Parallel Processing Letters Vol. 4 No. 4 (1994) 455–463

EFFICIENT PARALLEL SHUFFLE RECOGNITION

M. NIVAT
L.I.T.P., Universite Paris VII, 2
place Jussieu, 75251 Paris Cedex 05, France

G. D. S. RAMKUMAR*
Robotics Laboratory
Department of Computer Science
Stanford University, Stanford, CA 94305, USA
E-mail: ramkumar@cs.stanford.edu

C. PANDU RANGAN
Department of Computer Science and Engineering
Indian Institute of Technology, Madras-600036, India
E-mail: rangan@iitm.ernet.in

A. SAOUDI†
L.I.N.P., Universite Paris XIII, Institut Galilée
93430 Villetaneuse, France

R. SUNDARAM‡
Laboratory for Computer Science, NE 43-372
Massachusetts Institute of Technology
Cambridge, MA 02139, USA
E-mail: koods@theory.lcs.mit.edu

Received 22 February 1993
Revised 14 September 1994
Accepted by P. Quinton

ABSTRACT

This paper presents a parallel algorithm for verifying that a string X is formed by the shuffle of two strings Y and Z. The algorithm runs in $O(\log^2 n)$ time with $O(n^2/\log^2 n)$ processors on the EREW-PRAM model.

Keywords: Strings, parallel algorithms, systolic algorithms, digraph reachability.

*The research of this author was carried out while he was at the Indian Institute of Technology, Madras, India.
†Passed away on 11 August 1993.
‡The research of this author was carried out while he was at the Indian Institute of Technology, Madras, India.

1. Introduction

Let $Y = y_1, y_2, \ldots, y_n$ and $Z = z_1, z_2, \ldots, z_n$ be two strings of length n over a finite alphabet Σ. The *string shuffle problem* is to check if a third string $X = x_1, x_2, \ldots, x_{2n}$ of length $2n$ is the shuffle of Y and Z, i.e. whether X is formed by arbitrarily interleaving the characters of Y and Z, while maintaining the original order of the characters in the strings Y and Z.

This problem arises in the context of studies relating to the serialization principle for concurrent programming, where arbitrary interleaving of independent processes will be considered for an analysis of synchronous execution [4,8].

The string shuffle problem, also referred to as the *merge recognition problem*, has a simple sequential $O(n^2)$ algorithm based on dynamic programming [11]. A faster algorithm with $O(n^2/\log n)$ sequential complexity is presented in [10]. An optimal systolic algorithm running in $O(n)$ time with $O(n)$ processors is rather straightforward from the dynamic programming formulation of the problem. What remained as a challenging problem was the design of efficient parallel algorithms in *NC* (i.e. the class of parallel algorithms running in poly-log time using polynomial number of processors).

In [7] it is shown that the computations performed in an arbitrary dynamic programming scheme can be parallelized in poly-log time. However, the generalized schema results in a method employing $O(n^6)$ processors and running in $O(\log^2 n)$ time. Thus, applying this method, one immediately arrives at a poly-log solution to the problem, but with an extremely large processor complexity.

This problem can also be equivalently formulated as a digraph reachability problem. Consider a directed graph G with $(n+1)^2$ nodes, with a typical node denoted as $N(i, j)$, $0 \leq i, j \leq n$. Draw a directed arc from $N(i-1, j)$ to $N(i, j)$ iff $x_{i+j} = y_i$, $1 \leq i \leq n$, $0 \leq j \leq n$, and from $N(i, j-1)$ to $N(i, j)$ iff $x_{i+j} = z_j$, $0 \leq i \leq n$, $1 \leq j \leq n$. It is easy to see that the string shuffle problem is equivalent to checking whether $N(n, n)$ is reachable from $N(0, 0)$ in G. The string shuffle problem is therefore reducible to the reachability problem between a pair of vertices. In general, this problem is as complex as the transitive closure problem. For finding the transitive closure, the best algorithm in the EREW-PRAM model takes $O(\log^2 n)$ time and $M(n)$ processors, where $M(n)$ is the number of processors required to multiply two boolean matrices of order n, in $O(\log n)$ time, in the CRCW-PRAM model. At present $M(n)$ is $O(n^{2.376})$ [12]. It is easy to see that the directed graph G is planar, and hence the results of [9] can be applied immediately. Since G has $O(n^2)$ vertices, applying the result of [9] leads to an $O(\log^3 n)$ time algorithm with $O(n^2)$ processors in the CRCW-PRAM model. In [2], Apostolico *et al.* design an efficient parallel algorithm for the *string editing problem*. Independently, Aggarwal and Park have, in [1], designed efficient algorithms for the same, both in the CREW-PRAM and CRCW-PRAM models. The string shuffle problem can be viewed as a special case of the string editing problem or the grid graph path problem, in the terminology

of [2]. Their best known algorithms [1,2] take $O(\log^2 n)$ time and $O(n^2/\log n)$ processors in the CREW-PRAM model. We improve the processor complexity by a factor of $\log n$ using the weaker EREW rather than the CREW model.

In this paper we propose a parallel algorithm using only $O(n^2/\log^2 n)$ processors and running in $O(\log^2 n)$ time in the EREW-PRAM model. We exploit certain structural properties of the graph to arrive at a more efficient solution, but avoid the use of more complex subroutines such as parallel merge sort [5], cascading divide and conquer [3], or optimal list ranking [6]. Our method can be extended to any problem which admits a dynamic programming solution whose computational pattern resembles the pattern in the string shuffle problem.

2. Definitions and Results

The nodes of the graph defined above are imagined to be placed in the form of a matrix (or mesh). The *row index* of a node $N(i, j)$ refers to i while the *column index* refers to j. References to nodes in the column c denote nodes of the form $N(i, c)$, $0 \leq i \leq n$, while references to nodes in the row r denote nodes of the form $N(r, j)$, $0 \leq j \leq n$. Let $S(c_1, c_2)$ denote the ordered sequence of nodes (in ascending order of row index) in the c_1^{th} column from which some node in the c_2^{th} column is reachable, $0 \leq c_1 \leq c_2 \leq n$. $S(c_1, c_2)$ may be explicitly represented by the sequence of row indices of the corresponding nodes, stored in ascending order. Let $F(c_1, i, c_2)$ denote the sequence of nodes, ordered in ascending order of row index, in the c_1^{th} column from which the node $N(i, c_2)$ is reachable, $0 \leq c_1 \leq c_2 \leq n$, $0 \leq i \leq n$. The following lemma states an important property of $F(c_1, i, c_2)$.

Lemma 1. *Let* $N(r_1, c_1), N(r_2, c_1) \in F(c_1, i, c_2)$, $0 \leq c_1 < c_2 \leq n$, $0 \leq r_1 < r_2 \leq n$, $0 \leq i \leq n$. *Let* r' *be an integer such that*

- $r_1 \leq r' \leq r_2$ *and*
- $N(r', c_1) \in S(c_1, c_2)$.

Then $N(r', c_1) \in F(c_1, i, c_2)$.

Proof. Straightforward. □

Since $F(c_1, i, c_2)$ forms a contiguous subsequence of $S(c_1, c_2)$, it is sufficient to maintain only two end values to represent it. Specifically, let $h(c_1, i, c_2)$ denote the smallest row index and $l(c_1, i, c_2)$ the largest row index amongst the row indices of the nodes in $F(c_1, i, c_2)$. Then we may represent $F(c_1, i, c_2)$ by the pair $(h(c_1, i, c_2), l(c_1, i, c_2))$ with the understanding that every node in $S(c_1, c_2)$, with row index in the range $(h(c_1, i, c_2), l(c_1, i, c_2))$, will be a member of $F(c_1, i, c_2)$. Note that $F(c_1, i, c_2)$ may be empty if there is no node in the c_1^{th} column from which there is a path to $N(i, c_2)$. In this case, $F(c_1, i, c_2)$ may be represented as (ϕ, ϕ) where ϕ represents a dummy *undefined* value.

Lemma 2. *Let* $F(c_1, r_1, c_2)$ *and* $F(c_1, r_2, c_2)$ *be non-empty,* $0 \leq c_1 < c_2 \leq n$, $0 \leq r_1 < r_2 \leq n$. *Then*

- $h(c_1, r_1, c_2) \leq h(c_1, r_2, c_2)$ *and*
- $l(c_1, r_1, c_2) \leq l(c_1, r_2, c_2)$.

Proof. Straightforward. □

The above lemma implies that the sequences $F(c_1, i, c_2), 0 \leq i \leq n$, considered as intervals of the sequence $S(c_1, c_2)$, $0 \leq c_1 < c_2 \leq n$, are *monotonic*, in the sense that the lower endpoints (and similarly the higher endpoints) are sequenced in non-decreasing order with respect to i.

The primary technique employed by the algorithm is recursive doubling. Let $D[c_1, c_2]$, $0 \leq c_1 < c_2 \leq n$, denote the set of data : $S(c_1, c_2)$ and $F(c_1, i, c_2)$, for all i, $0 \leq i \leq n$. In order to handle computations arising at various stages we show how the following computations are done.

1. Construction of the directed grid graph G.
2. Computation of $D[c, c]$, for a given c, $0 \leq c \leq n$.
3. Given $D[c, c]$ and $D[c+1, c+1]$, the computation of $D[c, c+1]$, $0 \leq c < n$.
4. Given $D[c_1, c_2]$ and $D[c_2, c_3]$, the computation of $D[c_1, c_3]$, $0 \leq c_1 < c_2 < c_3 \leq n$.

Let $V = \{v_1, v_2, \ldots, v_n\}$ be an ordered sequence of values, some of which may be undefined. Define a set of intervals each defining a subrange of $[1..n]$. That is, let $[left_i, right_i]$ denote the contiguous set of integers in the range from $left_i$ to $right_i$, where $1 \leq i \leq q \leq n$, and $1 \leq left_i \leq right_i \leq n$. A set of q intervals as described above is said to be *monotonically ordered* if $left_i \leq left_{i+1}$ and $right_i \leq right_{i+1}$ for all $1 \leq i \leq q-1$. Let $Min([left_i, right_i])$ denote the least of the defined v_j's such that $left_i \leq j \leq right_i$. Note that $Min([left_i, right_i])$ may not be defined and we set $Min([\,]) = \phi$ in such a situation.

Lemma 3. *Given an array $V = [v_1, v_2, \ldots, v_n]$ of values and a set Q of q monotonically ordered intervals $[left_i, right_i]$ where $1 \leq i \leq q \leq n$, $Min([left_i, right_i])$ can be computed over all i, $1 \leq i \leq q$ in $O(\log n)$ time using $O(n/\log n)$ processors in the EREW-PRAM model.*

Proof. For each i, $1 \leq i \leq n$, let $nearest\_def(i)$ denote the least v_j such that $i \leq j \leq n$, and v_j is defined. Note that $nearest\_def(i)$ may be undefined. Correspondingly, let $index\_nearest\_def(i)$ denote the least j, $i \leq j \leq n$, such that v_j is defined. Note that $index\_nearest\_def(i)$ too may be undefined. If all concerned quantities are defined then obviously $v_{index\_nearest\_def(i)} = nearest\_def(i)$. Compute $nearest\_def(i)$ and $index\_nearest\_def(i)$ for all i, $1 \leq i \leq n$. This can be done easily in $O(\log n)$ time using $O(n/\log n)$ processors in the EREW-PRAM model. Now, if $index\_nearest\_def(left_i) \leq right_i$ then $Min([left_i, right_i])$ $= nearest\_def(left_i)$, otherwise $Min([left_i, right_i]) = \phi$ (undefined). But, the left endpoints of more than one interval could be the same. Since only exclusive reads are permitted, it is necessary to have as many copies of $nearest\_def(i)$ and $index\_nearest\_def(i)$ as there are intervals with i as their left endpoints. Since the

total number of copies required is at most $q \leq n$, this duplication can be easily done in $O(\log n)$ time with $O(n/\log n)$ processors in the EREW-PRAM model. Once sufficient number of copies are available, it is a simple matter to be able to compute $Min([left_i, right_i])$ for all i, $1 \leq i \leq q$, using only exclusive reads in $O(\log n)$ time with $O(n/\log n)$ processors. □

We shall now state a general form of the lemma proved above that will suit our purposes.

Lemma 4. *Let O be an ordered sequence consisting of p elements $[1..p]$, $1 \leq p \leq n$. Associated with each element i of the sequence let there be a value v_i, $1 \leq i \leq p$, such that, though some of the v_i's may be undefined (represented by ϕ), for some j, k, $1 \leq j < k \leq p$, if both v_j and v_k are defined, then $v_j \leq v_k$. Given a set of q, $1 \leq q \leq n$, monotonically ordered closed intervals $[left_i, right_i]$ $1 \leq i \leq q$, where $1 \leq left_i \leq right_i \leq p$, $Min([left_i, right_i])$, for all i, can be computed in $O(\log n)$ time using $O(n/\log n)$ processors in the EREW-PRAM model.*

Lemma 5. *Let V and Q be as defined in Lemma 3. The complement of the intervals in Q, i.e. the set $\{j | 1 \leq j \leq n, j$ is not in $[left_i, right_i]$ for any i, $1 \leq i \leq q\}$ can be computed in $O(\log n)$ time using $O(n/\log n)$ processors in the EREW-PRAM model.*

Proof. Let $max\_right(j)$ be the maximum of $right_i$ over all intervals with $left_i = j$. The problem of computing $max\_right(j)$ for all j, $1 \leq j \leq n$, using only exclusive reads is easily done in $O(\log n)$ time using $O(n/\log n)$ processors in the EREW-PRAM model, since the intervals are sorted with respect to left endpoints and the intervals with the same left endpoints are sorted with respect to right endpoints. Let $max(j)$ =maximum$\{max_right(i) | i \leq j\}$. This is a standard prefix sums computation. Then, the required set is nothing but $\{i | max(i) \leq i\}$. Hence, the complement of a set of monotonically ordered intervals can be computed in $O(\log n)$ time using $O(n/\log n)$ processors in the EREW-PRAM model. □

As before we state a generalized form of the lemma proved above that will suit our purpose.

Lemma 6. *Let O, V, Q, q be as defined in Lemmas 3 and 4. The complement of these intervals, i.e. the set $\{j | 1 \leq j \leq p$, j is not in $[left_i, right_i]$ for any i, $1 \leq i \leq q\}$ can be computed in $O(\log n)$ time using $O(n/\log n)$ processors in the EREW-PRAM model.*

Construction of the graph. For the construction of the graph, we have to perform comparisons of the type $x_{i+j} = y_i$ or $x_{i+j} = z_j$, $0 \leq i, j \leq n$. To read these entries in an exclusive way we need to have up to n copies of Y and Z and $2n$ copies of X. By repeatedly doubling the entries, i.e. making duplicate copies, it is possible to create the required number of copies in $O(\log^2 n)$ time by using one processor for every $\log n$ characters. Thus, by using $O(n^2/\log^2 n)$ processors it is possible to create a sufficient number of copies of X, Y and Z in $O(\log^2 n)$ time.

Then the comparisons can be performed simply in $O(\log^2 n)$ time with $O(n^2/\log^2 n)$ processors. In fact, a much stronger result is possible. But the following weaker result is sufficient for us.

Proposition 1. *Given $X \in \Sigma^{2n}$, Y, $Z \in \Sigma^n$, we can construct the graph G, in $O(\log^2 n)$ time with $O(n^2/\log^2 n)$ processors in the EREW-PRAM model.* □

Computation of $D[c, c]$, for a given c, $0 \leq c \leq n$. $S(c, c)$ is trivially the sequence $\{0, 1, \ldots, n\}$ and $F(c, i, c)$,$0 \leq i \leq n$, is the set of all nodes in the c^{th} column from which there is a directed path (chain) to $N(i, c)$. Thus $l(c, i, c) = i$ and $h(c, i, c) = i - t_i$, where t_i is the length of the longest chain ending at node $N(i, c)$ in the c^{th} column. By considering the outdegree sequences and performing standard computations such as prefix sum and list ranking [7], one can find t_i for all i, $0 \leq i \leq n$, in linear sequential time and $O(\log n)$ parallel time with $O(n/\log n)$ processors in the EREW-PRAM model. Hence we get

Proposition 2. *For a given c, $0 \leq c \leq n$, $D[c, c]$ can be computed in $O(\log n)$ time in EREW-PRAM using $O(n/\log n)$ processors. For all c, $0 \leq c \leq n$, $D[c, c]$ can be computed in $O(\log^2 n)$ time using $O(n^2/\log^2 n)$ processors.* □

Given $D[c, c]$ and $D[c+1, c+1]$, the computation of $D[c, c+1]$, $0 \leq c_1 < c_2 \leq n$. To compute $D[c, c+1]$, it is necessary to compute $S(c, c+1)$ and $h(c, i, c+1)$, $l(c, i, c+1)$, for all i, $0 \leq i \leq n$. First, we show how to compute $h(c, i, c+1)$, for all i, $0 \leq i \leq n$, (the computation of $l(c, i, c+1)$ being similar, its description is omitted) and then once $h(c, i, c+1)$ and $l(c, i, c+1)$ have been computed for all i, $0 \leq i \leq n$, we show how to compute $S(c, c+1)$.

(a) Computation of $h(c, i, c+1)$, for all i, $0 \leq i \leq n$. If there exists a directed edge from $N(i, c)$ to $N(i, c+1)$ then let

$$h'(c, i, c+1) = \text{Minimum}\{h(c, j, c) | 0 \leq j \leq n \wedge N(j, c) \in F(c, i, c)\} \quad (1)$$

else let $h'(c, i, c+1) = \phi$. Note that $h'(c, i, c+1)$ may be undefined for certain values. Intuitively, $h'(c, i, c+1)$ is the lowest row index of a node in the c^{th} column from which the node $N(i, c+1)$ is reachable by a path which is internally vertex disjoint to the $c+1^{st}$ column. Using Eq.(1) and Lemma 4, the computation of $h'(c, i, c+1)$, for all i, $0 \leq i \leq n$, can be done in $O(\log n)$ time with $O(n/\log n)$ processors in the EREW-PRAM model. To make this clearer we explain the correspondence between the parameters here and those in Lemma 4. The sequence O corresponds to the set $S(c, c)$, the values v_i correspond to $h(c, i, c)$. And, the monotonically closed ordered intervals $[left_i, right_i]$ correspond to $F(c, i, c)$, for those i such that there exists a directed edge from $N(i, c)$ to $N(i, c+1)$. It is easily seen that:

$$h(c, i, c+1) = \text{Minimum}\{h'(c, j, c+1) | 0 \leq j \leq n \wedge N(j, c+1) \in F(c+1, i, c+1)\} \quad (2)$$

Notice again that $h'(c, i, c+1)$ may be undefined. The computation of $h(c, i, c+1)$, for all i, $0 \leq i \leq n$, is done in two phases, first $h'(c, i, c+1)$, for all i,

$0 \leq i \leq n$, is computed using Eq. (1) and Lemma 4 as described above, and then $h(c, i, c+1)$, for all i, $0 \leq i \leq n$, is computed using Eq. (2) and Lemma 4 in $O(\log n)$ time with $O(n/\log n)$ processors in the EREW-PRAM model. Note that the sequence $h'(c, i, c+1)$ satisfies the property that, though some of the $h'(c, i, c+1)$'s may be undefined, for some j, k, $1 \leq j < k \leq n$, if both $h(c, j, c+1)$ and $h(c, k, c+1)$ are defined, then $h(c, j, c+1) \leq h(c, k, c+1)$. We make the application of Lemma 4 at this stage clearer, by showing the correspondence between the parameters here and those in Lemma 4. The sequence O corresponds to the set $S(c+1, c+1)$, the values v_i correspond to $h'(c, i, c+1)$. And, the monotonically closed ordered intervals $[left_i, right_i]$ correspond to $F(c+1, i, c+1)$.

(b) Computation of $S(c_1, c_2+1)$. At this stage, we already have $h(c, i, c+1)$, $l(c, i, c+1)$, for all i, $0 \leq i \leq n$. If we eliminate from $S(c, c)$ all those elements not contained in any of the intervals $[h(c, i, c+1), l(c, i, c+1)]$, for all i, $0 \leq i \leq n$, then we get $S(c, c+1)$. All that remains is the computation of the complement of the set of intervals, since, once it is computed, it can easily be eliminated from $S(c, c)$ to generate $S(c, c+1)$. Using Lemma 6, this can be done in $O(\log n)$ time with $O(n/\log n)$ processors in the EREW-PRAM model. The correspondence between the parameters here and those in Lemma 6 are as follows: the sequence O corresponds to $S(c, c)$, the monotonically closed ordered intervals $[left_i, right_i]$ correspond to $F(c, i, c+1)$.

Proposition 3. *For a given c, $0 \leq c < n$, using $D[c, c]$ and $D[c+1, c+1]$ we can compute $D[c, c+1]$ in $O(\log n)$ time using $O(n/\log n)$ processors in the EREW-PRAM model.* □

Given $D[c_1, c_2]$ and $D[c_2, c_3]$ the computation of $D[c_1, c_3]$, $0 \leq c_1 < c_2 < c_3 \leq n$. To compute $D[c_1, c_3]$, it is necessary to compute $S(c_1, c_3)$ and $h(c_1, i, c_3)$, $l(c_1, i, c_3)$, for all i, $0 \leq i \leq n$. First, we show how to compute $h(c_1, i, c_3)$, for all i, $0 \leq i \leq n$, (the computation of $l(c_1, i, c_3)$ being similar, its description is omitted) and then once $h(c_1, i, c_3)$ and $l(c_1, i, c_3)$ have been computed for all i, $0 \leq i \leq n$, we show how to compute $S(c_1, c_3)$.

(a) Computation of $h(c_1, i, c_3)$, for all i, $0 \leq i \leq n$. It is easily seen that:

$$h(c_1, i, c_3) = \text{Minimum}\{h(c_1, j, c_2) | 0 \leq j \leq n \wedge N(j, c_2) \in F(c_2, i, c_3)\} \quad (3)$$

Using Eq. (3) and Lemma 4, the computation of $h(c_1, i, c_3)$, for all i, $0 \leq i \leq n$, can be done in $O(\log n)$ time with $O(n/\log n)$ processors in the EREW-PRAM model. To make this clearer we explain the correspondence between the parameters here and those in Lemma 4. The sequence O corresponds to the set $S(c_2, c_3)$, the values v_i correspond to $h(c_1, i, c_2)$, for those i, for which $N(j, c_2) \in S(c_2, c_3)$. And, the monotonically closed ordered intervals $[left_i, right_i]$ correspond to $F(c_2, i, c_3)$.

(b) Computation of $S(c_1, c_3)$. At this stage, we already have $h(c_1, i, c_3)$, $l(c_1, i, c_3)$, for all i, $0 \leq i \leq n$. If we eliminate from $S(c_1, c_2)$ all those elements not contained in any of the intervals $[h(c_1, i, c_3), l(c_1, i, c_3)]$, for all i, $0 \leq i \leq n$, then we get $S(c_1, c_3)$. All that remains is the computation of the complement of the set of intervals, since, once it is computed, it can easily be eliminated from $S(c_1, c_2)$ to generate $S(c_1, c_3)$. Using Lemma 6, this can be done in $O(\log n)$ time with $O(n/\log n)$ processors in the EREW-PRAM model. The correspondence between the parameters here and those in Lemma 6 are as follows: the sequence O corresponds to $S(c_1, c_2)$, the monotonically closed ordered intervals $[left_i, right_i]$ correspond to $F(c_1, i, c_3)$.

Proposition 4. *For a given c_1, c_2 and c_3, $0 \leq c_1 < c_2 < c_3 \leq n$, using $D[c_1, c_2]$ and $D[c_2, c_3]$, we can compute $D[c_1, c_3]$ in $O(\log n)$ time using $O(n/\log n)$ processors in the EREW-PRAM model.* □

3. Parallel Algorithm

Informally, the algorithm first constructs the graph and computes $D[c, c]$ for all c such that $0 \leq c \leq n$. It then computes $D[0, \log n]$, $D[\log n, 2\log n], \ldots, D[\lfloor n/\log n \rfloor \log n, n]$ by repeated application of Propositions 3 and 4. Finally, it uses recursive doubling in conjunction with application of Propositions 3 and 4 to compute $D[0, n]$, and checks whether $N(0, 0) \in F(0, n, n)$.

Input → $X \in \Sigma^{2n}$, $Y, Z \in \Sigma^n$.

Steps →

1. Construct the required directed graph.
2. Compute $D[c, c]$, for all c, $0 \leq c \leq n$.
3. Compute $D[c, c+1]$, for all c, $0 \leq c \leq n-1$.
4. **For** $i = 1$ **to** $\lfloor n/\log n \rfloor$ **pardo**
   For $j = 0$ **to** $\log n - 2$ **do**
   use $D[i\log n, i\log n + j + 1]$ **and**
   $D[i\log n + j + 1, i\log n + j + 2]$ **to compute**
   $D[i\log n, i\log n + j + 2]$.
   end.

   parend.
5. **For** $i = 1$ **to** $\lceil \log(\lfloor n/\log n \rfloor) \rceil$ **do**
   For $j = 0$ **to** $\lfloor \lfloor (n/\log n) \rfloor / 2^i \rfloor$ **pardo**
   use $D[2j2^{i-1}\log n, (2j+1)2^{i-1}\log n]$ **and**
   $D[(2j+1)2^{i-1}\log n, (2j+2)2^{i-1}\log n]$ **to compute**
   $D[j2^i \log n, (j+1)2^i \log n]$.
   parend.

   end.
6. Output YES if $h(0, n, n) = 0$, else output NO.

Output → YES if X is a shuffle of Y and Z, else NO.

Theorem 1. *The string shuffle problem can be solved in* $O(\log^2 n)$ *time using* $O(n^2/\log^2 n)$ *processors in the* $EREW - PRAM$ *model.*

Proof. Follows from Propositions 1, 2, 3 and 4. □

4. Conclusion

We have been able to optimally decompose the dynamic programming based solution (which has $O(n^2)$ sequential complexity) in $O(\log^2 n)$ time with $O(n^2/\log^2 n)$ processors in the EREW-PRAM model by avoiding the transitive closure bottleneck. We did this by using certain special properties of the underlying graph to solve the corresponding path problem efficiently. However, the best known sequential algorithm has complexity $O(n^2/\log n)$. It remains to be seen if we can solve the string shuffle problem (also known as the merge recognition problem) with the processor time product matching the best sequential bound. It also remains a tantalizing open problem to establish a non-trivial lower bound for this problem (both sequential as well as parallel), as the gap between the algorithmic complexity (namely $O(n^2/\log n)$ and the trivial lower bound ($O(n)$) is pretty wide.

References

[1] A. Aggarwal and J. Park, Notes on searching in multidimensional monotone arrays, *29th Annual IEEE Foundations of Computer Science*, 1988, 497–512.

[2] A. Apostolico, M. Atallah, L. L. Larmore and S. McFaddin, Efficient parallel algorithms for string editing and related problems, *SIAM J. Computing* **19**, 5 (1990), 968–988.

[3] M. Atallah, R. Cole and M. T. Goodrich, Cascading divide and conquer: a technique for designing parallel algorithms, *28th Annual IEEE Foundations of Computer Science*, 1987, 151–160.

[4] M. Ben Ari, *Principles of Concurrent Programming* (Prentice Hall, 1982).

[5] R. Cole, Parallel merge sort, *27th Annual IEEE Foundations of Computer Science*, 1986, 511–516.

[6] R. Cole and U. Vishkin, Deterministic coin tossing and accelerating cascades: Micro and macro techniques for designing parallel algorithms, *18th Annual ACM Symposium on Theory of Computing*, 1986, 206–219.

[7] A. Gibbons and W. Rytter, *Efficient parallel algorithms* (Cambridge University Press, 1988).

[8] C. A. R. Hoare, *Communicating Sequential Processes* (Prentice Hall, 1985).

[9] M. Y. Kao and P. Klein, Towards overcoming the transitive-closure bottleneck: efficient parallel algorithms for planar digraphs, *22nd Symp. Theory of Computing*, 1990, 181–192.

[10] J. V. Leeuwen and M. Nivat, Efficient recognition of rational relations, *Information Processing Letters* **14** (1982) 34–38.

[11] A. Mansfield, An algorithm for a merge recognition problem, *Discrete Applied Mathematics* **4** (1982) 193–197.

[12] V. Pan and J. H. Reif, Fast and efficient parallel solution of linear systems, *SIAM J. Computing* **22**, 6 (1993) 1227–1250.