

# An Efficient Parallel Algorithm for Building the Separating Tree\*

Yijie Han

School of Computing and Engineering  
University of Missouri at Kansas City  
Kansas City, MO 64110, USA  
hanyij@umkc.edu

## Abstract

We present an efficient parallel algorithm for building the separating tree for a separable permutation. Our algorithm runs in  $O(\log^2 n)$  time using  $O(n \log n)$  operations on the CREW PRAM and  $O(\log n \log \log n)$  time using  $O(n \log n)$  operations on the CRCW PRAM.

*Keywords:* Algorithms, parallel algorithms, separable permutation.

## 1 Introduction

In the pattern matching problem for permutations a permutation  $T = (t_1, t_2, \dots, t_n)$  of  $1, 2, \dots, n$  is given as *text*. Another permutation  $P = (p_1, p_2, \dots, p_k)$  of  $1, 2, \dots, k$ ,  $k \leq n$  is given as *pattern*. We want to find whether there is a length  $k$  subsequence  $T'$  of  $T$ , where  $T' = (t_{i_1}, t_{i_2}, \dots, t_{i_k})$  with  $i_1 < i_2 < \dots < i_k$ , such that the elements of  $T'$  are ordered according to the permutation of  $P$ , i.e.,  $t_{i_r} < t_{i_s}$  iff  $p_r < p_s$ . This problem was proposed by Wilf[2, 4].

If  $T'$  exists, then we say that  $T$  contains  $P$  or  $P$  matches into  $T$ . When  $P = (1, 2, \dots, k)$ , then we have to find an increasing subsequence of length  $k$  in  $T$ . The general problem is known to be NP-complete[2]. However, when  $P$  is separable, i.e.  $P$  contains neither the sub-pattern  $(3, 1, 4, 2)$  nor its reverse, the sub-pattern  $(2, 4, 1, 3)$ , polynomial time algorithms for the decision and counting problems (decide whether  $T'$  exists and count the number of matches of  $P$  into  $T$ ) exist[2, 4].

Separable permutations can be sorted in a particular way. Imagine a straight segment of railway track, with the elements of the permutation lined up on the track. Imagine that a segment in the middle of the track can be rotated  $180^\circ$  and reversing the order of the elements on the middle segment. We may move any consecutive subsequence of the elements onto the middle of the track and reverse their order. There is one restriction: a sequence of elements to be rotated

on the middle section of the track must be first coupled together and remain coupled forever afterwards. Separable permutations are exactly the permutations that can be sorted using this method. The linear time algorithm [2] for sorting separable permutations also provides a test if a permutation is separable. It works as follows. Use a stack  $S$  whose elements are contiguous subranges  $l, l+1, \dots, l+m$  of the range  $1, \dots, k$ . To add a contiguous range  $r$  to  $S$ , check if the top element of  $S$  forms a larger contiguous range together with  $r$ , i.e. the union of the two sets of numbers is a contiguous range. If so, pop the top element of  $S$ , form a combined contiguous range and adding that contiguous range to  $S$ , as just described. If at the end of  $P$  the stack  $S$  contains a single contiguous range then the permutation is separable, otherwise it is not.  $P$ 's separating tree  $T$  can be easily constructed by this algorithm. This is a sequential algorithm takes  $O(k)$  time.

Yugandhar and Saxena gave parallel algorithms for constructing the separating tree[6]. One of their algorithm runs in  $O(\log n)$  time but uses  $O(n^2)$  operations, where operation is defined to be the time processor product. Another parallel algorithm of theirs could be more efficient. It runs in  $O(d \log n)$  time with  $O(nd)$  operations, where  $d$  is the depth of an optimal tree. Their first algorithm is not efficient since it uses  $O(n^2)$  operations compared to the sequential algorithm which has  $O(n)$  operations. Their second algorithm is efficient when  $d$  is small. However, they show that there exists a separable permutation (i.e.  $P = (1, 3, 5, 7, \dots, n-1, n, \dots, 8, 6, 4, 2)$ ) such that the depth of its optimal separating tree is  $\Theta(n)$ .

In this paper we present an efficient parallel algorithm for building the separating tree. Our algorithm runs in  $O(\log^2 n)$  time and  $O(n \log n)$  operations on the CREW(Concurrent Read Exclusive Write) PRAM (Parallel Random Access Machine) model and in  $O(\log n \log \log n)$  time and  $O(n \log n)$  operations on the CRCW(Concurrent Read Concurrent Write) PRAM model.

CREW PRAM and CRCW PRAM are parallel computation models. On the CREW PRAM model all

---

\*Research supported in part by NSF Grant 0310245

processors can read the shared memory concurrently, but only one processor can write into a shared memory cell in one step (although different processors can write into different memory cells concurrently). On the CRCW PRAM model more than one processors can concurrently read from or write into a shared memory cell. In the case of current write we use the COMMON model in which all processors writing into a memory concurrently have to write the same value.

## 2 Constructing the Separating Tree

A separating tree for a separable permutation  $P = (p_1, p_2, \dots, p_k)$  of  $1, 2, \dots, k$  is a binary tree  $T$  with leaves  $(p_1, p_2, \dots, p_k)$  in that order, such that for each node  $v$ , if the leaves of the subtree rooted at  $v$  are  $p_i, p_{i+1}, \dots, p_{i+j}$ , then the set of numbers  $\{p_i, p_{i+1}, \dots, p_{i+j}\}$  is a contiguous subrange of the range  $1, 2, \dots, k$ , i.e. is of the form  $\{l, l+1, \dots, l+m\}$ , for some  $l, m$  with  $1 \leq l \leq k$ ,  $0 \leq m \leq k-l$ . This contiguous subrange is called the range of the node  $v$ . If  $v$  is a node of the tree with left child  $v_l$  and right child  $v_r$  then either the range of  $v_l$  just precedes the range of  $v_r$  or the range of  $v_r$  just precedes the range of  $v_l$ .

**Lemma 1 ([2]):** A pattern  $P = (p_1, p_2, \dots, p_k)$  is separable iff it has a separating tree.

The difficulty in building a separating tree efficiently in parallel is to identify a node  $v$  of the separating tree such that  $v$  has  $k/c$  leaves for a constant  $c$ . Suppose we identified such a node with leaves in the contiguous range  $\{l, l+1, \dots, l+m\}$ , then we can take these leaves out, relabel each leaf  $t$  as  $t-l+1$ . This forms one subproblem for building the separating tree. For the leaves remaining in the original tree we relabel each leaf as follows. If the leaf is less than  $l$ , we need not to relabel it. Since node  $v$  is now a leaf (since we deleted the subtree rooted at  $v$ ), we label it as  $l$ . For each leaf  $t$  greater than  $l+m$  we relabel it as  $t-m$ . This forms the second subproblem for building the separating tree. These two separating tree building subproblems can be solved recursively in parallel. Since  $v$  has  $k/c$  leaves we know the recursion has  $O(\log k)$  levels. If in each recursion level we spend  $O(\log k)$  time ( $O(\log \log k)$  time) and  $O(k)$  operation on CREW PRAM (CRCW PRAM) then the time complexity of our algorithm will be  $O(\log^2 k)$  ( $O(\log k \log \log k)$ ) and the operation complexity will be  $O(k \log k)$ .

In order to identify node  $v$  we need the the following concepts.

The all nearest smaller values problem is defined as follows[1]. Let  $A = (a_1, a_2, \dots, a_n)$  be an array of elements from a totally ordered domain. For each  $a_i$ ,  $1 \leq i \leq n$ , find the nearest element to the left of  $a_i$

and the nearest element to the right of  $a_i$  that are less than  $a_i$ , if such elements exist. That is, for each  $1 \leq i \leq n$ , find the maximum  $1 \leq j < i$ , and the minimum  $i < k \leq n$  such that  $a_j < a_i$  and  $a_k < a_i$ .  $a_j$  is called the left match and  $a_k$  is the right match of  $a_i$ . The all nearest smaller values problem can be solved on the CREW PRAM in  $O(\log n)$  time and  $O(n)$  operations or on the CRCW PRAM in  $O(\log \log n)$  time and  $O(n)$  operations[1].

Given an array of  $n$  real numbers  $A = (a_1, a_2, \dots, a_n)$ , a range minimum (maximum) query requests the minimum (maximum) elements in a subarray  $A_{i,j} = (a_i, \dots, a_j)$ , for some  $1 \leq i \leq j \leq n$ . After  $O(\log n)$  time and  $O(n)$  operation preprocessing on the CREW PRAM or  $O(\log \log n)$  time and  $O(n)$  operation preprocessing on the CRCW PRAM,  $k$  queries can be processed in constant time using  $k$  processors[1].

The Cartesian tree is defined as follows[5]. The Cartesian tree for an array  $A = (a_1, a_2, \dots, a_n)$  of  $n$  distinct real numbers is a binary tree with vertices labeled by the numbers. The root has label  $a_m$ , where  $a_m = \min\{a_1, a_2, \dots, a_n\}$ . Its left subtree is a Cartesian tree for  $A_{1,m-1} = (a_1, \dots, a_{m-1})$ , and its right subtree is a Cartesian tree for  $A_{m+1,n} = (a_{m+1}, \dots, a_n)$ . (The tree for an empty subarray is the empty tree.)

Constructing a Cartesian tree can be done sequentially in linear time[3]. It can also be done in  $O(\log n)$  time with  $O(n)$  operations on the CREW PRAM, or in  $O(\log \log n)$  time with  $O(n)$  operations on the CRCW PRAM[1].

By the way the Cartesian tree is built, the inorder traversal of the Cartesian tree represents the order of the elements in array  $A$ .

Actually we just need the concept of Cartesian tree. We do not need to actually construct a Cartesian tree. For each number  $a_i$  in array  $A$  we need to know  $l$  and  $m$  such that  $a_l, a_{l+1}, \dots, a_m$  are the nodes of the sub-Cartesian tree rooted at  $a_i$ . For this purpose we use an observation in [1]. Let  $a_l$  and  $a_r$  be the left match and right match of  $a_i$  in  $A$ , if such exist. Then the nodes of the subtree rooted at  $a_i$  are  $a_{l+1}, a_{l+2}, \dots, a_{r-1}$ . If  $a_l > a_r$  then  $a_i$  is the right child of  $a_l$ . If  $a_l < a_r$  then  $a_i$  is the left child of  $a_r$ .

For an input separable permutation  $P = (p_1, p_2, \dots, p_k)$ , we first compute, for each  $p_i$  the minimum and maximum range queries for the range  $(p_l, p_{l+1}, \dots, p_r)$ , where  $p_l, p_{l+1}, \dots, p_r$  constitute all the nodes in the sub-Cartesian tree rooted at  $p_i$ . That is,  $p_{l-1}$  is the left match of  $p_i$  and  $p_{r+1}$  is the right match of  $p_i$ , if they exist. This can be done for all  $p_i$  in constant time and  $O(k)$  operations on the CREW PRAM after  $O(\log k)$  time and  $O(k)$  operation preprocessing on the CREW PRAM or  $O(\log \log k)$  time and  $O(k)$  operation preprocessing on the CRCW PRAM. Suppose the range maximum is  $a_M$  and range mini-

mum is  $a_m$ . If  $a_M - a_m = r - l$  then  $p_l, p_{l+1}, \dots, p_r$  form a contiguous range and the sub-Cartesian tree rooted at  $p_i$  can be taken out to form a subproblem for constructing the separating tree. What we wanted is  $k/4 \leq r - l \leq k/2$ . Suppose there exists a  $p_i$  for which  $a_M - a_m = r - l$  and  $k/4 \leq r - l \leq k/2$ . Then we are done at this recursion level since we can take the sub-Cartesian tree rooted at  $p_i$  and form a subproblem for constructing the separating tree. The remaining numbers form another subproblem for constructing the separating tree. Thus we are done at this recursion level and we can continue with the recursion at the next level.

If, however, there is no  $p_i$  such that  $a_M - a_m = r - l$  and  $k/4 \leq r - l \leq k/2$ , then we need more sophisticated solution. We need the following partition lemma.

If nodes  $v_1, v_2, \dots, v_t$  in a tree is contiguous in the inorder traversal of the tree then we say that  $v_1, v_2, \dots, v_t$  is a contiguous section of the tree ( $v_1, v_2, \dots, v_t$  need not form a contiguous range). If nodes  $v_1, v_2, v_3, \dots, v_t$  are descendants of  $v$  and are just preceding  $v$  in the inorder traversal of the tree, then we say that  $v_1, v_2, \dots, v_t$  are  $v$ 's left neighbors. If nodes  $v_1, v_2, v_3, \dots, v_t$  are descendants of  $v$  and are just succeeding  $v$  in the inorder traversal of the tree, then we say that  $v_1, v_2, \dots, v_t$  are  $v$ 's right neighbors.

Refer to Fig. 1. Let  $r$  be the root of a tree  $T$ . Let  $a_{i+1}$  be  $a_i$ 's left child (right child),  $1 \leq i < t$ , and  $a_1$  be the left child (right child) of  $r$  and  $a_t$  has no left child (right child), then we say that  $r, a_1, a_2, \dots, a_t$  is the left dividing line (right dividing line) of  $r$ . Let  $v$  be on a left (right) dividing line of  $w$ , let  $b_{i+1}$  be  $b_i$ 's right child (left child),  $1 \leq i < s$ , and  $b_1$  be the right child (left child) of  $v$  and  $b_s$  has no right child (left child), then we say that  $v, b_1, b_2, \dots, b_s$  is the right dividing line (left dividing line) of  $v$ . Only root can have both left and right dividing line. Internal nodes of the tree can have only either left dividing line or right dividing line.

We have the following partition lemma:

**Lemma 2 (Partition Lemma):** Let  $v$  be a node in the Cartesian tree. Suppose  $v$ 's dividing line is a right (left) dividing line. Let  $c(v)$  be  $v$ 's right (left) child. Then the subtree rooted at  $c(v)$ ,  $v$  and a set  $S$  of  $v$ 's left (right) neighbors form a contiguous range. Besides, nodes in  $S$  partitions nodes in the subtree rooted at  $c(v)$  in the following sense, let  $a_1 < a_2 < \dots < a_t$  are elements in  $S$ , then nodes  $u$  satisfying  $a_i < u < a_{i+1}$  form a contiguous section in the right (left) subtree of  $v$ . If  $u, w$  are nodes in the right (left) subtree of  $v$  and  $a_i < u < a_j < w$  then  $u$  precedes (succeeds)  $w$  in the inorder traversal of the tree and  $a_j$  precedes (succeeds)  $a_i$  in the inorder traversal of the tree.

**Proof:** See Fig. 2. Let  $v$  have both left and right

children. Let  $a_1 < a_2 < \dots < a_t$  be the nodes in the left subtree of  $v$ . If there are  $u, w$  in the right subtree satisfying  $a_i < u < a_{i+1} < w$ , let  $a_k$  be the lowest common ancestor of  $a_i$  and  $a_{i+1}$ , then  $a_k \neq v$  and  $a_{i+1}$  must be in the left subtree rooted at  $a_k$ , for otherwise suppose  $a_{i+1}$  is in the right subtree of  $a_k$ , then we have  $a_k, a_{i+1}, v, u$  forms a sequence of 2, 4, 1, 3, and therefore the permutation is not separable. Thus whenever  $a_i < u < a_j$  then  $a_j$  is in the left subtree of  $v$  which is the lowest common ancestor of  $a_i$  and  $a_j$ . Let  $x$  be the lowest common ancestor of  $u$  and  $w$ , then  $x \neq v$  and  $w$  must be in the right subtree of  $x$ , for otherwise  $a_{i+1}, v, w, x$  forms sequence 3, 1, 4, 2 and therefore the permutation is not separable.

Now let  $v$  be the topmost node in the tree which has both left and right children. The nodes in the subtree rooted at  $v$  form a contiguous range. By the proof above we know that nodes in the left subtree are partitioned into contiguous ranges and nodes in the right subtree are also partitioned into contiguous ranges. Further branches in the tree in each contiguous range again follow our proof above and being further partitioned into contiguous ranges.  $\square$

We obtain a node  $v$  of the Cartesian tree such that the sub-Cartesian tree rooted at  $v$  has  $n(v)$  nodes, where  $k/8 \leq n(v) \leq k/4$ . Let  $p(v)$  have a right dividing line,  $v$  be  $p(v)$ 's right child, where  $p(v)$  is the parent of  $v$ , if such a node  $v$  exists. We sort these  $n(v)$  nodes. Let  $v_1 < v_2 < \dots < v_{n(v)}$ . We say that  $(v_i, v_{i+1})$  is an interval of size  $v_{i+1} - v_i - 1$ .

If there is an interval of size  $s$  satisfying  $s \geq k/8$ , then these  $s$  nodes are in the left subtree of  $p(v)$ . Besides, by the Partition Lemma, these  $s$  nodes form a contiguous section. Since  $v$  has at least  $k/8$  nodes  $s \leq 7k/8$ . Therefore we can take these  $s$  nodes out to form a subproblem and the recursion can continue.

If there is no interval of size  $\geq k/8$  then all intervals are of size  $< k/8$ . In this case we take  $v_1, v_2, \dots, v_i$  such that interval  $(v_i, v_{i+1})$  are of size  $\geq 1$  and  $k/4 \leq v_i - v_1 \leq 3k/4$ . Since we can add a contiguous section alternatively in the left subtree and right subtree of  $p(v)$  and each such contiguous section is of size at most  $k/4$ , therefore such  $v_i$  exists. Interval  $(v_i, v_{i+1})$  of size  $\geq 1$  guarantees that  $v_1, v_2, \dots, v_i$  are in a contiguous section of the tree. Also all nodes  $u$  satisfying  $v_i < u < v_{i+1}$ ,  $1 \leq i < n(v)$  are in a contiguous section in the sibling tree. Besides, all nodes  $w$  satisfying  $v_1 \leq w \leq v_{n(v)}$  form a contiguous section in the tree. And therefore these nodes can be taken out to form a subproblem for the recursion.

The situation where  $p(v)$  has a left dividing line and  $v$  is  $p(v)$ 's left child can be treated similarly.

If such a node  $v$  does not exist, then we end up with the situation that there exists a node  $w$  such that the total number of descendants of nodes on the  $w$ 's

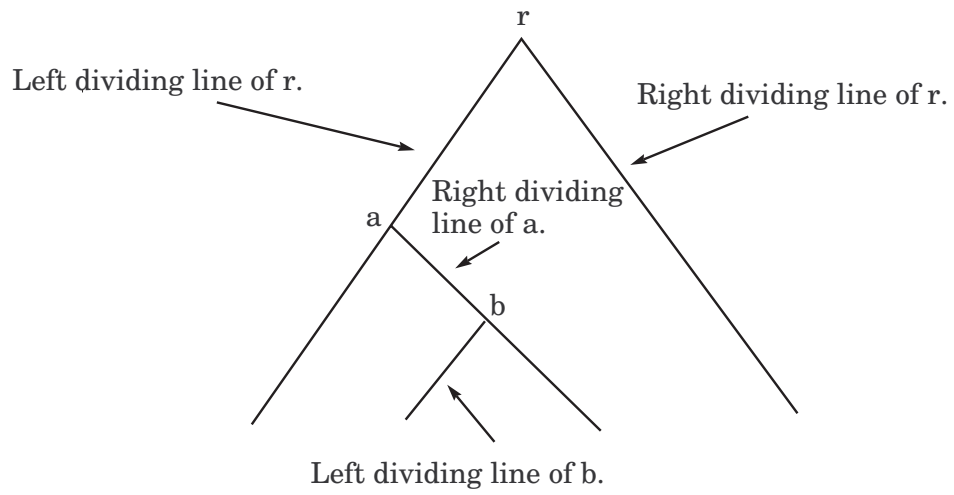


Fig. 1. Dividing lines.

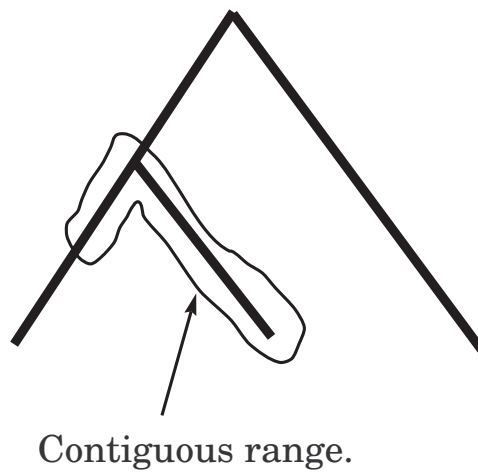


Fig. 2.

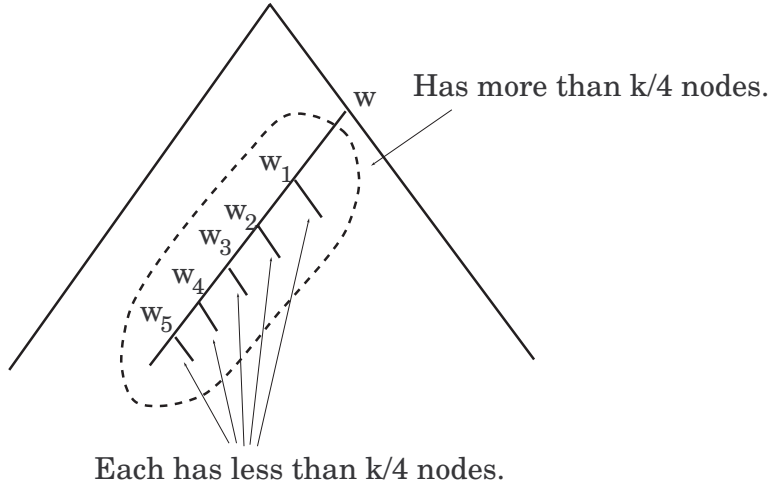


Fig. 3.

dividing line is at least  $n/4$ . Let  $w_1, w_2, \dots, w_t$  be the nodes on the dividing line of  $w$ . The number of nodes on the dividing line of each  $w_i$  and their descendants is  $< n/4$ , as shown in Fig. 3. We sort the subtree  $T_i$  on  $w_i$ 's dividing line (right subtree of  $w_i$  in Fig. 3),  $1 \leq i \leq t$ , separately in parallel. Let  $w_{i,1} < w_{i,2} < \dots < w_{i,a}$  be the nodes in  $T_i$ . If there is an interval  $(w_{i,j}, w_{i,j+1})$  of size  $s$  satisfying  $k/8 \leq s \leq k/4$  then we are done since these  $s$  nodes form a contiguous section in  $T_i$ 's sibling tree. Otherwise there is a contiguous section  $S$  contains contiguous range of nodes with size  $\geq k/4$  but further partition (with the Partition Lemma) has all intervals of size  $< k/4$ . Since each partition is of size  $< k/4$ , by grouping several partitions we can form a contiguous range and section  $T$  of size  $s$  satisfying  $k/4 \leq s \leq k/2$ . This grouping can be done by simply checking  $w_i$ 's. Let  $w_i, w_{i+1}, \dots, w_a$  be the nodes on  $w$ 's dividing line and also in  $T$ . Let  $k/4 \leq w_j - w_i \leq k/2$ , then we take the nodes on the dividing line of  $w_i, w_{i+1}, \dots, w_{j-1}$  and their descendants and form a set  $U$ . We sort set  $U$ . Let  $u_1 < u_2 < \dots < u_b$  be the nodes in  $U$ . Let  $c(w_i)$  be the child of  $w_i$  on the dividing line of  $w_i$ . Let  $c_s(w_i)$  be the sibling of  $c(w_i)$ . We also take nodes  $z$  in the interval  $(u_i, u_{i+1})$ , i.e.  $u_i < z < u_{i+1}$ ,  $1 \leq i < b$ , from the subtrees rooted at  $c_s(w_i)$ 's. This gives us a contiguous section and contiguous range of size  $s$  with  $k/4 \leq s \leq k/2$ . This is the section we will take out to form a subproblem.

**Theorem 1:** A separating tree for a separable permutation of size  $n$  can be computed on the CREW PRAM in  $O(\log^2 n)$  time with  $O(n \log n)$  operations, or on the CRCW PRAM in  $O(\log n \log \log n)$  time with  $O(n \log n)$  operations.

## References

- [1] O. Berkman, B. Schieber, U. Vishkin. Optimal doubly logarithmic parallel algorithms based on finding all nearest smaller values. *Journal of Algorithms* **14**, 344-370(1993).
- [2] P. Bose, J.F. Buss, A. Lubiw. Pattern matching for permutations. *Proc. of the Workshop on Algorithms and Data Structures, Lecture Notes in Computer Science*, 1993, 200-209.
- [3] H.N. Gabow, J.L. Bentley, R.E. Tarjan. Scaling and related techniques for geometry problems. *Proc. 16th Annual ACM Symp. on Theory of Computing*, 1984, 135-143.
- [4] L. Ibarra. Finding pattern matchings for permutations. *Inform. Process. Lett.* 61 (1997) 293-295.
- [5] J. Vuillemin. A unified look at data structures. *Comm. ACM* **23** (1980) 229-239.
- [6] V. Yugandhar, S. Saxena. Parallel algorithms for separable permutations. *Discrete Applied Mathematics*, 146 (2005) 343-364.