasteful) deterministic algorithm for the problem under consideration.

The analysis of Lemma 11 applies, in particular, when the problem at each vertex is its local suffix problem, and the randomized algorithm used is the prefix product algorithm of Lemma 11. Recalling that all the local suffix problems can be solved in parallel, we obtain an algorithm for the range product problem.

A CRCW PRAM with n processors can compute the maximum of n numbers in constant time with probability $1 - 2^{-n^{\Omega(1)}}$ [1, Theorem 3.9], and a deterministic algorithm for the problem that uses n^2 processors and constant time is obvious. Furthermore, interval allocation problems of size n can be solved in $O(\log^* n)$ time on an n-processor CRCW PRAM with probability $1 - 2^{-n^{\Omega(1)}}$ [9, Theorem 5.1], while processor allocation is free in the *parallel comparison-tree* (PCT) model of Valiant [13]. Using these facts in the general framework, we obtain the following results of [3].

Theorem 12. *For all $n, k \in \mathbb{N}$, range maxima preprocessing problems of size n can be solved (a) in $O(k \log^* n)$ time on a CRCW PRAM and (b) in $O(k)$ time on a PCT, so that in each case the number of processors needed is $nkI_k(n)$, the resulting sequential query time is $O(k)$, and the failure probability is $2^{-n^{\Omega(1)}}$.*

6 Making the algorithms work-optimal

In this section we show how to make all of the algorithms work-optimal without affecting the time performance. For any operator $\circ$ such that $a \circ b$ can be evaluated in constant sequential time (which is the case in all our applications), it is easy to see that range product problems of size n can be preprocessed in $O(\log n)$ time with $O(n)$ operations for a subsequent query time of $O(\log n)$ by means of a balanced binary tree.

Proposition 13. *If a range product problem of size n can be preprocessed in time $t(n)$ with $O(n2^{t(n)})$ operations, yielding query time $O(t(n))$, then it can be preprocessed in time $O(t(n))$ with $O(n)$ operations, yielding query time $O(t(n))$.*

Proof. Partition the input elements into $O(n/2^{t(n)})$ groups of size $O(2^{t(n)})$ each, preprocess each group using the optimal logarithmic-time algorithm above and preprocess the group products using the nonoptimal algorithm assumed in the proposition, each of which uses $O(t(n))$ time and $O(n)$ operations. Since each range is the disjoint union of two ranges within groups and one range spanning a number of whole groups, we can subsequently answer any range query in $O(t(n))$ time.

Proposition 14. *If a range product problem can be preprocessed in time $t(n)$ with $O(n)$ operations, yielding query time $O(t(n))$, then the corresponding prefix product problem can be solved in time $O(t(n))$ with $O(n)$ operations.*

Proof. First preprocess the input for range queries. Then partition the input elements into $O(n/t(n))$ groups of size $O(t(n))$ each, compute the prefix products for the first elements of all groups by means of range queries, which needs

$O(t(n))$ time and $O(n)$ operations, and then compute all remaining prefix products sequentially within each group, which also needs $O(t(n))$ time and $O(n)$ operations.

Theorems 3, 5, 6, 12(b) and, for constant c, Theorems 9 and 10 imply that each of the respective problems can be solved in $O(\alpha(n))$ time using $O(n(\alpha(n))^q)$ operations, for some fixed q. Theorem 12(a) implies that the problem can be solved in $O(\log^* n)$ time using $O(n \log^* n)$ processors. Applying Propositions 13 and 14, we obtain

Corollary 15. *The problems of Theorems 3, 5, 6, 12(b) and, for constant c, Theorems 9 and 10 can all be solved optimally in $O(\alpha(n))$ time; for the range product problems, both the preprocessing time and the sequential query time is $O(\alpha(n))$. The problem of Theorem 12(a) can be solved optimally in $O(\log^* n)$ time, yielding constant sequential query time.*

References

1. N. Alon and N. Megiddo, Parallel Linear Programming in Fixed Dimension Almost Surely in Constant Time, *J. Assoc. Comput. Mach.* **41** (1994), pp. 422–434.
2. N. Alon and B. Schieber, Optimal Preprocessing for Answering On-line Product Queries, Tech. Rep. No. 71/87, Tel-Aviv University, 1987.
3. O. Berkman, Y. Matias and U. Vishkin, Randomized Range-Maxima in Nearly-Constant Parallel Time, *Comput. Complexity* **2** (1992), pp. 350–373.
4. O. Berkman and U. Vishkin, Recursive Star-Tree Parallel Data Structure, *SIAM J. Comput.* **22** (1993), pp. 221–242.
5. O. Berkman and U. Vishkin, On Parallel Integer Merging, *Inform. and Computation* **106** (1993), pp. 266–285.
6. A. K. Chandra, S. Fortune and R. Lipton, Lower Bounds for Constant Depth Circuits for Prefix Problems, Proc. 10th International Colloquium on Automata, Languages and Programming (1983), Springer Lecture Notes in Computer Science, Vol. 154, pp. 109–117.
7. A. K. Chandra, S. Fortune and R. Lipton, Unbounded Fan-in Circuits and Associative Functions, *J. Comput. Syst. Sci.* **30** (1985), pp. 222–234.
8. D. Eppstein and Z. Galil, Parallel Algorithmic Techniques for Combinatorial Computation, *Ann. Rev. Comput. Sci.* **3** (1988), pp. 233–283.
9. T. Hagerup, The Log-Star Revolution, Proc., 9th Annual Symposium on Theoretical Aspects of Computer Science (1992), Springer Lecture Notes in Computer Science, Vol. 577, pp. 259–278.
10. R. E. Ladner and M. J. Fischer, Parallel Prefix Computation, *J. Assoc. Comput. Mach.* **27** (1980), pp. 831–838.
11. P. Ragde, The Parallel Simplicity of Compaction and Chaining. *J. Alg.* **14** (1993), pp. 371–380.
12. P. Raghavan, Probabilistic Construction of Deterministic Algorithms: Approximating Packing Integer Programs, *J. Comput. Syst. Sci.* **37** (1988), pp. 130–143.
13. L. G. Valiant, Parallelism in Comparison Problems, *SIAM J. Comput.* **4** (1975), pp. 348–355.

This article was processed using the LaTeX macro package with LLNCS style